

Setting up a dynamic, data-driven website.

by Matthew Webster.

Introduction.

Having now set up a data-driven website for static and interactive content, I have been getting a few questions on how to do this. This document is an effort to answer these questions, hopefully some, possibly all. It will not attempt to describe how you should design your proprietary software, except where basic code for achieving a certain function between server applications is necessary.

This tutorial will not describe how to acquire and install the Java environment as it is assumed the developer reading this already has at least a professional understanding of Java. If not, don't attempt to follow this tutorial as it will not be productive, unless you really only require Apache.

Security is also beyond the scope of this tutorial because of the size of the subject. It is recommended that you implement some form of firewall, of course, which one is up to you. Most developers and site admins will agree that the firewalls built into Windows are not suitable, so if you are connected to the 'net 24/7 I would urge getting decent protection. Appendix C lists some very useful sites, with security and general issues included.

Please bear in mind that I cannot guarantee the end result of the actions carried out by following this tutorial. I will do my best to describe how you should go about installing and configuring a server as I have it, either on one or more machines.

Please also be aware that although many people and organisations use Windows NT as a server operating system, most open source organisations make no guarantee for their software on NT. This means that if you have problems using Apache, for example, on the Windows platform you are essentially on your own. There are, of course, numerous forums and help sites for those using Windows, because it is such a popular system and Apache is probably more reliable and stable on NT than most other server applications.

Errors and omissions should be sent to eudoxus@freeuk.com. Please note the version numbers of the applications and tools used before commenting.

My website contains much information about myself, coding and game playing. If you are looking for other tutorials I have many good links and may well write some myself. These will probably cover coding and Unreal Tournament (map creation etc):

<http://www.matthewwebster.homeunix.net/>

History.

It should be made known that I am a Java Developer by trade, going on 3 years now and have been a member of at least one large website development team so far. Rather than getting mixed up in the server side of things I tended to focus on the software. It is only later on that I started developing this side of my experience and I must say it's coming along quite nicely. I am not an artist, btw.

Current set up.

At the moment I run my whole website, from web server to database, on a single Windows XP box with Sygate Personal Firewall protecting me on a Telewest Blueyonder broadband internet connection. This should soon be moved to a dedicated Windows 2000 Advanced Server box, with development taking place on the existing, glitchy, XP box.

All my static content is served from a single instance of Apache version 2.0.43, with numerous virtual hosts to provide unique sites under a dynamic naming service. Mostly server side

includes are used to provide the content, rather than plain html. Apache connects to Tomcat via the mod_jk2 web connector.

Tomcat runs as one worker instance for the explicit serving of Apache requests via the web connector. It contains a number of small servlets and JSP's in order to provide dynamic and data-driven content. The content is retrieved either from the database or referenced on the static content resource. Servlets run under Tomcat as standard clients to JBoss via typical JNDI access.

JBoss provides database access for the servlets and any remote, thick-client applications I might want to develop in the future. JBoss connects using the MySQL Java connector to MySQL 3, which serves as the local database.

The application servers use a configuring appropriate for installation on one or across a number of machines; thus simplifying the installation next time around. Because I am running my server environment on a Windows operating system, I have opted to install my server applications as NT Services wherever possible. Currently, and for the foreseeable future, this includes Apache, Tomcat, MySQL and Sygate Personal Firewall. JBoss must be run as a launched application, until a more suitable, automatic method is found. It is perfectly reasonable and feasible to have each application launched as a stand-alone application, without the use of NT services, but I am employing the Microsoft NT architecture as intended.

Things to watch out for.

When things go wrong it can be extremely infuriating. For example, I could not get a Stateless Session EJB to access an Entity EJB for a (very) long time. It turned out that I was not prefixing *comp/env/* to the start of the JNDI name I was searching for when doing a bean lookup. Before realising this I was absolutely convinced I was doing everything right and there was simply something I had not been told about which everyone else on the planet appeared to take for granted. It's at times like this you need to take a step back, read each key point of the process very carefully and maybe take a different viewpoint in your thinking process; there may be something you are unaware of. That's what this tutorial is for.

In that case and for the benefit of those using this document as a quick reference, I shall mark each key action and important entry with a red asterisk *. You can use these as a guide for when things go wrong or as a quick solution to reading the whole tutorial.

If you do need to go over anything, check the list in *Appendix I*. Each section also has a testing list for code and responses you will need from the practical execution before being able to continue. Almost as an addendum following the test list is a, typically longer, list of notes indicating items you should be aware of now and for the future.

It can sometimes help to explain things to someone else. This has the effect of tidying the facts up in your head. I was told of one University which actually used to have a teddy bear which students would stand in front of and explain their problem in as much detail as possible. This, apparently, cut down student-lecturer questioning by something like 60%. And that was a teddy.

Case study.

The process of constructing the site will require us to build certain code elements and configuration files. You can find these complete in Appendix C. As a naming structure they will use the value 'MyCaseStudy' (with appropriate letter casing) whenever we must distinguish package, path and value names. Aside from this, the installation described will be as it is for my own server.

Putting the whole thing together.

The following chapter will break the construction of the whole site down into its component applications and application connections. By this I mean that the installation, configuration and testing

of each application will be described, each followed by a section detailing how to connect the applications for a useful purpose. Not all of these will be needed by everyone, so just pick and mix for your needs. A quick skim through never hurts though.

One final point before we get started; you should always back up your work. I have found CVS very useful in keeping track of my program code and server applications. I also regularly archive my entire server to CD.

Preparations.

Introduction.

It often helps to have a pre-defined directory structure laid out before starting something like this, so that you know where everything is going to go, roughly, before you start on the big work.

Primarily, you are going to need somewhere for each download to be stored, just in case you need to reinstall. You will also need a location for each installed application to reside. The root directory of each installation is usually called the application's 'home' and has an 'environment' variable either set up by the operating system's shell script or, in the case of Windows XP, by the registry. Opening Windows Explorer can access this and following *My Computer -> Properties -> Advanced -> Environment Variables*. But don't worry about that just yet.

Aim in brief.

Here we will simply create useful directories for creating our tutorial project in a sensible place.

Creating the directory structure.

First off, please create the following directory structure for your installed server applications and downloads. It is not essential that you follow the structure below, but we will use it demonstration purposes. You will probably want to use a modified version which is compatible with your own, existing structure, but please bear in mind that each section here has a specific use. (It is often wise to install software on a separate drive, away from your Operating System's installation, just in case of serious failure).

*

```
<Drive letter>
  Development
    Client
      build
      src
      resources
    Server
      build
      src
      resources
      META-INF
    Shared
      build
      src
      resources
    Web
      build
      src
      resources
      jsp
      html
      img
```

WEB-INF

Downloads
Servers
Websites
MyCaseStudy

You will notice that there are two other root directories created. The first of these is for 'development' source code – the Java, C, C++, etc. applications that you will be writing for your actual web-based applications. I realise you may already have a development directory, so please bear in mind that this may require slight modification to allow building web-based applications for easy deployment. I will let you choose how you do this, though I'll make certain recommendations.

One recommendation is to use four source code directories for building each component of your application. *Client* contains code for deploying to remote clients of your server. *Server* contains code to be run exclusively under your JBoss server. *Web* contains code to be run exclusively under your Tomcat server. *Shared* contains code which is compiled into the build output of the other three, so that they can access the same code in certain situations. If it is not clear why, it will be in the JBoss section.

The second extra directory is for the 'websites' you will be hosting. There is nothing magical about the static content of a website – it really is just a series of normal html and other files. They need somewhere to live and be served from: *Websites/<site name>*.

Notes.

A well-defined directory structure is useful for laying out your work.

Regular backups and archiving of the installation ensures against cataclysmic loss.

Environment variables are often used by the applications to locate each other and resources.

Installing software on a separate drive, away from your Operating System, offers some protection.

Create directories for source code, downloaded files, server applications and web site files.

The source directory should contain directories for client, shared, server and web structures.

Apache.

Introduction.

Now that we have somewhere to download and install the web server to, please go to:

<http://httpd.apache.org/>

* Download the Win32 binary version of Apache Web Server 2.0.43 with *no SSL* to your Downloads directory.

Aim in brief.

Here we will install the Apache web server and configure it to server the static content of the website created in the previous section. We will test the website serving for the default website (provided with Apache), then locally for our own, then provide an Internet name for it.

Installation.

Once downloaded just run the installer and set it to install with * 'NT Service on' into the *Servers/Apache/Apache2* directory. This will install all the Apache files and create the Windows NT service, which can be accessed by going through *My Computer -> Control Panel -> Administrative Tools -> Control Panel*.

* You should find the *Apache2* service near the top of the services list with a status of *Started*. If you open the Properties for the service you should see it has been set to *Automatic*.

Set up.

To ensure the installation has worked, go to <http://localhost/> or <http://127.0.0.1/>, which is the I.P. address that should be bound to the *localhost* name. You should see an introductory page, this is stored in the *Servers/Apache/Apache2/htdocs* directory, but don't worry about that now.

Once we have served our test web page we need to configure Apache to serve our web site. This will not involve anything fancy, but you can learn more by reading the very helpful Apache manual found online and within the Apache installation.

- * Create a directory for the website *MyCaseStudy* inside *Websites/* and give it the *index.html* file from Appendix G.

Now we have a website to serve, lets configure Apache to serve it. This is quite simple.

- * Backup the file *Servers/Apache/Apache2/conf/httpd.conf* and add the entry from Appendix G to the end of it, making appropriate pathname changes.

On closer inspection of the Appendix G entry, you may notice that it does rather more than just make a website available. Please ignore most of this as we are only interested in Blocks 5 and 6.

Block 5 indicates to Apache that we are going to define numerous Virtual Hosts to listen on port 80 (the HTTP port) but that we don't know their I.P. addresses. Block 6 defines the first of these Virtual Hosts and identifies it with a specific domain name by use of the *ServerName* directive. Just so you know, the domain name given here will be the one people on the internet find your website by, but we will cover this later.

- * The *DocumentRoot* directive defines exactly where the static files for this website are to be served from. You should change this to reflect your website's location.

Now we have covered the set up of our website, lets test it. The <http://localhost/> will still find the default *Servers/Apache/Apache2/htdocs* pages, so we need to access the new site separately.

- * Change the **:80* in blocks 5 and 6 to be *mylocalcasestudy:80*
- * Add the fragment from Appendix G to your *C:\WINDOWS\system32\drivers\etc\hosts* file.
- * Restart your Apache2 service and open a new web browser (I recommend IE for this test).

You should now be able to enter <http://mylocalcasestudy/> to access your new website from the browser as if it were a real internet site.

- * Be sure, however, that when you serve a website to the outside world you change the *mylocalcasestudy:80* back to **:80* and remove the fragment from your *hosts* file. (We will briefly cover this to ensure it is done correctly in the next section).

The above is certainly not the only way to handle this issue of naming, but I'm not going to go into setting up named hosts in this tutorial – there is plenty of information on that in the Apache manual.

Testing.

NT Service 'Apache2' should be installed.
Go to <http://localhost/> for default test serve page.
Go to <http://mylocalcasestudy/> for local site.

Notes.

Get the 'no SSL' .exe or .msi installer.
Apache is installed as an NT service and therefore launches as soon as Windows starts up.
Add *httpd.conf* entry from Appendix G to the end of the *httpd.conf* file.

Change pathnames to reflect your machine's pathnames.
Use a *NamedVirtualHost* value in *httpd.conf* and an IP in *hosts* to access your website locally.
Use **:80* to serve to the internet, or read the Apache manual to find out more.

Allocating a Dynamic DNS name.

Introduction.

I am including this section because if, like me, you are just starting out with your own site you will need some way for people to find it. If you have a dynamically allocated IP address then you cannot assign a domain because it will change every time and your two year subscription fee will be wasted. Instead, use a dynamically bound, fourth level domain name. This is how.

I will be using the DynDNS.org website and DirectUpdate client in this section because I have found them to be free and easy to use. There are plenty more which you may wish to use – please take a look around the internet and my links. Of course, if you choose not to go with my choices, this section can only be a rough guide.

Aim in brief.

Here we will sign up to a free dynamic naming service and install a client to handle updating the domain name (chosen in the previous section) with our server's IP address. We will also ensure that we have the correct names in the *hosts* and *httpd.conf* files.

Installation.

First off, lets go to the <http://www.dyndns.org/> website and download the client. This can be found under *Dynamic DNS* on the left-hand menu. Follow the links through *Client -> Windows*. When you have 'DirectUpdate' click on the little disc icon on the right to download the client installer and save it to your *Downloads* directory.

* Before you can use the client you will have to sign up to the website in order to use it. Go to *Sign Up Now* and follow the instructions. Make a note of your details as you go. Once you are signed up go to *Login -> Dynamic DNS* and enter your details to login. We will now add the host name your Apache server will serve pages under.

* Click on *Add New Host* and enter the following details:

Hostname (part 1): mycasestudy
Hostname (part 2): homeunix.net
IP Address: *Ignore this*
Enable Wildcard: *True* (checked)

Then click on *Add Host* and it should take you to the maintenance page for that host.

You will find that adding, removing and changing as many host name details as you wish are all very simple. Now we simply have to set up the DirectUpdate service engine to update the DynDNS servers with our I.P. address.

* Go to your *Downloads* directory and install the *DirectUpdate* (it's name will be something like *DUSetup358.zip*) application to your *Servers* directory. You will probably have to specify explicitly that it install into *Servers/directupdate*. Be sure to select *Typical* or *Custom* (and select all options).

Set up.

Now we have the host name configured on the *DynDNS.org* website and have installed the *DirectUpdate* client we can configure the client to update the website with our IP. If you open your *NT*

Services (if you can't remember how to do this, refer to the *Apache Installation* section) you should see a new service called *Direct Update* started.

* You should also have a new icon in your system tray (bottom right, on the *Start* bar). Double click on it. You should have a window with a number of tabs – select *Engine connection*. You want it to connect to *Machine: <Localhost>* on *Port: Default* with no password. If the text in the bottom left of the window says *Successfully connected to 127.0.0.1* then everything is ok. If not, click on *Connect*.

* Now click on the *Status* tab and click the *Create* button. On the *Settings* panel you should be able to enter the following details:

Account type: *dyndns.org (dynamic)*
Domain: *mycasestudy.homeunix.net*
User name: *Your user name as entered on the website*
Password: *Your password as entered on the website*
Disable account: *False (Unchecked)*

* Now click on *Advanced settings* and enter the following details:

All options unchecked.
Enable wildcard: *True (checked)*

Then click *Ok* and the domain should be updated with your IP address. You may have to wait a few minutes for this update to happen. If it does not you should check the details for each host and possibly the information on the website for any current issues. Also check the list of common problems earlier in this tutorial, as it is usually something quite trivial.

We must now make sure that your web browser is not going to find your website by resolving it's domain name internally. As noted in the previous section this will require:

- * Removing the references to *mycasestudy* from your *hosts* file.
- * Ensuring the *httpd.conf* file contains **:80* and NOT *mylocalcasestudy:80* for *NamedVirtualHost* and *<VirtualHost ...>*.
- * Ensuring the *ServerName* directive within the *VirtualHost* tag contains the value www.mycasestudy.homeunix.net
- * Restarting *Apache*, in order to reload the updated *httpd.conf* file).
- * Restarting your web browser in order to reload the updated *hosts* file.

You should now be able to point your browser at <http://www.mycasestudy.homeunix.net/> and view your new website.

Testing.

Direct Update NT service should be installed.
The login details for the client must match the details on the site.
The hosts file must not contain a reference to the website we are trying to serve.
Apache's *httpd.conf* must contain wildcard information.

Notes.

A separate, free client is needed in conjunction with a dynamic domain naming service to connect your host name to your IP address.

The Direct Update client is an NT Service and will start whenever you boot up.

Apache must be aware of your host's domain name.

Your *hosts* file must not override the domain name otherwise it will not be resolved by the dynamic domain naming service.

Tomcat.

Introduction.

Now we have Apache installed and serving a website which can be accessed from the Internet. If you don't have it accessible from the 'net, just make sure you can access it.

This section will take the small step of installing the Tomcat Catalina servlet container, which will host our servlets and JSP's, thus giving us the ability to provide more intelligent web applications than just plain HTML, SHTML or CGI scripts.

The next section will take the large step of connecting Apache to Tomcat so that Tomcat only responds to requests from Apache, not the outside world.

Please go to:

<http://jakarta.apache.org/>

* Download Tomcat version 4.1.12. Get the Standard edition, preferably the .exe installation file, as it will be easier to install as a service on Windows NT.

Aim in brief.

We will install and configure Tomcat version 4.1.12 to serve a couple of demo servlets and JSP's. This is the easy part.

Installation.

* You should now be able to install Tomcat into *Servers/Apache/Tomcat 4.1* with NT Service option switched On.

Once you have finished installing, check the NT Services window and you should see an entry marked *Apache Tomcat 4.1*. It should be set to *Started*, if not then start it.

Set up.

When Tomcat is installed and running you should be able to point your browser to:

<http://localhost:8080/>

To see the demonstration page. This will host a number of example JSP's and Servlets. Once you have this page visible play around with the servlets and so on. You should be able to see that any application written for Tomcat is installed under *Servers/Apache/Tomcat 4.1/webapps* and that within this directory:

- Each application is deployed to the */webapps* directory and is within it's own named directory.
- All configuration concerning serving the web application is done using the */webapps/appname/WEB-INF/web.xml* file.
- Any servlet available to the browser is named within the *web.xml* file.
- All free-floating, compiled classes (servlets, utilities, etc.) are within the */WEB-INF/classes* directory.
- All support JAR's (whether directly part of the application or not) are within the */WEB-INF/lib* directory.
- All static files (HTML, etc.) are placed in the web application's directory and should be nested within an appropriately named subdirectory (html, jsp, etc.).
- All JSP's are also within the web application directory and should also be placed in an appropriately named directory (E.g.: jsp).
- It can be a good idea to place static pages that refer to servlets or JSP's within their own sub-directory.

You should be able to structure your development environment so that when you build and deploy your web application it's directory structure resembles something like the structure within */webapps/appname*. If not, try to write a build script for Ant, which does this.

Testing.

Check the demonstration site works via <http://localhost:8080/>
Ensure the demo servlets and JSP's are available and working.

Notes.

Tomcat is installed as an NT service and therefore launches as soon as Windows starts up.
Take a look at the structure of the */webapps* directory and the web applications within it.
Take a look at the Tomcat manual, accessible via the demo site.

Connecting Apache to Tomcat.

Introduction.

Now we have Apache and Tomcat installed we can get the two talking so that Tomcat is asked to handle Servlet and JSP requests and Apache will keep hold of the normal, static content. We can do this in a number of ways:

File associations for Tomcat to handle explicitly.
Everything under specific aliases (of a sort) get handled by Tomcat.

After this section you should see how to do both, but we will opt for the latter because it keeps things a little neater. Later on you can define specific associations if you need to.

For the time being, just know that we will simply access the *example* demo site provided with Tomcat from an association defined in Apache. For this job we will need to modify both Apache's *httpd.conf* and Tomcat's *server.xml* files. To keep the *server.xml* file simple we will remove the stand-alone configuration for Tomcat so that it only serves requests coming from Apache. Don't worry; this is only a matter of removing one large block of XML.

It should be said here that if you are thinking of connecting Apache Web Server to JBoss or MySQL that I cannot help you because I have not done it. I also don't intend to do it. However, I do know that it is quite possible and the procedure is not too far from the procedures described here. There are also plenty of websites with help.

Aim in brief.

Create a connection between Apache and Tomcat so that certain requests are forwarded from Apache to Tomcat. Also, restrict Tomcat from server to the Internet by explicitly serving Apache requests.

Set up.

First of all, you will need the web connector *mod_jk*. You should get this from:

* http://jakarta.apache.org/builds/jakarta-tomcat-connectors/jk/release/v1.2.0/bin/win32/mod_jk-2.0.42.dll

* And copy it to the *Servers/Apache/Apache2/modules* directory. It is the most recent version of any web connector I can get to work. It should soon be super-ceded by Coyote, but I have had no luck with that yet.

Now to get the connection between the servers configured. I strongly suggest at this point that you backup the *Servers/Apache/Apache2/conf/httpd.conf*, the *Servers/Apache/Tomcat*

4.1/conf/server.xml and (if it is present) the *Server/Apache/Tomcat 4.1/conf/workers.properties* files. This is where versioning systems can come in useful, helping to create a history of modifications to numerous files.

It might also be wise here to shutdown both servers. Got into the NT Services panel and select Apache and Tomcat in turn, hitting *Stop* for each.

* Replace the content of the *Servers/Apache/Tomcat 4.1/conf/server.xml* with the code from Appendix G. Modify the appropriate pathnames where necessary.

Be aware that while there is a lot of code in the new *server.xml* file I have stripped out everything required for running Tomcat as a stand-alone server. It should now only respond to Apache on *port 8009*. I advise comparing the old and new files to see the differences and learn a bit more about the configuration. Obviously some serious documentation is required to learn how to modify *server.xml* properly.

* Copy the *Servers/Apache/Tomcat 4.1/conf/workers.properties* file from Appendix G.

This file defines workers for the Tomcat server. Don't worry about the details just yet, although you might want to read up on how to use the workers file. It's pretty simple.

That's it for Tomcat's set-up. Tomcat can now listen for requests from Apache and serve them when they come in. We now need to configure Apache so it knows when to send requests to Tomcat. There are two ways of doing this.

* The first option you have already done. Look at Block 4 in file *Servers/Apache/Apache2/conf/httpd.conf*. You will notice that it:

- Loads the web connector module.
- Loads the Tomcat *worker.properties* file.
- Sets the web connector (*mod_jk*) log filename.
- Set the web connector log level (defines how much information to dump to the log file).

Now look at the end of Block 6. You will notice that we have 'mounted' the */examples* directory. This means that if Apache is asked for anything under the */examples* directory Apache will just pass the request on to Tomcat. I have been a little too explicit with the *JkMount's* than is strictly necessary, but it might help when understanding what can be done with the *JkMount* directive.

* If you would like to know how these details were generated, follow these steps:

- Ensure the Tomcat server is stopped.
- Uncomment the two lines in *Servers/Apache/Tomcat 4.1/conf/server.xml* which begin:

```
<!-- Listener className="org.
```

Note: They are preceded with comments indicating which two lines should be uncommented.
- Restart the Tomcat server.
- Wait a few moments for files to be generated in *Servers/Apache/Tomcat 4.1/conf/*
- Stop the Tomcat server.
- Re-comment the two lines in *server.xml*.

You should now find a new directory inside *Tomcat 4.1/conf* called *jk*, which should contain a *.log* file for *mod_jk*.

You should also find a new directory called *auto*. This will contain the automatically generated *mod_jk.conf* file needed to tell Apache how to talk to Tomcat. It will not be the same as the code that has been placed inside the *httpd.conf* file, but it will be similar.

It is possible, using the *Include* directive, to tell Apache to read the auto-generated *mod_jk.conf* file, rather than copying it into the *httpd.conf* file, but this way we can be sure it won't get changed without our doing so and hosting Tomcat on another server is less of a problem.

Btw, be sure to comment-out the lines in *server.xml* otherwise the *mod_jk.conf* will get generated every time Tomcat starts up.

You should now be able to call Apache from a web browser, giving the */examples* directory name. The result should be a call to Tomcat from Apache to run the examples provided in the demo webapps directory.

* Please note: The */examples* on it's own won't work because the examples directory does not have a default index page. Therefore, you will actually have to provide the name of the first HTML file you wish to see *within* the examples directory. This call will still be processed via the web connector because the */examples* name has been mounted using *JkMount*. As a kick-off, you might want to use:

<http://localhost/examples/servlets/index.html>

We will cover setting a default index page later. Don't worry; it's quite easy, too.

Testing.

Call <http://localhost/examples/servlets/index.html> to test the web connector.

Notes.

Tomcat is installed as a separate application.
JSP's and servlets are 'mounted' and are the only content served by Tomcat.
Ensure you restart Tomcat *then* Apache, to ensure Apache can find Tomcat.
Ensure the marked <Listener...> lines are commented-out of the *server.xml* file.
Make sure the workers.properties file is present in *Servers/Apache/Tomcat 4.1/conf*.
/examples has no default *index.html* so look for an HTML file within the directory to launch.
Ensure you have the complete list of environment variables set, check Appendix H for these.
When accessing a bean from another bean ensure you have:
 <ejb-ref> block inside the bean doing the accessing
 Use: *java:comp/env/* when looking for a bean from within another bean.

JBoss.

Introduction.

By now, you should know that JBoss is an Enterprise Application Server. It provides a lot of functionality, most of which won't be covered here. This section will simply aim to cover installation and configuration of the pure Java JBoss EAP. We will later cover using it in conjunction with Tomcat and a MySQL database.

Aim in brief.

We shall install and configure JBoss to run locally. This involves creating a bean to run on the server and a client to access that bean 'remotely'. This will be a good prelude to the following section.

Installation.

* First off, please go to:

<http://www.jboss.org/>

And download the *JBoss-2.4.4.zip* file. This is stand-alone JBoss, without the integrated Tomcat (we have already installed a separate Tomcat, able to run on a separate server).

While the 2.4.4 version is not the most recent, it is very stable and usable and I don't want to complicate this tutorial describing a version I have not fully used. You may, however, feel brave enough to download version 2.4.10, but using version 3.0.4 may not be wise as this tutorial does not focus on the very latest version of the EJB specification (as used by the latest version of JBoss).

* Installing JBoss is surprisingly easy, just extract the zip file to the *Servers/* directory and you will find *JBoss-2.4.4* containing the JBoss EAP. JBoss cannot run as an NT Service, so it will have to be started as a normal script. Lets run it from the command line, so go to:

```
Servers\JBoss-2.4.4\bin\
```

Then execute the JBoss Windows start-up using:

```
run.bat
```

You should see a lot of text spilling into your command line window. This is the Log4J output indicating what is starting up inside JBoss. Later on, when we have some beans created, you will be able to see the deployment of your EJB's towards the bottom of the log output. It might be wise to ensure that your command line window has enough buffer space to display quite a large portion of the log!

Set up.

Because the point of this exercise is to get Tomcat using JBoss we are going to have to copy some JAR's from JBoss to Tomcat. These will enable servlets running under Tomcat to act as clients to the EAP server. * Go to *Servers/JBoss-2.4.4/client/* and copy the following files to *Servers/Apache\Tomcat 4.1\common\lib*:

```
jbossmq-client.jar  
jbossex-client.jar  
jnp-client.jar  
jaas.jar  
jboss-client.jar  
jboss-j2ee.jar
```

* You will also need to include the above JAR's into your build classpath and have them present when the remote client is run. You can do this using the environment variable CLASSPATH, or use Jakarta Ant to automate the process. However you implement this will depend on your existing or eventual development environment.

JBoss can be started up as numerous instances, just as long as each instance listens on a unique port number. We are going to create our own start-up configuration for customisation and to avoid messing up the default.

* Go to *Servers/JBoss-2.4.4/conf/*, copy the directory */default* and call it */service*.

Normally, when we start JBoss it will use the */default* settings but if we start JBoss by using the command:

```
run.bat service
```

it should use the configuration settings from our new */service* directory. We will ignore this new configuration for the time being because the only other change we need to make is to add the database connection details, which will be done later.

Ok, now here is where it can get a little complicated. In order to make use of JBoss we are going to provide a server side utility in the form of a Stateless Session Bean. This is comprised of a number of components:

Convenience class '*ServiceClient.java*' – Providing just an easy way to access the bean remotely.

Remote Interface '*Service.java*' – Java Interface used by the client to access instance methods on the bean.

Home Interface '*ServiceHome.java*' – Java Interface used by the client to find instances of the bean on the server.

Bean Class '*ServiceBean.java*' – The Java Class instantiated on the server, holding actual bean methods and data.

XML – Entries in *EJB-JAR.xml*, *JAWS.xml* and *JBOSS.xml*, all within *META-INF*.

There is also the issue of the client class, in our case called *Client.java* and a support class, called *EJBUtil.java*, which helps our code find the EJB's located on the server. One other file is needed to make life easier, this is *jni.properties*.

All of the above files are provided in Appendix G. We will place and edit them as we go through this section. Their internals will not be discussed in great detail because they should be well commented, but plenty of information will be provided to get the whole caboodle up, running and later extended.

Code Structure.

As mentioned in the Preparation section, you will need a development environment which allows for client, server, shared and web source code and build structures. Ok, this is not strictly necessary and you probably have a good source structure developed already, but this tutorial will refer to files as being placed in certain directories to indicate their use. We will build each source section individually...

The Server-Side.

This structure contains code for running under JBoss. We will place server-side utilities, private resources and the actual bodies of the EJB beans here.

* First, we copy the files *application.xml*, *ejb-jar.xml*, *jaws.xml* and *jboss.xml* into *META-INF*. This is all reflected in Appendix F.

Application.xml provides information about the Enterprise Application Resource (EAR) which is deployed to JBoss. An EAR is nothing more than a specially named JAR which contains other JAR's to be deployed to various locations (like Tomcat).

EJB-JAR.xml describes the EJB's to JBoss so that it knows how they relate to one another, how they are intended to work and what their permissions are etc.

JAWS.xml describes the Entity bean relationships to the database(s) they will access. This file contains much information which must be matched properly to the data in *EJB-JAR.xml*, so be careful when changing either file. Most problems with getting EJB's to work with a database occur when fields are missing, unnecessary or incorrect between these two files.

JBOSS.xml simply tells JBoss which EJB's are to be made available to the remote client.

* Next, copy the bean class *ServiceBean.java* into the source code directory. The bean is in a package called *server.service.facade*, so create a directory structure to reflect this. This is also reflected in Appendix F.

The Shared Source.

That's it for the server-side only files. Now we have to provide the shared classes which the remote client (autonomous applications and web applications) can use for accessing the utilities provided by our application running on JBoss.

* Once you've created the package structure (as found in Appendix F) in the *Shared/src* directory, copy the files *ServiceClient.java*, *ServiceHome.java* and *ServiceRemote.java* into the *shared.service.facade* package.

These are the classes used to access the Service bean. They are shared by remote clients and servers so that clients and other beans can use the Service bean. On closer inspection, *ServiceClient* may appear a little unnecessary as it is very simple, but it is just a convenience API class and for some Stateless Session beans it may be superfluous. For Entity or complex Stateless Session beans a front-end API class for single (or even multiple) beans can come in very handy to co-ordinate utilities. Be careful, however, that you do not over-complicate matters. A very common trap is to create so many beans that a lot of bean API classes are needed. This easily causes such a complicated system that it is un-maintainable, so if you find yourself writing many SS beans and/or API classes go back and re-think your design.

* Now copy the *EJBNotFoundException.java* and *EJBUtil.java* classes into the *shared.util* package.

These are for cleanliness in our code. The exception class should speak for itself, the *EJBUtil* class provides methods for finding EJB's from within both a client and a bean running on the server.

If everything has gone to plan, you should be able to compile and deploy your server side code into JBoss. The Jakarta foundation's Ant application is very good for managing operations like this because, with some work, you can write build scripts which are dynamic, flexible and powerful.

* The JAR your build script creates should contain:

The code compiled from *Development/Server/src*.
The code compiled from *Development/Shared/src*.
The content of the *resources* directory from Server (including the *META-INF* directory).
The content of the *resources* directory from Shared.

* Once built, your script could also deploy your JAR by copying it into the *Servers/JBoss-2.4.4/deploy* directory.

Now, start up your JBoss and you should see the JAR get deployed, followed by the beans.

The Client-Side.

Fortunately, because we have already created the *Shared* source code, there is not much work in creating a stand-alone remote client. In fact, it's one class and a *jndi.properties* file. This file is nothing special, it is automatically located and loaded by the *InitialContext* class (used in *EJBUtil*) to locate the server. If you look in *EJBUtil.java* you'll see that it's information can be faked, mostly for flexibility or insurance, but it is useful to have it stored this way.

* Create the directory structure for *Client*. Include the source code structure and *resources* directory, then copy the *Client.java* class into the client package.

When JBoss is up and running with the beans deployed, you should be able to start another command line and execute something like:

```
java -cp <include jars here>;Client.jar client.Client
```

The output in this command line should be something along these lines:

```
Client: Service test...  
EXTERNAL LOOKUP  
Client: Server: Someone called ServiceBean->service()
```

And the output on the server command line should be something like:

Server: Someone called ServiceBean->service()

Testing.

Make sure you are launching the correct configuration (*service*).
Make sure the JAR created at build time is deployed properly into *Servers/JBoss-2.4.4/deploy*.
Check the beans have deployed properly in the JBoss window.
Check the client can find and access the Stateless Session Bean properly.

Notes.

Unlike Apache, JBoss is a single, command-line program and must be started manually.
JBoss is installed as a separate application, rather than being combined with Tomcat.
Ensure you have the complete list of jar's and that they are the right version.
Ensure you have the complete list of environment variables set, check Appendix H for these.
Ensure you have copied across the complete list of client jar's from JBoss to Tomcat.
Always shutdown the server before changing configuration settings.
Make sure EJB-JAR.xml and JAWS.xml relationships are accurate, check missing fields etc.
Make sure the content of both *resources* directories is copied to the root of the build JAR.
Make sure both *Server* and *Shared* sources are compiled into the JAR for deployment.
The *jndi.properties* file should be getting loaded, if not then the properties within *EJBUtil*.
Avoid writing so many Stateless Session beans and API client classes, to keep things simple.
To get as much information from JBoss as possible, set up the line:

```
log4j.appender.Console.Threshold=DEBUG
```

in the *JBoss/conf/mycasestudy/log4j.properties* file.

The type mappings for your Java/JDBC/SQL are defined in:

```
JBoss/conf/mycasestudy/standardjaws.xml
```

In your *JAWS.xml* file the `<jdbc-type>` and `<sql-type>` tag blocks must be treated as a pair – they cannot exist without each other.

If the tables are apparently not being created, but JBoss appears to think they are, it may not be able to create them because of a fault in their definition. Check the log output for the SQL.

If you have a finder with more than one argument it must have a finder tag in the *JAWS.xml*.

Be sure you are using the correct reference to another bean - `<ejb-local-ref>` or `<ejb-ref>`.

Connecting Tomcat to JBoss.

Introduction.

As we have installed Apache and Tomcat, it would be nice to make some use of the (admittedly pointless, so far) utility bean we have running on JBoss. To this end we will now create a servlet to run inside the Tomcat servlet container and use the same EJB API we have for remote, stand-alone clients to find the JBoss JNDI service and call the *Service* bean.

As you can see, the demonstration here is simple, but it does not take much to create much more useful beans and expand the API to quite a large scale, providing much functionality which remote clients of all types can use. It is important not to build too much API for your EJB's – remember, they only provide a service and if that begins to consume more resources than the building of your application, they are a waste of time and effort.

We will show here how a simple servlet can make use of the same EJB API as any other application.

Aim in brief.

Create a servlet to use the existing API and use it to provide the same output as the stand-alone client, but in a web page.

Set up.

* The development environment will need somewhere to create and build the servlet, as well as a configuration file to describe the web application to Tomcat. Build a directory structure following the description for the *Development/Web* directory in Appendix F for this.

* Copy the *ServiceServlet.java* into the *web.service* package, from Appendix G.

* Copy *web.xml* and *jndi.properties* into *WEB-INF/*, also from Appendix G.

* Now all you need is a build script which compiles the source code from *src/* into the *classes/* directory and copies the entire contents of *resources/* into the outgoing JAR (or, more appropriately for web applications, a WA) and your webapp is built. Have the JAR copied into *Servers/Apache/Tomcat 4.1/webapps/* and when you restart Tomcat it should get deployed.

Restart your servers in the following order: JBoss, Tomcat, Apache. Each will find its own resources and allow the next server to do the same. Point your browser to the url:

<http://localhost/examples/service>

You should see basically the same output as you had in the remote client's command line, with some additional output. The servlet and *web.xml* include some additional code to define an environment entry which is accessed via the JNDI service within Tomcat. As you can see, there is no special treatment for finding the JBoss JNDI service, other than informing *InitialContext* of the location of the JNDI server (already done inside *EJBUtil.java*).

Tomcat Bug.

It is appropriate here to point out that you should not try too hard to get Tomcat to automatically load the *jndi.properties* file from the classpath, as is done with stand-alone applications. This is because a bug in Tomcat stops the properties file from being found and we must therefore provide the properties for *InitialContext* ourselves. This is done inside *EJBUtil* until a fix for Tomcat is available.

Testing.

Restart servers in order: JBoss, Tomcat then Apache. This allows each to find its resources. The WAR (just a JAR named for webapps) gets deployed into Tomcat's *webapps/* directory.

Notes.

Remember, try to use the same API as for the stand-alone client, this keeps things simple. Be sure that any support JAR's your webapp needs are copied into the *WEB-INF/lib* directory. The *WEB-INF/classes* directory contains your own compiled classes, but not static files. A flaw in Tomcat stops it from loading *jndi.properties* so *EJBUtil* provides the values itself.

MySQL.

Introduction.

Here we are then, the final important piece to the puzzle: the database. Arguably the most important part of any organisation's business. For the purposes of this tutorial we are going to install MySQL, a free and easy-to-use database. Don't let that fool you though, MySQL is used by many companies as their main DB, even against such strong contenders as AS400 and Oracle.

This section will guide you through installing MySQL and creating a table into which we will place some test data. This will then be pulled out and manipulated in the next section. However, remember that proper testing requires a programmer to start with an empty table, populate it and continue on testing through all permutations of data configuration to avoid unexpected situations as much as possible.

Aim in brief.

Install MySQL as an NT Service and create a test table with some test data for EJB use later on.

Installation.

* Go to:

<http://www.mysql.com/downloads/index.html>

Download the Windows (Win32) version of MySQL 3.23.

* Probably the best way to install MySQL on Windows is to go to:

http://www.mysql.com/documentation/mysql/bychapter/manual_Installing.html#Windows_installation

and follow the instructions. On the same page are instructions for installation on other operating systems as well. Remember, this tutorial is only covering Windows.

* I will not cover installation of MySQL as the details in the tutorial (linked to above) really are straight forward. If it is requested of me I will insert more explicit instructions. See above for contact details.

* Essentially, you want to install the MySQL server as an NT service into the `/Servers/MySQL` directory.

Once you have finished installing you should have the pathname of the MySQL `/bin` directory somewhere in your PATH environment variable. If not, add it manually.

Set up.

* As with installation, using the database is probably best learnt by following the MySQL.com instructions, found at:

http://www.mysql.com/documentation/mysql/bychapter/manual_Tutorial.html#Tutorial

However, for the purposes of setting up data specific to this tutorial, we will go through the important parts of the process of logging in, creating a database and table, entering data and modifying that data. Again, the tutorial on the MySQL website really is very good and even if you don't read all of it, read the introduction so that you know how to modify data, manipulate databases and administer the database server.

There is also a GUI interface for the database:

<http://www.mysql.com/downloads/gui-mysqgui.html>

Although it is not an ideal interface, it will help with certain administration functions. It will not be covered here because it is not an essential tool and is quite easy to learn. Again, if requested, details of it's installation and use will be added.

Creating data.

Now to create a database on our MySQL database server, then to create a table within the database, then add some data and mess around a bit.

Login in to your database using the command line:

```
mysql -h localhost -u admin -p
```

You will then be asked for a password. Just press enter and you should be logged straight into the database.

Testing.

Notes.

MySQL is installed as an NT service and therefore launches as soon as Windows starts up.

Connecting JBoss to MySQL.

Introduction.

Here, we will create an Entity Bean which will represent a row of data from one of the tables stored in MySQL. We will also modify the Stateless Session Bean and the servlet so we can make some practical use of having an entity bean which can modify database data.

Aim in brief.

Installation.

* Also, get the Java/MySQL connector version 2.0.14 from:

<http://www.mysql.com/downloads/api-jdbc-stable.html>

Set up.

Install *mm.mysql-2.0.4.jar* and copy fragment of *jboss.jcml*.

Testing.

Notes.

Creating A Default Index Page.

...Using Apache Web Server.

...Using Tomcat Servlet Container.

Getting Server Side Includes Working.

Introduction.

This section will help you get server side includes working. These are simply HTML comments in your web page, which get pre-processed before being transmitted to the browser, in order to output dynamic content, to a degree.

This is by no means a replacement for CGI, servlets or JSP's, but can come in useful when building HTML-based menus etc. The tutorial on the Apache site is very informative and easy to follow, so I recommend you read through as it also lists the complete set of commands available in SSI:

<http://httpd.apache.org/docs-2.0/howto/ssi.html>

Aim in brief.

Activate and test some simple SSI commands in a new index page.

Installation.

You should already have the normal *index.html* page in your *WebSites/MyCaseStudy/* directory. * Now copy the *index.shtml* file from Appendix G into the same directory.

As you follow this section, read through Block A and Block 6 in file *httpd.conf* (Appendix G) to see just how to place the lines of directives.

Set up.

The new index page contains a simple SSI command which will get executed on the server and output some HTML in it's place when transmitted to the browser. In this case it will print the date and time, instead of the HTML comment. All SSI commands take place within normal HTML comments – it's the # character and special identification of the file type which indicates that it should be pre-processed by the SSI module before getting sent to the browser.

* To activate the SSI module within Apache, you will need to add the following line (See Block A in *httpd.conf* in Appendix G):

```
LoadModule include_module modules/mod_include.so
```

This loads the SSI module when Apache starts up.

* To tell Apache which types of file should be pre-processed add the following two lines:

```
AddType text/html .shtml  
AddOutputFilter INCLUDES .shtml
```

Don't worry about filters just yet, you have plenty of time to read up on those in the Apache documentation.

* Add the following line to the *VirtualHost* directive block for your *WebSites* directory and any file with a name ending *.shtml* will be pre-processed:

```
Options +Includes +Indexes
```

* Restart Apache and you should now have SSI activated for the virtual host which implements *+Includes* in it's *Options* directive. You can see the new page by visiting:

<http://localhost/index.shtml>

Testing.

Visit <http://localhost/index.shtml> to see the SSI produced date and time.

Notes.

Make sure *LoadModule* is called when Apache starts up, it must be outside of any blocks. The *AddType* and *AddOutputFilter* allow Apache to identify the appropriate SSI files. *.shtml* is used to separate normal HTML files from those needing pre-processing. The *Options +Includes* directive indicates SSI is to be used where-ever it appears.

Getting CGI Scripts Working.

Introduction.

This section describes a useful function of Apache which allows Common Gateway Interface scripts, such as Perl scripts, to be executed rather than downloaded, from a web page. As with SSI, CGI is not a replacement for servlets or JSP's, but pre-dates them and is a very powerful option for anyone not wishing to get bogged down in Java.

It is quite easy to get CGI scripts working on Apache, but a lot of people find misleading information or just hit minor, yet annoying, problems, so this tutorial is here to hopefully make the journey easier.

However, you should take a look at the official Apache tutorial page:

<http://httpd.apache.org/docs-2.0/howto/cgi.html>

Aim in brief.

Activate CGI scripts in Apache and provide a new index page utilising the Perl interpreter.

Installation.

* The installation process should be simple enough, so download the Windows (I recommend the MSI version because it comes with an Uninstaller) and install this into *Servers/*.

<http://www.activestate.com/Products/Download/Register.plex?id=ASPNPerl&a=e>

* Be sure to check that your *Path* environment variable has had the perl *bin/* directory added. On Windows you can check this by opening My Computer properties in Windows Explorer and going through Advanced -> Environment Variables.

Set up.

Testing.

Notes.

```
LoadModule cgi_module modules/mod_cgi.so
ScriptLog "E:/Java/WebSites/logs.eudoxus.homeip.net/cgi.log"

ScriptAlias /cgi-bin "E:/Java/WebSites/logs.eudoxus.homeip.net/cgi-bin"
IndexIgnore cgi-bin
# Alias /cgi-bin "E:/Java/WebSites/logs.eudoxus.homeip.net/cgi-bin"
# <Directory> directives cause directories to appear as a kind of file with a trailing /
<Directory "E:/Java/WebSites/logs.eudoxus.homeip.net/cgi-bin">
#   AddHandler cgi-script .pl
   DirectoryIndex index.pl
</Directory>
```

Appendix A: Required links.

Here are the websites you will have to visit for actual downloads of applications. Please download these applications as you will need them all for a database-driven website as I describe.

<http://httpd.apache.org/>

Download the Apache web server.

<http://jakarta.apache.org/tomcat/index.html>

Download the Tomcat servlet container.

http://jakarta.apache.org/builds/jakarta-tomcat-connectors/jk/release/v1.2.0/bin/win32/mod_jk-2.0.42.dll

The Windows version of the Apache/Tomcat connector.

<http://www.jboss.com/>

Download the JBoss Enterprise Application server.

<http://www.mysql.com/downloads/index.html>

Download the MySQL database application.

<http://www.mysql.com/downloads/api-jdbc-stable.html>

Download the Java connector for JBoss to MySQL connection.

http://java.sun.com/j2ee/sdk_1.3/techdocs/api/

<http://java.sun.com/j2se/1.4.1/docs/api/>

The Java Enterprise and Standard edition API's.

<http://www.dyndns.org/>

Provides dynamic name allocation for servers running with a dynamic IP address.

<http://www.directupdate.net/index.html>

The direct update for dynamic name allocation I use in conjunction with DynDNS.org.

<http://www.activestate.com/Products/ActivePerl/>

http://aspn.activestate.com/ASPN/Downloads/ActivePerl/Install_Notes

Perl interpreter for all platforms. Lots of utilities available too.

Appendix B: *Development links.*

These are some downloads I have found very useful for development of the site.

<http://jakarta.apache.org/index.html>

The open source community where you can find the Tomcat (Catalina) servlet container and the Ant build engine.

<http://www.webattack.com/>

Freeware and Shareware applications of all kinds.

<http://cygwin.com/>

The Cygwin UNIX environment for all platforms (inc. Windows) – a command line utility.

http://soho.sygate.com/products/shield_ov.htm

The Sygate Personal Firewall, a free download for protection on a small scale.

<http://www.wincvs.org/>

The Concurrent Versioning System for historical control of development code.

<http://www.scootersoftware.com/index.html>

For the “Beyond Compare” file comparison application, can be used with WinCVS.

<http://www.intellij.com/eap/>

The IntelliJ IDEA Java development Early Access Project.

Appendix C: *Useful links.*

Here are some links to very useful pages such as tutorials.

http://www.jboss.org/j2ee/dtd/jaws_3_0.dtd

The JBoss organisation's DTD directory.

<http://java.sun.com/dtd/>

Sun Corporation's DTD directory.

<http://bruno.vernay.free.fr/HowTo/Apache-tomcat/bWebServer/bWebServer.html>

A simple guide to setting up Apache and Tomcat.

http://www.cnr.berkeley.edu/~salazar/apache_tomcat.html

How to install Apache 2.0.43 and Tomcat 4.1 on Windows XP.

<http://www.mysql.com/documentation/mysql/bychapter/index.html>

The online manual for the MySQL database.

<http://www.mysql.com/downloads/gui-mysqgui.html>

The GUI for MySQL database.

<http://httpd.apache.org/docs-2.0/>

The online manual for Apache web server, version 2.

<http://httpd.apache.org/docs-2.0/howto/ssi.html>

The Apache 2 tutorial on getting Server Side Includes working.

<http://httpd.apache.org/docs-2.0/howto/cgi.html>

The Apache 2 tutorial on getting Common Gateway Interface scripts working.

<http://www.javaranch.com/>

The Java Ranch community hosts a number of forums, all of which have been indispensable.

<http://jakarta.apache.org/ant/manual/index.html>

The Ant application manual. Useful for developing your own build environment.

<http://www.jboss.org/online-manual/HTML/>

The JBoss online manual, which teaches EJB 2.0 for beginners. Also refer to for download of the JBoss Enterprise Application Server.

<http://www.inetsoftware.de/English/Produkte/CrystalClear/Doc/EJB/addingDatabasesJboss244.htm>

<http://ejbca.sourceforge.net/docs/HOWTO-database.html>

Connecting JBoss to MySQL HowTo.

http://java.sun.com/dtd/web-app_2_3.dtd

The DTD (document type definition) for the WEB.xml file format.

<http://www.alexandriasc.com/software/JavaService/index.html>

Module for turning Java programs into Windows NT Services.

<http://www.matthewwebster.homeunix.net/HowToApacheJBoss3.0.3.html>

Connecting Apache to JBoss and configuring JBoss with integrated Tomcat as an NT Service.

http://www.matthewwebster.homeunix.net/Apache2_Jk2_TC4.1.x_JSDK1.4.x.doc

HowTo for Apache and Tomcat on Windows NT and 2000.

<http://www.faqchest.com/>

The website for FAQ's of all genre's.

<http://www.mail-archive.com/>

The website for archived mailing lists of all topics.

<http://www.insecure.org/>

Advanced security discussion site.

<http://www.internetnews.com/>

Internet-oriented news site.

<http://www.javaperformancetuning.com/>

Informative site on performance tuning your Java code.

<http://www.devx.com/>

Software developer's community site.

<http://www.w3schools.com/>

Tutorials of all kinds, for many, many subjects.

<http://www.apacheweek.com/>

Apache web server community site.

<http://www.experts-exchange.com/>

Professional IT community site.

<http://www.javaskyline.com/index.html>

Java developer's community site.

<http://www.devdaily.com/>

Developer's community site.

<http://www2.theserverside.com/home/index.jsp>

J2EE developer's community site.

<http://freshmeat.net/>

Provides information on software.

<http://www.w3.org/>

World Wide Web Consortium, defining the world's XML-bred languages.

Appendix D: *Funny links.*

Lastly, some humorous links, to drive away the evil spirits...

<http://www.dilbert.com/>

The reluctant office hero.

<http://www.garfield.com/>

Living the life we all wish for.

<http://www.theonion.com/>

Providing the news we wish we could read.

<http://bofh.ntk.net/Bastard.html>

The most evil Admin there ever existed.

<http://www.softwarereality.com/>

The technical news we wish we could read.

<http://www.despair.com/>

For when it really isn't working.

Appendix E: *Useful books.*

This is a list of books, which I have found invaluable.

“Professional Apache Tomcat”, Vivek Chopra, Ben Galbraith, Sing Li, Chanoch Wiggers.
Wrox Press, <http://www.wrox.com/>

“Teach yourself Java 1.1 in 21 days”, second edition, Laura Lemay, Charles L. Perkins.
Sams.Net Publishing, www.sampublishing.com

Appendix F: *Advised Development Environment Directory Structure.*

The following structure describes the important directories which should be present once this tutorial is completed.

<Drive letter>/

Ensure this is not on the same drive as the Operating System.

Development/

Your source code for building web-based applications.

Client/

Source and build data for the remote client.

build/

The compiled output.

src/

The source code Java.

client/

service/

Client.java

Service remote client.

resources/

Static file resources, such as configuration files and images.

jndi.properties

A properties file for locating the JBoss server.

Server/

JBoss only, server-side code and resources.

build/

The compiled output.

src/

The source code Java.

server/

service/

facade/

ServiceBean.java

Stateless Session Bean.

resources/

Static file resources, such as configuration files and images.

META-INF/

Configuration files for the server-side application.

ejb-jar.xml

Describes and links EJB's.

jaws.xml

Describes Entity Bean database relationships.

jboss.xml

Describes the EJB's to JBoss for serving.

application.xml

Describes the Enterprise Application to JBoss.

Shared/

Shared classes and resources which are compiled into the other sources.

build/

The compiled output.

src/

Deployment directory for servlets, JSP's etc.

JBoss-2.4.4/
Enterprise Application Server serving EJB's for our servlets.
 conf/
JBoss configuration files.
 mycasestudy/
Our JBoss instance configuration.
 deploy/
Directory where EJB .jar and .ear files are deployed to.
 log/
Log file for JBoss Enterprise Application Server.

MySQL/
Database application providing data for our EJB's and servlets.

Websites/
*The root path for all the static content provided by each website.
 You may choose to place this directory on another drive.*

MyCaseStudy/
Files to be served from your website.
 index.html
Normal HTML index page.
 index.shtml
Index page with SSI commands.
 index.pl
Perl CGI index page.

Appendix G: Case study files.

`C:\WINDOWS\system32\drivers\etc\hosts`

This is just a fragment which should be appended to your hosts file for testing and local site access.

```
127.0.0.2    mylocalcasestudy
```

Websites/MyCaseStudy/index.html

```
<html>
<head><title>My case study main page.</title></head>
<body>
Hello, this is a plain index page.
</body>
</html>
```

Websites/MyCaseStudy/index.shtml

```
<html>
<head><title>My case study main page.</title></head>
<body>
Hello, this is an index page with some SSI.<br>
Current date: <!--#echo var="DATE_LOCAL" -->
</body>
</html>
```

Websites/MyCaseStudy/index.pl

```
#!<drive letter>/Servers/ActivePerl 5.6.1/bin/perl
```

```
print "Content-type: text/html\n\n";
print "<html>";
print "<head><title>My case study main page.";
print "<body>";
print "Hello, this is an index page generated by a Perl CGI.";
print "</body>";
print "</html>";
```

Servers/Apache/Apache2/conf/http.conf

Please note: This section should be added to the end of your *httpd.conf* file.

```
#####
# BLOCK 1
# All the information here is explained in detail in the Apache 2
# manual, provided with the Apache 2 installation under
# Apache/Apache2/manual.
#####

#####
# BLOCK A
# This line loads the Server Side Includes module when Apache starts.
LoadModule include_module modules/mod_include.so
#
# The following two lines tell Apache which files to pre-process when
# dealing with .shtml (SSI) files.
AddType text/html .shtml
AddOutputFilter INCLUDES .shtml
#
# Take a look at the Options directive in Block 6 to see how SSI is
# activated for a particular VirtualHost.
#####

#####
# BLOCK 2
# The next two directives allow your server to pre-process .shtml
# files containing SSI code.
# This is not covered in the tutorial, but it can be useful.
#####

#####
# BLOCK 3
# Filters allow you to process content before it is sent to the
# client.
#
# To parse .shtml files for server-side includes (SSI):
# (You will also need to add "Includes" to the "Options" directive.)

AddType text/html .shtml
AddOutputFilter INCLUDES .shtml
#####

#####
# BLOCK 4
# To connect Apache to Tomcat...

LoadModule jk_module <drive
letter>/Servers/Apache/Apache2/modules/mod_jk-2.0.42.dll
```

```

JkWorkersFile "<drive letter>/Servers/Apache/Tomcat
4.1/conf/workers.properties"
JkLogFile "<drive letter>/Servers/Apache/Tomcat
4.1/conf/jk/modjk.log"
JkLogLevel info
#####

#####
# BLOCK 5
# This is required to identify multiple virtual hosts with a dynamic
I.P.

NameVirtualHost *:80
#####

#####
# BLOCK 6
# Each virtual host is specified the same way: wildcard and port
number.

<VirtualHost *:80>

    # This specifies the domain name and is found via the dynamic
naming client.
    ServerName www.mycasestudy.homeunix.net

    # This is the email of the webmaster.
    ServerAdmin webmaster@youremail.com

    # This is the directory for this website's files.
    # All pathnames within this virtual host are relative to this
unless explicitly provided.
    DocumentRoot "<drive letter>/WebSites/MyCaseStudy"

    # This will produce an error log in the website's directory.
    # See below for why logs cannot be accessed without a
username/password.
    ErrorLog error.log
    # This will produce a log of access activity in the website's
directory.
    CustomLog access.log common

    # This allows server side includes to be used in html files and
an
    # index of the website directory to be generated.
    Options +Includes +Indexes

    # This allows the server to produce a suitably attractive
directory listing.
    IndexOptions +FancyIndexing +NameWidth=*

    # Matching tomcat served pages to tomcat connector...
    JkMount /examples/* ajp13
    JkMount /examples/*.jsp ajp13
    JkMount /examples/servlets/* ajp13

</VirtualHost>
#####

```

Servers/Apache/Tomcat 4.1/conf/workers.properties

```

# Start setup file
#
workers.tomcat_home=<rive letter>:/Servers/Apache/Tomcat 4.1
workers.java_home=<drive letter>:/Servers/j2sdk1.4.1
ps=\\
worker.list=ajp12, ajp13

# Definition for Ajp13 worker
#
worker.ajp13.port=8009
worker.ajp13.host=localhost
worker.ajp13.type=ajp13
#
# End setup file

```

Servers/Apache/Tomcat 4.1/conf/server.xml

```

<Server port="8005" shutdown="SHUTDOWN-APACHE-TOMCAT" debug="0">

<!-- Only uncomment this line for auto-generation of the mod_jk.conf
-->
<!-- Listener className="org.apache.jk.tomcat4.config.ApacheConfig"
      modJk="<drive letter>\Servers\Apache\Apache2\modules\mod_jk-
2.0.42.dll"
      workersConfig="<drive letter>\Servers\Apache\Tomcat
4.1\conf\workers.properties"
      jkLog="<drive letter>\Servers\Apache\Tomcat
4.1\conf\jk\modjk.log"
      jkDebug="info" / -->

      <!-- Uncomment these entries to enable JMX MBeans support -->
      <Listener
className="org.apache.catalina.mbeans.ServerLifecycleListener"
      debug="0"/>
      <Listener
className="org.apache.catalina.mbeans.GlobalResourcesLifecycleListene
r"
      debug="0"/>

      <!-- Global JNDI resources -->
      <GlobalNamingResources>

      <!-- Test entry for demonstration purposes -->
      <Environment name="simpleValue" type="java.lang.Integer"
value="30"/>

      <!-- Editable user database that can also be used by
      UserDatabaseRealm to authenticate users -->
      <Resource name="UserDatabase" auth="Container"
      type="org.apache.catalina.UserDatabase"
      description="User database that can be updated and saved">
</Resource>
      <ResourceParams name="UserDatabase">
      <parameter>
      <name>factory</name>

<value>org.apache.catalina.users.MemoryUserDatabaseFactory</value>
      </parameter>
      <parameter>
      <name>pathname</name>

```

```

        <value>conf/tomcat-users.xml</value>
    </parameter>
</ResourceParams>

</GlobalNamingResources>

<Service name="Tomcat-Apache">

    <Connector className="org.apache.jk.tomcat4.Ajp13Connector"
port="8009" minProcessors="5"
        maxProcessors="75" acceptCount="10" debug="0"

protocolHandlerClassName="org.apache.jk.server.JkCoyoteHandler" />

    <Engine name="Standalone" defaultHost="localhost" debug="0">
        <Logger className="org.apache.catalina.logger.FileLogger"
            prefix="catalina_log." suffix=".txt"
            timestamp="true"/>
        <Realm className="org.apache.catalina.realm.MemoryRealm" />
        <Host name="localhost" debug="0" appBase="webapps"
unpackWARs="true">

<!-- Only uncomment this line for auto-generation of the mod_jk.conf
-->
<!-- Listener className="org.apache.jk.config.ApacheConfig"
append="true" / -->

        <Logger className="org.apache.catalina.logger.FileLogger"
            directory="logs" prefix="localhost_log." suffix=".txt"
            timestamp="true"/>
    </Host>
</Engine>
</Service>

</Server>

```

Servers/JBoss-2.4.4/conf/mycasestudy/jboss.jcml

Development/Client/resources/jndi.properties

```

java.naming.factory.initial=org.jnp.interfaces.NamingContextFactory
java.naming.provider.url=localhost:1099
java.naming.factory.url.pkgs=org.jboss.naming:org.jnp.interfaces

```

Development/Server/cfg/META-INF/application.xml

```

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

<application>
    <display-name>Service application</display-name>

    <module>
        <web>
            <web-uri>srvice.war</web-uri>
            <context-root>/service</context-root>
        </web>
    </module>

    <module>

```

```
        <ejb>service.jar</ejb>
    </module>

</application>
```

Development/Server/cfg/META-INF/ejb-jar.xml

Development/Server/cfg/META-INF/jaws.xml

Development/Server/cfg/META-INF/jboss.xml

Development/Web/cfg/WEB-INF/web.xml

Development/Shared/src/shared/util/EJBUtil.java

```
package shared.util;

import javax.naming.InitialContext;
import javax.rmi.PortableRemoteObject;
import java.util.Properties;
import javax.ejb.RemoveException;
import javax.ejb.EJBHome;
import javax.naming.NamingEnumeration;
import javax.naming.NamingException;
import javax.naming.Binding;

/**
 * Provides methods for finding beans, whether being called from a
 * remote client or within another bean
 * on the server.
 */
public class EJBUtil
{
    /** Generic properties object for remote server lookup */
    private static Properties props = null;

    /**
     * Ensure presence of properties content.
     * This is really only here in case the jndi.properties file
     * cannot be found - it should be found
     * automatically by the InitialContext class.
     */
    static
    {
        if (props == null)
        {
            props = new Properties();
            props.setProperty(
                "java.naming.factory.initial", "org.jnp.interfaces.NamingContextFactor
                y");
            props.setProperty(
                "java.naming.provider.url", "localhost:1099");
            props.setProperty(
                "java.naming.factory.url.pkgs", "org.jboss.naming:org.jnp.interfaces")
            ;
        }
    }
}
```

```

    }

    /**
     Returns the home interface for a given ejb.

     @param jndiName The JNDI name of the bound ejb
     @param homeRef The Class object of the home interface of the ejb
     */
    public static EJBHome lookup( String jndiName, Class homeRefClass
)
    throws EJBNotFoundException
    {
        return lookup( jndiName, homeRefClass, null );
    }

    /**
     Returns the home interface for a given ejb.

     @param jndiName The JNDI name of the bound ejb
     @param homeRef The Class object of the home interface of the ejb
     */
    public static EJBHome remoteLookup( String jndiName, Class
homeRefClass )
    throws EJBNotFoundException
    {
        return lookup( jndiName, homeRefClass, props );
    }

    /**
     Returns the home interface for a given ejb.
     Note: This is not the best way to searche for beans, but it does
work.

     @param jndiName The JNDI name of the bound ejb
     @param homeRef The Class object of the home interface of the ejb
     @param props
     */
    public static EJBHome lookup( String jndiName, Class
homeRefClass, Properties props )
    throws EJBNotFoundException
    {
        try
        {
            // Get a naming context
            InitialContext jndiContext = new InitialContext( props );

            // attempt to retrieve a reference to the bean
            try
            {
                // lookup from remote client
                Object ref = jndiContext.lookup( "java:"+jndiName );
                System.out.println("EXTERNAL LOOKUP");
                // Get a reference from this to the Bean's Home
interface
                return (EJBHome)PortableRemoteObject.narrow( ref,
homeRefClass );
            }
            catch (Exception e)
            {
                // lookup from one bean to another

```

```

        Object      ref      =      jndiContext.lookup(
"java:comp/env/"+jndiName );
        System.out.println("INTERNAL LOOKUP");
        return      (EJBHome)PortableRemoteObject.narrow( ref,
homeRefClass );
    }
}
catch (Exception e)
{
    // unaccessible
    throw new EJBNotFoundException( e );
}
}

/**
 * Returns information on the beans bound under the provided jndi
name.
 * @param jndi The category name under which beans are to be
found.
 */
public static String listEJBtoString( String jndi )
{
    StringBuffer out = new StringBuffer();

    try
    {
        NamingEnumeration enum = listEJBs( jndi );

        // We're using JDK 1.2 methods; that's OK since J2EE
requires JDK 1.2
        while (enum.hasMore())
        {
            Binding binding = (Binding) enum.next( );
            out.append( bindingToString( binding )+"\n" );
        }
    }
    catch (NamingException e)
    {
        out.append( "\n"+e.toString()+"\n" );
    }

    return out.toString();
}

/**
 * Returns information on the beans bound under the provided jndi
name.
 * @param jndi The category name under which beans are to be
found.
 */
public static NamingEnumeration listEJBs( String jndi )
throws NamingException
{
    InitialContext initCtx = new InitialContext( props );

    //
        NamingEnumeration enum =
initCtx.listBindings("java:comp/env/ejb");
        NamingEnumeration enum = initCtx.listBindings( "java:"+jndi
);

    return enum;
}

```

```

    }

    /**
     * Returns a string providing the contained information from
    within a Binding object.
     * @param binding The Binding object pulled from a
    (initial)context object listing.
     */
    public static String bindingToString( Binding binding )
    {
        StringBuffer out = new StringBuffer();

        out.append( "Name: " + binding.getName( )+"\n" );
        out.append( "Type: " + binding.getClassName()+"\n" );
        out.append(
            "Value:          \n====\n"
binding.getObject()+"\n====\n" );
        out.append( "Is relative: "+binding.isRelative()+"\n" );

        return out.toString();
    }
}

```

Development/Shared/src/shared/util/EJBNotFoundException.java

```

package shared.util;

import javax.ejb.EJBException;

/**
 * Simple exception class provided for when the EJBUtil cannot find a
    bean requested by a client.
 */
public class EJBNotFoundException
    extends EJBException
    {
        public EJBNotFoundException()
        {
        }

        public EJBNotFoundException( Exception e )
        {
            super( e );
        }

        public EJBNotFoundException( String mssg )
        {
            super( mssg );
        }

        public EJBNotFoundException( String mssg, Exception e )
        {
            super( mssg, e );
        }
    }
}

```

Development/Client/src/client/Client.java

```

package client;

```

```

import shared.util.EJBUtil;
import shared.service.facade.ServiceClient;

/**
 * This is the cmdln launch class for all stand-alone client
 programs.
 * This is the class which should be modified in order to start the
 program in different ways, not the XML.
 */
public class Client
{
    public static void main( String[] args )
    {
        System.out.println( "Client: Service test..." );
        System.out.println( "Client: "+new ServiceClient().service()
);
//        System.out.println("\nEJB's...");
//        System.out.println( EJBUtil.listEJBtoString( "main" ) );
    }
}

```

Development/Shared/src/shared/service/facade/ServiceClient.java

```

package shared.service.facade;

import shared.service.facade.ServiceHome;
import shared.service.facade.ServiceRemote;
import shared.util.EJBUtil;

import javax.ejb.RemoveException;

/**
 * The ease-of-use class for the Service bean. Use this in your
 client to make finding the Service
 * bean more convenient.
 */
public class ServiceClient
{
    /**
     * Finds the service bean and calls it's service method.
     * @return
     */
    public Object service()
    {
        try
        {
            // Get a reference from this to the Bean's Home interface
            ServiceHome home = (ServiceHome)EJBUtil.remoteLookup(
"main/ServiceFacade", ServiceHome.class );

            // Create an ServiceRemote object from the Home interface
            ServiceRemote serviceRemote = home.create();

            // call the service method
            return serviceRemote.service();
        }
        catch (Exception e)
        {
            return e;
        }
    }
}

```

```
    }  
  }  
}
```

Development/Server/src/shared/service/facade/Service.java

```
package shared.service.facade;  
  
import javax.ejb.EJBObject;  
import java.rmi.RemoteException;  
  
/**  
 * Provides the client with a remotely accessible instance of the  
 * Service bean.  
 * Instances of this interface are produced by use of the ServiceHome  
 * class.  
 */  
public interface Service  
    extends EJBObject  
{  
    public Object service()  
        throws RemoteException;  
}
```

Development/Shared/src/shared/service/facade/ServiceHome.java

```
package shared.service.facade;  
  
import java.rmi.RemoteException;  
import javax.ejb.CreateException;  
import javax.ejb.EJBHome;  
  
/**  
 * Home interface for use in the client.  
 * Home interfaces provide a way of finding and creating instances of  
 * the remote interface.  
 */  
public interface ServiceHome  
    extends EJBHome  
{  
    /**  
     * Creates an instance of the `ServiceBean' class on the server,  
     * and returns a  
     * remote reference to an ServiceRemote interface on the client.  
     */  
    ServiceRemote create() throws RemoteException, CreateException;  
}
```

Development/Server/src/server/service/facade/ServiceBean.java

```
package server.service.facade;  
  
import javax.ejb.SessionBean;  
import javax.ejb.SessionContext;  
  
/**  
 * Service EJB.
```

```

* This class resides on the server and provides the server-side
functionality used by the client
* via the remote and home interfaces.
*/
public class ServiceBean
    implements SessionBean
{
    /**
     * This is our own method.  Provided to do whatever we want to
do.
     */
    public Object service()
    {
        System.out.println( "Server:  Someone  called  ServiceBean-
>service()" );
        return new String( "Server:  Someone  called  ServiceBean-
>service()" );
    }

    // Methods from here are required by inheritance...

    /** Empty method body
     */
    public void ejbCreate()
    {}

    /** Every ejbCreate() method ALWAYS needs a corresponding
    ejbPostCreate() method with exactly the same parameter types.
     */
    public void ejbPostCreate()
    {}

    /** Empty method body
     */
    public void ejbRemove()
    {}

    /** Empty method body
     */
    public void ejbActivate()
    {}

    /** Empty method body
     */
    public void ejbPassivate()
    {}

    /** Empty method body
     */
    public void setSessionContext(SessionContext sc)
    {}
}

```

Appendix H: Required environment variables.

ANT_HOME	ANT home path.
CATALINA_HOME	Tomcat home path.
CLASSPATH	Source code home path.
J2EE_HOME	J2EE installation home path.
JAVA_HOME	J2SDK home path.

JBOSS_DIST Path
JBoss home path.
MySQL home path, along with any other pathnames the system normally requires.

Appendix I: *Checking problem symptoms.*

- Spelling of file names (spaces, commas, full-stops, hyphens, etc.).
- Spelling of keywords, such as XML tags.
- Proper completion of XML tags. E.g.: <a>
- JNDI names being correctly filled out. In code as well as scripts and XML. (Note: External JNDI names to EJB's are not the same as EJB-to-EJB names).
- Does the bean require an <ejb-ref> tag?
- Are all the fields matching and correctly spelt between EJB-JAR.xml and JAWS.xml?
- Are the fields in EJB-JAR.xml and JAWS.xml all there and all necessary?
- The correct application, reference, routine, etc. is being called, rather than what you *think* is.
- Inheritance flowing correctly.
- Global versus Local configuration entries – Apache httpd.conf entries work similar to code inheritance; you need to override settings occasionally.
- Is something overriding your configuration? E.g. With Apache you might have a .htaccess file overriding your httpd.conf file.
- Is restarting a server necessary? Sometimes Apache will not be able to talk to Tomcat if Tomcat was restarted after Apache was restarted. The same goes for JBoss.
- Is the server where it is supposed to be? Are you looking for it locally when it is on another machine or behind a firewall?
- Is the server/application listening/binding/accessing the correct location/port/JNDI/object?
- Is the version of the API, server or web connector the one described in this tutorial? I cannot guarantee success if it isn't.
- Is the data definition in the table correct? You may have to check the data type definitions for your DB.
- Have you recompiled *all* of your code?
- Have you commented out something which should not be commented out?
- Have you un-commented something which should be commented?
- Have you *cleaned* your code before recompiling? There may be a class or configuration hanging around, being used instead of your intended class, configuration or settings.
- Is the NT service currently running? (If you have been tearing your hair out because Apache is not returning a web page and you discover the Apache service is stopped, go to bed and tackle it tomorrow).
- Is the problem listed on one of the forums and is it common? Please check the forum archives and FAQ's before posting what may be a regular problem. Don't feel silly asking a question; maybe no one has seen it before. Feel silly asking a question if it is in the FAQ and you were too lazy to read it. It is also useful to subscribe to the mailing lists and keep a record of interesting problems and solutions people post. (You should be able to find subscription details on the pages listed in Appendix C).
- Yes, you can even mail me. But not all at once.

Appendix J: *Glossary.*

Apache

The award winning web server from the Apache Foundation. This serves static files and can perform calls to other programs to expand functionality. Very flexible and powerful. Also known as the Apache HTTPD Web Server.

Apache Foundation

The non-profit, open source organisation responsible for creating the Apache HTTPD web server and starting the Jakarta non-profit, open source project.

Catalina

The current name for the Tomcat servlet container engine.

DirectUpdate

The freeware Dynamic Domain name updater. This allows a computer with a dynamically allocated IP to update a fixed-IP server so that it's IP can be re-bound to a fixed domain name.

DNS

Domain Name Server. A DNS service binds computer IP addresses to domain names for human-readable locating of servers.

Domain (Name)

A domain is one or more IP addresses which can be used to uniquely identify a computer on the internet. The domain name is a string associated (bound) with each IP address to provide a sensible name, which humans can use to locate the computer.

DynDNS.org

Just one website providing dynamic DNS services.

EAP

Enterprise Application. Any large application which is deployed and run on an EAP Server, such as JBoss, and provides database and remote access functions typically.

EJB

Enterprise Java Bean. Any class fitting the Sun specification for an EJB so that it can be called and execute on an EAP Server.

ejb-jar.xml

The configuration file for EAP Servers, which provides descriptions of EJB's for the internal workings of the server.

HTML

Hyper Text Mark-up Language. The language used to write web pages with. Can be expanded to SHTML.

HTTP

Hyper Text Transfer Protocol. The protocol used by computers to transmit web content across the Internet.

HTTPS

Secure Hyper Text Transfer Protocol. The secure (encrypted) version of the HTTP protocol, which provides a level of security for transmitting web content. Typically used on websites for submitted forms containing sensitive information, e.g.: credit card numbers.

HTTPD

Hyper Text Transfer Protocol Daemon. A web server that supports the HTTP protocol, such as Apache.

httpd.conf

The primary configuration file used to set up Apache web server.

Internet

Worldwide computer network that allows communication and data transfer between people connected to it.

IP

Internet Protocol. Standard which regulates computer connection on networks that make up the Internet.

SHTML

Server Hyper Text Mark-up Language. A normal HTML file, which also contains Server Side Includes. An SHTML file must be pre-processed (executed) before being transmitted to a client (e.g.: web browser) or the outgoing content will not be pure HTML.

SSI

Server Side Include. A simple language comprised of individual commands placed into HTML comment tags to be executed by an HTTP Daemon before being transmitted to a client.

SSL

Secure Socket Layer. The layer of abstraction which is used to de/encrypt HTTP data when implementing HTTPS.

Jakarta

The open source project spawned from the Apache Foundation which is responsible for many useful Java projects.

JBoss

The free, open source Enterprise Application server written in pure Java.

JNDI

Java Naming Directory Interface. An API that provides naming and directory functionality for applications to locate resources and applications which may be local or remote.

jndi.properties

The properties file which indicates how a client should locate a JNDI service.

JSP

Java Server Page. An HTML file which contains specially formed tags containing java. This is pre-processed into a servlet and executed to produce a web page on any servlet container.

META-INF

The configuration directory which resides within any Sun EAP Specification compliant application.

mod_jk.conf

The configuration file which allows Apache to talk to Tomcat. May be referenced by or directly copied into the httpd.conf file.

MySQL

The free, open source database.

Servlet

A Java class written to be invoked by a servlet container, such as Tomcat, to output HTML.

server.xml

The main configuration file for the Tomcat servlet container.

Tomcat

The name of the application based on the Catalina servlet container.

Web connector

A plugin which allows a web server to proxy requests to another application such as a servlet container.

WEB-INF

The configuration directory which resides within any Sun Web Application Specification compliant application.

web.xml

The configuration file within a web application which defines how the servlet container should provide access and execute the web application servlets.