Good Practices Struts

Hands on with

Examples

JSP 2, Struts, Eclipse 3, DAO, ANSI SQL DB and more

Preamble

Are you a Mechanic or a Driver?

Some people wake up in the morning and go to their garage. Then they roll under the car to tweak the engine. They worry about four-stroke versus rotary, the plug timing and cycles.

Other people also wake up in the morning and go to their garage. Then they drive the car to work and look down the highway. They tend not to worry what cycle the engine is in at the moment, the exhaust or the fuel injection system.

In programming, there are also two kinds of people: system programmers and application programmers.

System programmers write code that is used and valued by application programmers. System programmers write frameworks or DAO implementations. Application programmers, however, write for end users—to **provide business value**. They leverage other people's work so that they can focus on the application and not the mechanics.

When writing applications, it is more effective to leverage a framework and other reusable JARs and components than to reinvent the wheel

This book shows you how

- to build *quality* web applications
- at dramatically *increased speed*
- *cost-effectively*

with the open source Jakarta Struts framework.

The Jakarta Struts framework, a project of The Apache Software Foundation (www.apache.org), provides a sound application architecture and dramatically reduces the time it takes to write web applications that deliver value to the business.

Struts is based on published standards and proven design paradigms, such as the Model 2 approach, a variation of the well-established Model-View-Controller (MVC) architecture (in use since 1970s) and is compatible with Sun's Java/*Java 2 Enterprise Edition* (J2EE) platform.

Together with *Java Standard Tag Library* (JSTL), Struts brings *Java Servlets*, *JavaBeans*, *Java Server Pages* (JSP), *Extensible Markup Language* (XML), and message resources into a unified framework.

What Will You Learn?

In the first part of the book (The Foundation), you will learn to pave the way for the development phase with Struts and JSTL. It covers software and hardware requirements, sound requirements specification, database setup, the use of simple actions and beans, the use of JSP and some tips on project planning.

In the second part of the book (The Bricks and Mortar), you will learn how to create a solid and high-performance application infrastructure, and benefit from a high degree of reuse. With this infrastructure, you will have a means to build solid application modules very quickly and efficiently.

In the third part of the book (The Perspectives), you will learn how to provide a richer user interface, enhance security, and how to make your application future-safe and ensure that it scales well in a production environment.

Best Practices

Anyone can build a bridge or a web application. But by knowing (and applying!) best practices, an engineer can build web applications optimally: best quality, the fastest, the cheapest.

The authors have seen development projects fail miserably when architects and developers mistakenly took the excellent J2EE blueprints as their only guidance. Therefore, this book does not cover everything that you could do with J2EE and Struts. The authors rather expose practices that they have found to work best in real life to build web applications that deliver the highest value to the business in terms of quality, speed and cost.

For example, this book addresses issues such as how to increase development productivity by maximizing reuse of beans, actions and events, how to increase reliability of the application by the choice of software and hardware, and the importance of sound requirements specifications and project planning for cost-effective development that meets client needs.

“Fools ignore complexity, experts avoid it; geniuses remove it.” (Alan Perlis

The authors do not claim to be geniuses; they have simply tried to keep the book simple and short by concentrating on the approaches that work best in practice and still cover the complete project cycle.

The following are 12 best practices which contribute to producing quality web applications speedily and cost-effectively. They are presented in the following chapters:

1. Meet client needs efficiently via sound requirements specifications.

2. Save money and increase reliability with open source and effective hardware.
3. Develop iteratively with a proven process.
4. Unit test.
5. Apply MVC(Model-View-Controller).
6. Use OO (object-oriented programming) to maximize reuse.
7. Use DAO.
8. Use CRUD Events.
9. Use CMA.
10. Stress test.
11. Use Q/A and manage releases.

Hands on

This book wants you to be immediately operational, so it relies heavily on tutorial exercises. To really benefit from this book, better do not skip the labs as they help you understand and gain confidence in the demonstrated technologies.

Once you have completed the book and the labs, you may wish to test your new competencies on basicPortal CMS (Appendix A) which has been built using the best practices illustrated in this book.

basicPortal CMS is a dynamic, Struts-based portal application with content management that combines the functionalities and features that are required for about 80% of all web projects. It thus allows you to concentrate on features that are unique to your web project.

BasicPortal includes support for e-commerce/credit cards, news, lead tracking, content syndication, fora, calendars, web logs ("blogs"), wikis, email support, high-speed standard J2EE security, row-based security, images, blobs and uploads.

Why Struts?

To speed up development, you can acquire a proprietary framework, but you may find that the vendor does not allow you to open the hood to change the oil. Jakarta Struts is open source and does allow you to open the hood. *You* have control over the source code and can change it yourself to build a web application that fully meets your client's needs.

Besides, Struts offers more advantages for both developers/programmers and for your client's business:

Major developer benefits:

- Runs on top of any J2EE application server, including *IBM WebSphere*, BEA *WebLogic*, *Oracle IAS*, *Borland*, *Novell exteNd*, *Tomcat*, *Resin* and *Orion*.
- Extremely solid and stable architecture (community's code contributions increase stability).
- Ease of use, yet suitable for large-scale data-driven web applications.
- Breaks complex applications into simple, consistent sets of components.
- Ensures all developers in the team develop code using the same approach.
- Complete documentation (user's and developer's guide).
- Encodes best practices as many users help debug and improve open source code.
- Large developer community provides virtually instant support.

Major business benefits:

- Speedy and cost-effective development.
- Substantially reduced time-to-market and quicker product releases.
- Open source: license is either free or almost free.
- Freedom from closed, proprietary systems, thus no costly licenses to keep track of, renew and upgrade.
- No dependence on specialist labor and vendors.
- Numerous features meet a variety of business needs.
- Platform-independent.
- Greatly modular.

Who Should Read this Book

This book has been written for experienced Java developers and programmers who are either working as part of a multi-disciplinary IT team for an internal client or as independent consultants for external clients.

If you are an independent consultant responsible for the client project from A (such as analysis of user needs) to Z (such as zipping up a completed project), then you should understand and be able to implement all project phases yourself.

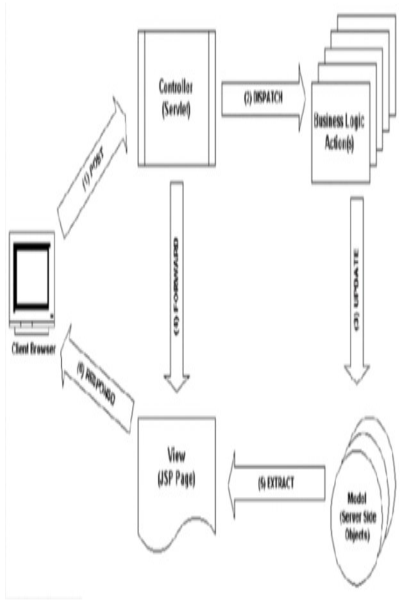
If you are part of an inhouse, multidisciplinary IT team, it may be wise to also understand the taks and responsibilities of others in the team, such as the project manager, the system analyst, the architect, technical support staff, etc.

If a project fails, all too often the blame is put on the developer(s)/programmer(s), so in the interest of the profession in the long run and in your own interest, it can't hurt to be competent and able to suggest better approaches, for instance, to insist on a complete requirements specification prior to development, recommend a choice of software or advocate for an effective project management technique.

This book is great for a team effort. Feel free to compare notes with your colleagues. If you are or will be participating in a large project, building and working on the basis of a common understanding of technology can do wonders for project success.

•

What Readers Should Already Know (Prerequisites)



To benefit from this book, you should already have an understanding of the Model-View-Controller paradigm (MVC). Figure 1-1 shows the MVC2 architecture that we will use with Struts and JSP. You should be aware of the request-response process and terminology:
(1) The Client Browser posts a request to the Controller servlet.
(2) The Controller servlet dispatches to an Action class.
(3) The Action positions the Model (i.e. retrieves, updates, inserts, deletes data).
(4) The Controller then forwards to a JSP page.
(5) The JSP extracts the positioned data from the Model.
(6) The Controller returns the JSP content as a response.

You should know what JSPs are and how they work.

You should also have some experience with the Java Servlet API, HTML, SQL and JavaScript.

The more Java programming experience you have, the better. In an ideal world, you will have already participated in developing a Java application that has made it into production.

Quickcheck 1: On the Internet, go to http://java.sun.com/products/servlet/2.3/javadoc/index.html.

Are you familiar with the API?

Quickcheck 2: Do you understand the following HTML fragment?

```
<FORM ACTION=http://localhost:80/myServlet>
First Name: <INPUT TYPE=”TEXT” NAME=”firstName”><BR>
Last Name: <INPUT TYPE=”TEXT” NAME=”lastName”> <BR>
<INPUT TYPE=”SUBMIT”>
</FORM>
```

If the answer is “no” to either question, the authors strongly recommend that you do some prior reading or find a team member that will bring you up to the required level.

Code and Software Used in this Book

This book uses standard programming techniques that work with any choice of servlet engine, application server, SQL database and IDE.

This book is a sort of cookbook because a lot of projects don’t just use Struts, but other software ingredients as well. Whenever other “ingredients” are required, this book advocates the preference of open source software over proprietary software because of the many advantages that open source offers (see Project Setup).

The authors have tested and applied open source and proprietary solutions and learnt “how to fish”.

No single software company, whatever its size, can create and deliver all “best-of-breed” tools as proprietary; there are just too many “brains” in the open-source community to consistently match and beat proprietary efforts.

Intellectual Property and License

This book discusses and demonstrates the use of licensed products. Most are open source licenses but all need a compliance to the license. The software in this book is discussed and available for download only as a convince, so by illustrating an example you do not own the software, and you need to make sure you know who owns it and what you are licensed to do.

Table of Contents

Fools ignore complexity; experts avoid it; geniuses remove it.

- Alan Perlis

Part I The Foundation

Chapter 1: Project Setup

Software Requirements

This book uses standard programming techniques that work with any choice of servlet engine, application server, SQL database and IDE.

To complete the tutorials in this book, you need the following:

- A Java application server/container (JSP 2.0) such as Tomcat 5
- An SQL database, such as pgSQL
- An integrated development environment editor (IDE) such as Eclipse
- A 1.4 Java Virtual Machine (JVM) such as J:Rockit
- Commonly needed JARs, such as for Struts 1.2, JSTL 1.1, Data Access Object (DAO) implementation (such as iBatis).

You could download and install these by yourself. However, configuring a complete environment with IDE and application server speedily and that works smoothly together can be time-consuming.

The book's tutorials and sample code, therefore, use *Jasic*, an integrated development "suite" that the authors have created to enhance your project's cost-effectiveness and that has already shown to work well together. *Jasic* provides enormous productivity gains and allows you to concentrate your resources on features that are unique to your project.

Jasic, named after "Jakarta" and "basic", leverages several of The Apache Foundation's Jakarta Struts projects into a comprehensive set of features and functionalities for web application development. Its components are either pure Open Source or free for development and testing purposes at the time of releasing this book.

- *Jasic* includes the following popular components:Tomcat 5.x and Resin 3.x Application Server (for development)
- PostgreSQL 7.4 database with pgAdminII administration tool

- Eclipse 3 IDE with SolarEclipse web applications plug-inJava Virtual Machine 1.4 (non Sun)Struts 1.2, JSTL 1.1, iBatis DAO implementationbasicPortal CMS best practices sample codeiReports Visual Report Designer

Jasic can be downloaded from the Web by following instructions available at http://www.basebeans.com and http://www.infonoia.com.

The components provided with the *Jasic* suite are for the Windows 2000, NT, and XP operating systems. All components are, however, also available for Linux, so you could create a complete environment for Linux instead.

A cheat sheet with links to further resources is maintained at http://www.basebeans.com. Feel free to use your favorite IDE in parallel with the tools provided with the *Jasic* suite, and explore the differences and advantages of either.

The authors further recommend that you download and install a Netscape or Mozilla browser. Some HTML tags that work in Internet Explorer (www.microsoft.com) may not work in other browsers, so it is best to test on a variety of platforms. The mentioned browsers are available for download at http://www.netscape.com and http://www.mozilla.org.

Finally, make sure that you are using a recent Java Development Kit (JDK). You can test this by typing the following at a command prompt:

```
java -version
```

The response must indicate a JDK 1.4 or higher.

Hardware andOperating System Configuration

The development environment hardware should allow the software engineer to work productively.

Large monitors, such as used by graphic artists, improve development productivity and lower the cost of developing software. A 21 inch monitor, with a minimum screen resolution of 1200 x 1600 pixels, as well as dual monitor configurations. It is a shame to see an organization where a software engineer is using the same PC that office automation staff uses. For a workstation I like a Dell Precisions 450 class machine.

You will wish to see all the code you work on, such as a complete if/then block, without scrolling. On a dual monitor setup with a supporting graphics card, you can see the browser and the log output on the console of the application server on the second screen, side by side with your code on the first screen, without moving the mouse.

Memory is also very important for productivity. Try getting 512 megabyte of RAM or more on the development machine; you will be able to work much faster than on a machine with 384 megabyte of RAM only.

If you develop on Windows, the authors recommend that you remove or disable all memory resident programs that autoload. In the task manager of your system, take a look at the processes that take up thread slices. Rename the executable so it will not autoload when you reboot the machine. For example, rename MSIM.exe to MSIM_bak.exe. You can then still load it when you need to.

Virus detection software which runs in the background can also be detrimental to developer productivity. Mostly, virus detection software is necessary to avoid execution of macros in the Outlook or Outlook Express mail programs which are tightly integrated with the operating system. If you use a mail program such as Netscape Messenger which does not execute macros, you may be able to disable the virus checker without having to worry too much about getting infected.

Java is cross platform which gives you the freedom to choose the best operating system to meet your needs. If you deploy your application to a Unix system, consider making Linux your operating system for development. Linux uses little system resources and can therefore be comparatively fast on a given hardware, which reduces the cost of development. Also, the stability of Linux may allow you to avoid having to re-boot your machine several times a day to free up system resources.

The book labs are Windows based, but developers who use Linux should be able to pick up on the instructions.

IDEs and Code Generators

At the time of writing, the authors are aware of a few IDEs that have *Graphic User Interface* (*GUI)* tools with two-way code generation and editing functionality in relation to Struts:

- Camino (http://www.scioworks.com)
- Exadel (http://www.exadel.com)
- ObjectAssembler (http://www.objectventure.com)
- M7 (http://www.m7.com)

Code generators can also be used to automate the creation of Java classes, JSPs and XML configuration files.

Check http://www.basebeans.com and http://www.infonoia.com for latest information about available code generators.

I

Production Environment

The cost of production operations is an import factor for larger commercial applications. Departmental users have traditionally accepted runtime licensing costs for application server, portal server and the SQL engine. The operations cost for commercial applications that run in order to generate profits must be scrutinized very closely. These commercial applications typically require a rack of servers running in a cluster. If the total license cost was $5,000 (USD) per machine, and there are 192 machines in the rack, you end up paying one million USD just in license fees. For this kind of money, you may in theory be able to develop your own SQL database software rather than license one and still come out on top. The licensing costs are much higher than $5K per CPU. Oracle DB, MS Advanced Server, Application Server, Portal Framework, etc.

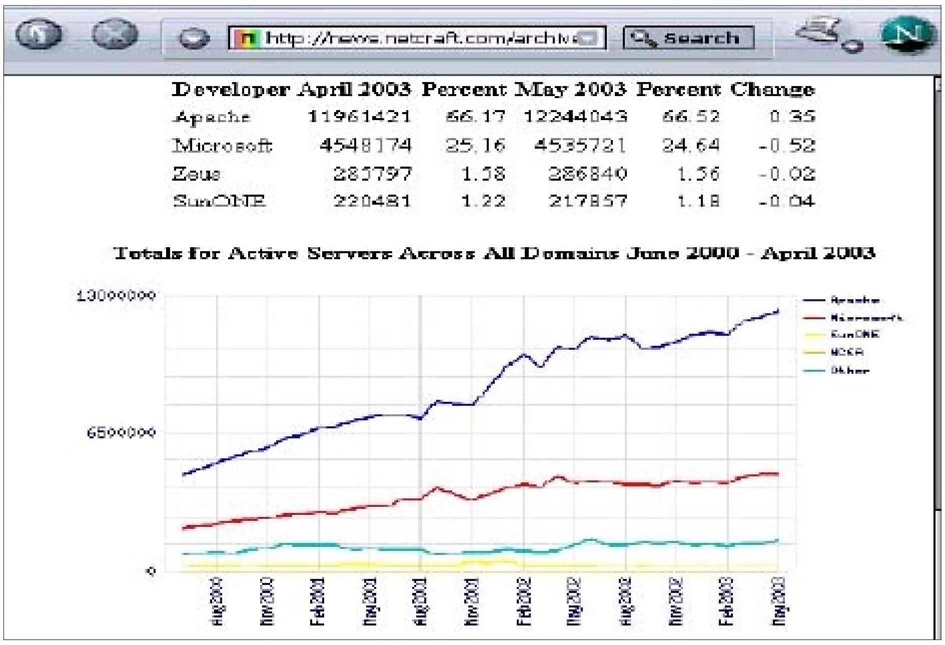
Under the *open source* model you obtain a license which is either free or almost free. This removes licensing cost as an issue when choosing the best software for your application. In addition, you have access to the source code. Examples for open source software are Linux, OpenOffice, PostgreSQL, Eclipse, Struts and basicPortal. Open source will be discussed in more detail in the next section. When choosing hardware, good places to start are http://www.tpc.org andhttp://rubis/objectweb.org/results.html. The latter body rates cost per transaction. An operations environment for a commercial application will typically have many application servers with about 2 gigabytes of RAM each. But it would typically also have only one database transaction server or one database cluster; maybe one more for reporting and/or staging and failover. This is because replication between several databases can be detrimental to performance. Since you are typically limited to one database server, you want to make sure it has maximum memory, such as anywhere upwards from 8 or 16 gigabytes of RAM. Higher memory allows the database to cache more indexes, and thus perform faster.

Up to around 80 per cent of the cost of an application can be in production, so by carefully choosing the production environment you may well be able to help your organization save some real money and be more profitable.

Popularity of Open Source

Open source has increased in popularity over the past years and become a standard in web application development.

A high degree of open source use translates into higher reliability as many people contribute to the code, and virtually instant, global and experienced support.



Developer	April 2003	Percent	May 2003	Percent	Change
Apache	11961421	66.17	12244043	66.52	0.35
Microsoft	4548174	25.16	4535721	24.64	-0.52
Zeus	285797	1.58	286840	1.56	-0.02
SunONE	220481	1.22	217857	1.18	-0.04

Figure 1-1: Popularity of Open Source

Figure 1-1 shows how the open source Apache server software has increased in popularity since the year 2000. One can almost predict when it will achieve a 90 per cent market share.

Reliability of Open Source Software

Many people think that zero or low license cost is the major reason to use open source software. Or maybe not having to go to budget meetings. Or not having to get purchase contracts approved.

Reliability, however, is the true main reason for open source. There are usually so many eyes on an open source software that an issue is likely to be recognizable and obvious to someone and thus resolved fast. Taking the example of operating systems software, the result is that many organizations today consider Linux more secure than Windows.

Access to source code is another feature of open source software which is advantageous to organizations. Without it, organizations are forced to negotiate an escrow for the source code, should the licensor change the business model. Organizations regularly spend large amounts on migrating or rewriting applications because commercial ventures turn out not to be viable or decide to drop support, but do not provide the source code so the organization could maintain the application itself.

Vendors also try to lock you into their support, which could be insufficient.

Support for Open Source Software

In open source, besides the code, the support is also accessible to anyone. Every question ever asked and every answer ever given is open and available for searching. For example, a search on the struts-user newsgroup on "validate date" will return more than 2,000 entries (http://www.mail-archive.com/struts-user%40jakarta.apache.org/).

With the popularity of open source on the rise, chances are that your question has been asked and answered before. The pool of resources is also much larger in comparison with commercial vendors.

In terms of quality of answers, you may also find a difference between open source and vendor support. Vendor support may be staffed with developers just out of college. If you have 10 years of development experience, you may sometimes be frustrated with the answers you get.

Open source support includes some of the most gifted developers you can find. Some may have authored requirements specifications. If you have 10 years of development experience, you may appreciate the help provided by someone with 15 years of experience.

Time is also an issue: open source support is likely to be provided faster than commercial support. Globally, there is always someone online to help solve your problem virtually instantly.

The authors recommend that you ask to see the commercial vendor's newsgroup before you buy a proprietary software or framework.

If not having a commercial vendor makes you feel uneasy, commercial support for open source software is also available. For more information about commercial support for open source software, see for example http://www.basebeans.com or http://www.infonoia.com.

Tomcat 5

Tomcat is a standard servlet container that has average (doesn't sound so well) performance, in many cases it is faster than commercial containers. It works best as a front end, itself answering port 80, no container should be fronted with legacy Apache or IIS. Tomcat and other containers can serve both HTML pages and JSP pages. A container like Tomcat provides services like session tracking, data source connection pools, logging, security, etc. Some people integrate Tomcat with Axis so they can use

SOAP web services. Tomcat has built in single sign on, so that a user authenticated in one WAR does not need to sign in if they go to another WAR. It also comes with a load balancer.

You can deploy a WAR file by placing it in a web apps folder and restarting the container. When placing a container in production, you want to configure the JVM settings.

Testing Sample WARs

A WAR file is a JAR file with some features that are specific to web applications. One of these features is that the WAR package has a directory named /WEB-INF, and that it contains a deployment descriptor XML file at /WEB-INF/**web.xml**.

With popular servlet containers like Tomcat or Resin, you can deploy the WAR file simply by copying it into the /webapps folder of the container.

The authors recommend that you deploy and examine the following example WARs:

- Java Standard Tag Library (JSTL): standard-examples.war
  It shows the use of the standard JSP tags.

- Struts: struts-examples.war
  It shows the use of Struts tags.

- Struts-Tiles: tiles-examples.war
  It has examples of using tiles.

- Struts-Menu: menu-examples.war
  It shows many different ways of using the menu tag.

- Display Tag: display-examples.war
  It shows how to use the display tag.

- Struts-Validation: validation-examples.war
  It shows how to use validation.

The WARs are provided with the *Jasic* suite, but you should always be able to download the latest versions from the Web.

After you have deployed a WAR, browse the examples to get a feel for what you can do with the libraries used in each example.

Creating a Web Application Project in the IDE

A developer can lay out the project file structure in many ways. The authors advocate that you have one project folder per project, which is the project root. Example: /jasic/myproject. It will contain everything related to the project, such as the project file from your editor (.project for Eclipse, *.jpx for JBuilder etc.), source code directories, build directories, etc.

One level down from this project root is the application root. Example:/jasic/myproject/**myapproot**. The application root folder will contain the items that will be included in the WAR that is to be deployed. Often you do not wish to deploy the source code, just the class files. Such class files would typically go to C:/ jasic/myproject/myapproot/WEB-INF/**classes**.

In most cases, the file structure inside /myapproot will mirror the structure of your WAR. This is not a must: advanced editors such as Eclipse allow you to create WAR structures that may be different from the file system layout. They allow you to include files that are possibly shared with another project.

Eclipse 3

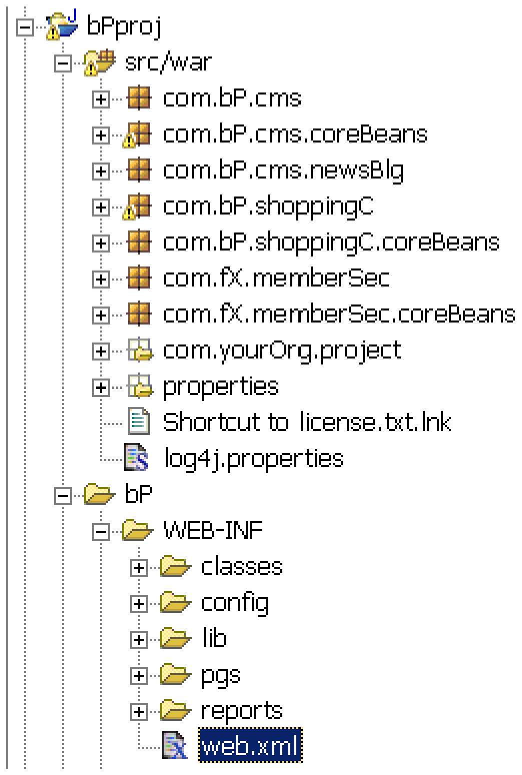
The Eclipse workbench is an open source rewrite of Visual Age for Java. Eclipse runs in Linux, OSX and Windows. The first thing you need to do is create a project. You can do that by clicking New/Project. While coding, you will note that it has code completion features. Also, you can just click on a class or a method and hit F3 to drill down into the code. Eclipse has great support for CVS. It also has a lot of plug ins, such as one that allows you to run Tomcat in Eclipse.
A favorite feature is that it allows you to write macros:
Preferences/Java/Editor/Templates will make you more productive.

You should not need to use ANT builds in day to day development. You know that you are configured right if you can change action, bean or view classes, refresh the browsers and see the effect of new code.

Classpath

During deployment, the class path should be blank. Just by placing a WAR file to webapps, the container knows how to handle it. During development, your IDE needs to be configured to include all the JAR files in the lib folder.

Example project:



```
bPproj
    src/war
        com.bP.cms
        com.bP.cms.coreBeans
        com.bP.cms.newsBlg
        com.bP.shoppingC
        com.bP.shoppingC.coreBeans
        com.fX.memberSec
        com.fX.memberSec.coreBeans
        com.yourOrg.project
        properties
        Shortcut to license.txt.lnk
        log4j.properties
    bP
        WEB-INF
            classes
            config
            lib
            pgs
            reports
            web.xml
```

Most containers have an auto-deployment feature which allows you to test code changes without having to package and deploy the WAR every time. These mechanisms can be great time savers because you do not have to go through the deployment step each time when running or testing your code. These mechanisms typically assume that all files needed in the project are inside the application root (//jasic/myproject/**myapproot**). Here is how to set up Tomcat server.xml, for example, to see your project:

```
<Context path="" docBase="c:/jasic/bPproj/bP"
    reloadable="true" >
</Context>
</Host
where the folder /bP is your web application root (myapproot).
```

Application Server Configuration

While the WAR deployment descriptor web.xml has a standard format, configuring the application server container varies from server to server.

The container provides a Data Source Pool and security. It also needs to know where you placed the application files. The following is an extract from a Resin-specific resin.conf file:

```
<database>
        <jndi-name>jdbc/bppool</jndi-name>
        <driver type="org.postgesql.jdbc3.Jdbc3ConnectionPool">
                <user>bpuser</user>
                <password>changeme</password>
                <serverName>localhost</serverName>
                <databaseName>basicportal</databaseName>
                <max-connections>3</max-connections>
        </driver>
</database>

<web-app id='/' app-dir='\jasic\bPproj\myapproot'>
   <authenticator id='com.caucho.server.http.JdbcAuthenticator'>
<password-query>select passwrd from usr where real_email_id=?</password-query>
   </authenticator>
</web-app>
```

The server configuration files contain many more property entries for data source pool failover, memory, etc. When taking the app to production, you must maximize memory and connections by properly configuring the application server.

Note that having a html server in front of the application server is neither necessary nor recommended. The application server is fast enough for static pages and can also serve dynamic pages.

Review

You should have setup an application server and editor. You should also know how to find independent rankings of price performance of application servers. You must know how to get support and ask questions for open source tools.

Lab A: Install/Unzip

1. TStart with installing a sample development environment that includes IDE and application server.

If you have a slow Internet connection, get a CD. If you have a fast Internet connection, such as DSL or cable, it could take one hour to download. You can download the integrated environment as recommended. If you want to or are using Linux, you can download each component individually.
Download Jasic.zip.

2. Unzip the files to C:\
3. Using the environment variable, set JAVA_HOME to your java location.

Type in java –version. Make sure your system environment path is simple and set to:
%JAVA_HOME%\bin;%SystemRoot%\system32;%SystemRoot%

4. To have JDK 1.4 or higher on your machine, do a search for java.exe in all your hard drives and rename those folders to *.bak. Ex: SunJava1.3.BAK

5. Make sure you do not have memory resident programs running on your machine, then press CNTRL-ALT-DEL.

For higher productivity, do not use Outlook/MS Internet Explorer, but Netscape, for example. You may disable the virus scanner, provided you do not violate a company policy.

Note: Your machine should have more than 256 MB of RAM to be responsive.

6. Optional:

Review the downloads listed on the downloads page of baseBeans.com
Review links listed on the cheat sheet page of baseBeans.com.
Print and study the iBatis DAO and JSTL documentation (both are links in the above-mentioned cheat sheet or download page).

Review windows installation instructions for pgSQL.

Lab B: Deploy Sample WARs

1. Examine the WARSamples folder. It has several apps.
They are in samples WAR files, else download them from the Web.

A calendar, a menu, JSTL examples

2. Copy the listed WARs to web apps folder of the application server.

3. Start the application server. (Tomcat 5 and Resin 3 are provided, t.bat or r.bat should start it, else review app. server documentation).
Make sure you are running the app server on port 80!

4. Using a browser (use non MS/IE browser, for example Netscape/Mozilla), browse to localhost/WAR name of each of the sample WAR files.

Manage your time to allow you to browse each of the WAR files. Explore what they could do. Do not spend more than 10 minutes on each.
Look at how validation works, how tiles work, how display tag work, etc. First by browsing, then examine web.xml and some of the JSPs.
If you can't deploy a WAR, then examine the live samples running on the Web.
Note the menu possibilities and display tag possibilities.

5. Now shut down the application server.

Delete all files from the web apps folder.
The idea is that you should be able to deploy apps. By looking at the features available, you can adopt those ideas and use several examples together.
Example Tiles with menu and display tag.

6. Locate the Struts HTML tag page that has all the docs on the HTML tag.

Locate the JSTL PDF.
Locate the iBatis DB Layer PDF.
Locate the display examples on the web.

Search HTML Q&A mail list archive (Ex: Marc)

Lab C: IDE Setup

1. Start up the Editor, Eclipse is provided.
2. Right click on first item in project.

Note that target is WEB-INF/classes
Examine classpath.
Note that all the jars from lib are there.
Also external files
  Rt.jar (from J:Rockit VM run time for example) and
  servlet2.4 jar from your container (Resin calls its 24.jar, and tomcat calls is servlet.jar), pick one.

A nice things about J2EE is that you can run same app in 2 different containers.

3. Review web.xml.

It should all be familiar to you.

Servlets
Error page
Review java source folder.

4. Optional: review struts-config, server.xml, resin.config, etc. using a text editor (Vi is provided, do not use notepad)

Do not run it yet, it needs a working DB, etc. for JDBC pools/Realms.

Chapter 2: The Requirements Specification

Create business value fast and efficiently

A very important step in any web application development process is to have or create a requirements specification before the development phase begins.

A requirements specification states what the end-users want from the system as outputs. If you write the web application and it has these required outputs, you accomplished what the internal or external client has asked you to do – you created business value.

A good way to lay down requirements for end-users and clients, especially if they are not very IT literate, is to create prototypes, mock-ups and report samples that illustrate the features, look and feel, and navigation of the web application.

A mock-up does not need to be functional, so it can often be built with just plain HTML. Simply showing the mock-ups in a browser invariably helps engage the (prospective) client. It is nice, yet not strictly required to get the navigation working, but it gives the mock-up an extra feel of interactivity. You can also create report samples in PDF. Even end users can do mock-ups to specify what they want as output.

To avoid any misunderstandings that may complicate the developer-client relationship and project delivery, it is important to have mock-ups of *all* screens and reports.

Beware that a requirements specification is neither a design nor a technology; it is about the "what", not about the "how". So, if you received a document that has "Requirements" written on the cover and inside it says: "We will use Struts and EJB, and these are the flowcharts," it is not a requirements specification.

If there are no outputs defined by the client, focus groups, individual interviews and questionnaires may help to identify and analyse the end-users' business needs that the web application is to address.

The requirements specification should be signed off by the internal or external client *before* the construction phase begins. Fixing the application and adding features not foreseen from the start have a habit of delaying project delivery and increasing costs.

Real life case: A bank

The authors had been brought in by a client to recover their project and prevent a delay of project delivery caused by another consulting firm.

One of the authors had been brought in by a client to recover their project which risked to be delayed by another consulting firm. The client said they were to build an employee data entry screen and that they had a requirements specification document.

When the mock-up was presented, the client was very unhappy; that was not what they asked for. Of course, the requirements specification documents were designs, not requirements… and data input is not a requirement… it does not show how the system will get used. While the author socialized with the client, he found out that they wanted to report bank teller fraud to the FBI for prosecution! In fact, what they wanted was an application that outputs all the forms that make it easy for the FBI to arrest a teller that steals. They also had to track the status of this paper work. Of course, after that analysis of real client needs, it was easy to build the application that did this for them. When they saw the new prototype, the client was thrilled.

Reporting with iReports

A good way to see if there are outputs out of your requirements is to examine the reports. You can create mock-ups and later deploy reports using iReports. If there are no reports it is likely that there is no value in the system, and, therefore, it will be hard to justify the costs.

IReport allows you to visually drag and drop fields to a GUI tabular WYSIWING editor and see the results.

Return on Investment (ROI)

Are mock-ups all that you need for a requirements specification? No, there is an extra item that should be in the requirements specification: value!

You should understand how the outputs of the system you are about to develop and put in operation save or make the organization money.

It is the system outputs that create the value. Some examples:

- online ordering system leading to cost and headcount reduction potential on sales force and accounting
- increase in revenue per click-through with advertisements displayed on a page
- increase in customer loyalty and reorder levels through the new portal

Sometimes, the value is not realized until the application has millions of users.

As an independent consultant, you may have to justify your budget and demonstrate that your project will be financially successful. These are the formula to observe

PROFIT = VALUE OF THE SYSTEM - (COST TO DEVELOP + COST TO OPERATE)

ROI = (PROFIT / INVESTED CAPITAL) X 100

If you have a project with an ROI that is less than 100 per cent, it means that it costs more than it is worth. Such a project should be abandoned. An ROI of 300% means that you triple your money over the period calculated (say, five years).

In a break even analysis, you find out how many months after the investment the invested money is recouped. Insert FORMULA

If the ROI of a project is good, and this is understood by the organization, it is extremely unlikely that the project will get canceled.

Lab D: Mockup

1. Start up an html editor of choice. (Vi, TopStyle, Netscape, Eclipse)
2. Create a file TaskEdit.html (you can use a similar name for the file that you like). Create this file in \jasic\bPproj\bp. (same folder that contains WEB-INF).

Requirements! Think of what a task html form should have on it?

It should have fields that are used to edit/update a task.

For example taskName, Status, AssignedTo, description, a few more fields that you think a task tracking software should have. Be creative..

4. You could hard code values of the fields.

For example:

```
<td> Task Name : </td> <td> Create an army of clones </td>
```

etc.

Create a 2nd file taskList.html (again a similar file name would do).

In it create only taskName and maybe 1.. or 2 more fields, repeating one under each other.

```
<tr><td>Create an army</td></tr>
<tr><td>Attack the closest star ship</td></tr>
```

etc.

This is your mock up.

5. Start up application server.
6. Using a browser, navigate to: localhost\taskName.html and then localhost\taskList.html.

You can start up IDE (Eclipse) to touch up the files if needed.

Note that your container has or should have lines like this that points your editing enviroment as “/”:

```
<Context path=”” docBase=”/jasic/bPproj/bP” debug=”0”
reloadable=”true” crossContext=”true”>
</Context>
</Host>
```

Chapter 3: Database Access Setup

Data Base

One of the challenges and key success factors when building a web application is to make sure that you put each piece of code “where it belongs”. This helps the development team effort by making communication about the code easier. It also makes the code more readable and therefore maintainable for the future.

For car design, you will probably agree that it helps project participants if they “speak the same language” on construction techniques. This “construction lingo” is a layer above “English”. It is the same for web applications: There is also an important layer of communication above the base language Java/J2EE.

For web applications, the commonly accepted methodology to use is *Model-View-Controller* (MVC). Simply speaking, “Model” represents the data encapsulation. “View” is what the user sees, the user interface (UI) and “Controller” is the mechanism to control the application flow.

Struts is an implementation of the MVC methodology. If you write code using Struts, you have found your construction language. It is a language that you share with some of the most gifted web application developers you can find.

The authors sometimes say that Struts is like a kindergarden: “You put your coats here. You put your shoes there. Toys are over here. So, now we know where to find things.”

In software development, there is a great benefit in knowing where to put things, and find things that other developers have written. We put our data layer here, we put our view presentation there. And we put the controller code over there. The result is that everyone knows where to find a module that they are looking for, be it related to application flow, to data or to the application look and feel. This is true MVC.

PostgreSQL 7.5

The database pgSQL is known for its high volume transaction support and stability. It offers significant cost savings on staffing and the authors find that it has better support than proprietary vendors.

It supports full ANSI SQL, including stored procedures, and full Joe Celko designs, such as correlated joins, etc. www.sf.net is an example of a large site based on pgSQL.

Designing the Model

Once the requirements specification has been approved, you can proceed to analysis and design. Based on your analysis of outputs and how the application will be used, you can design a data model.

This should be done by the person with the most production experience you can get. Depending on the size of the application, the authors have specified a minimum of 7 (seven) years experience in bringing applications to production. Of course, if your application will only have to support twenty concurrent users, a good schoolbook design may be sufficient.

You analyze report fields and how they are computed. You must ensure that you can aggregate that data from the fields in your data model. When you find fields in data entry screens that are not used anywhere, remove them. There is no point in building and testing screens if they are not needed; aim. aim not to build them in the first place.

If the web application needs to be scalable to hundreds or thousands of concurrent users, you need someone with performance-tuning experience.

If you are not allowed to touch the model because it is pre-existing, you should still do your design. If there is a mismatch between the existing model and the requirements specification, you can create the mismatched tables in the staging area.

Generally stay away from using constraints in the database because they create issues for development and performance.

Also, your models should be normalized in that all tables have an auto-incremented sequence field that does not depend on any other fields.

Creating an Entity-Relationship (E/R) Diagram

Once the design is completed, we should document it for our developers using an E/R tool of our choice. With *Open DataBases Connectivity* (*ODBC*) or *Java DataBase Connectivity* (*JDBC)* connection to the database, the E/R tool allows to generate diagrams from the database structure. Any tool would do, such as *Microsoft Visio*.

The only document developers tend to refer to a lot is the E/R diagram. This is the design of our application. Try to make sure it does get treated with the confidentiality that it merits.

JNDI DataSource Configuration

To allow your application to access the database using JNDI, you use container-specific syntax. For example, in the case of Tomcat, you would add something similar to the following in the “context” section of server.xml:

```
<context …>

<Resource name="bppool" auth="Container"
type="javax.sql.DataSource"/>

<ResourceParams name="bppool">


<parameter><name>url</name>

    <value>jdbc:postgresql://localhost:5432/bp?autoReconnect=true</
value> </parameter>

    <parameter>  <name>driverClassName</name>
    <value>org.postgresql.Driver</value> </parameter>

    <parameter>  <name>maxIdle</name>         <value>3</value>
    </parameter>

    <parameter>  <name>username</name>  <value>bpuser</value>
</parameter>

    <parameter>  <name>maxWait</name>        <value>5000</value>
</parameter>

    <parameter>  <name>maxActive</name>       <value>10</value>
    </parameter>

    <parameter>  <name>password</name>
    <value>changeme</value>      </parameter>

    <parameter>  <name>removeAbandoned</name>
<value>true</value>  </parameter>

    <parameter>
```

For Resin resin.conf, the following would have the same effect:

```
 <database>
    <jndi-name>jdbc/bppool</jndi-name>
    <driver type=”org.postgesql.jdbc3.Jdbc3ConnectionPool”>
          <user>bpuser</user>
          <password>changeme</password>
          <serverName>localhost</serverName>
          <databaseName>basicportal</databaseName>
          <max-connections>3</max-connections>
    </driver>
```

```
</database>

You can test your pool via a simple servlet such as this:
public class XServlet extends HttpServlet {
protected void doPost(HttpServletRequest req, HttpServletResponse
res) {
   PrintWriter out = null;
   Connection conn = null;
   try {
        out = res.getWriter();
         res.setContentType("text/html");
         Context ctx = new InitialContext();
         if (ctx == null)
             throw new Exception("Boom - No Context");
          DataSource ds = (DataSource)
ctx.lookup("java:comp/env/bppool");
           if (ds != null) conn = ds.getConnection();
           Statement stmt = conn.createStatement();
            ResultSet rs = stmt.executeQuery("select * from usrs");
            if (rs.next()) {
                foo = rs.getString(2);
            } //if
            stmt.close();
            conn.close();
            out.println("DB Con. data is: " + foo);
```

Lab E: SQL Install

1. http://techdocs.postgresql.org/guides/Windows

Read the documentation above and follow instructions to install pgSQL (or a SQL engine of your choice, pgSQL is recommended). PostgreSQL is a fast and free ANSI SQL open source DB with support for stored procedures, etc.
If you can't get pgSQL to start, search questions asked by other people **in the postgresql.org** newsgroups and mail list on the step you are stuck on. (Cygwin should be avoided for pgSQL, it's slow and not stable)
You should know how to start it up and shut it down and login.

2. Install a db admin tool, such as pgAdmin (there are others, such as EMS manager).
In pgSQL folder you can right click on pgadmin file to install it
3. Using the admin tool, create a db, call it "bp".
4. Using the admin tool, create a user, call it "bpuser" with a password of change me.
5. Using a file manager, locate the create table sql script in (fX) docs folder. Review the sql, (tables created, permissions, etc.).

6. Using the db admin tool, access the created db. Click on the new bp db.
In pgAdmin, there is a sql tool (icon looks like a 3 headed monster), click on it. Paste in the create SQL script. Click execute.
7. In docs, there is also a sample data script, execute that as well.
   Review Tomcat server.xml and resin.conf files. Notice how the db connection data source pools are configured.

   Configure the connection DS Pools

8. Optional
You can now connect to this db using…. Excel, or any ODBC tool (PowerBuilder)
Also, you could use Squirrel SQL client (or other JDBC tool) to connect to it using JDBC and be cross platform. The optional step is to connect to the DB using Eclipse jFaceDB plug in.
- Right click on Connection to create a new connection.
- Now you can explore the db in Eclipse as well. Mostly you will be using the pgAdmin tool

Lab F: Model Design

1. Create a file design.sql.

I use gVim.org.

2. Examine your mock up html files.

Based on the analysis, create a table design in a text file that will allow you to store those fields. (Do not use date fields for now, we are just doing Struts. There is lab support built in for int.)

Ex:

```
CREATE TABLE tasks (
id        serial NOT NULL UNIQUE PRIMARY KEY,
name            varchar(80)       NOT NULL ,
task_status     int,
…
);


GRANT ALL ON tasks    TO bpuser;
GRANT ALL ON tasks_id_seq  TO bpuser; -- a quirk in pgSQL needs
rights to id seq
```

3. Execute the SQL script
4. Using the DB admin tool, enter at least a few rows of data.

The more data the better.

5. Using the DB admin tool, issue a sql query.

Ex: Select * from tasks where task id = 2

6. Optional: Do explain sql, so you can see the plan on how the query will be executed.

This would be important for scalability down the road.

7: Optional: Connect to db using a JDBC tool (SQUirel client or jFace)

(Optional) Lab G : Output/reports

1.Start up iReports (ir.bat)
Read up the documentation on iReports on iReports.sf.net.

2. Configure the JDBC, and class path in the iReprots.

An easy way it so copy all the jars from web-inf lib folder, and place each in the path. Important are JDBC driver for pgDB, XML related jars, commons related jars, and itext.

3. Test the connection.

If you can't connect using JDBC here, the rest of the labs and examples will not work for you.

3. Configure the DB connection and ping. User id is bpuser.
Use the menu to set active connection.

6. Use the wizard to create a "report".
Save it as tasks.xml or something similar.

7. Now examine the xml file, such asgVim.org.

8. Read up documentation on JasperReports.sf.net.

You can drag bands around and drag fields or other elements around in iReport.

9. Now compile the report so you get a PDF output.

10. Massage the report until it looks nice. You can examine some of the more advanced features such as subtotals, grouping of bands, etc.

Remember, outputs are key to value, so make the report useful.

IReport is a band based report tool.

Chapter 4: Using Simple Actions

The “C” in Model-View-Controller

The Controller is the mechanism that manages the application flow. In Struts, this management role is taken by a central action servlet. The rules for the application flow are not hard-coded in Java, but defined in a configuration file often called struts-config.xml. The action servlet follows the rules defined in this configuration file.

Using Action Mappings

The first thing we do when we start writing a module is to create its action mapping in struts-config.xml. The following is an example of a mapping in the <action-mappings> section:

```
<action path="/contentAdminAct"
          type="com.basebeans.basicPortal.cms.ContentAdminAct" >
    <forward name="Success" path="contentAdminLst.jsp"
          redirect="true" />
    <forward name="Edit" path="contentAdminEdit.jsp" />
    <forward name="Saved"
          path="/do/contentAdminAct" redirect ="true" />
</action>
```

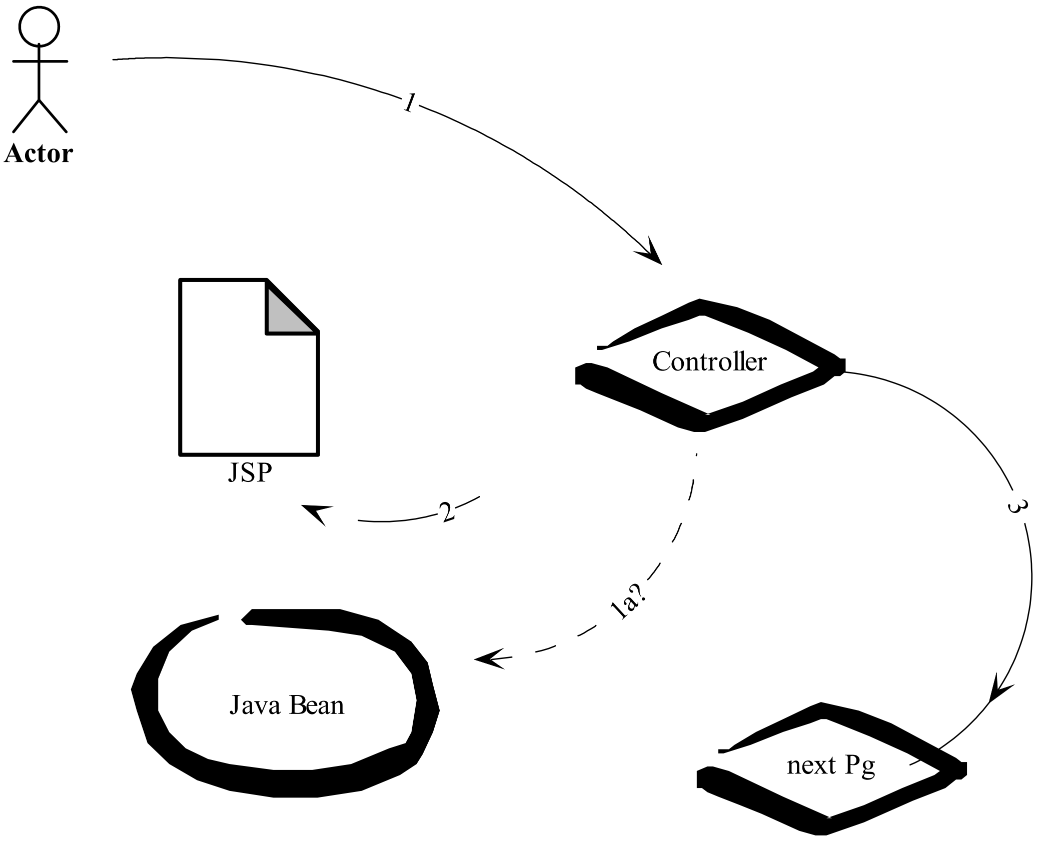
The action path attribute is what we call the action within the browser. In the example, this may be a URL like http://localhost/do/**contentAdminAct**. The action type attribute is the implementation class for the action being called.

The action class will implement some business logic, such as populating a bean, or validating data. Based on the outcome, the action class decides where to forward to. In the example, the action forwards to a list of contents via contentAdminLst.jsp which the user should see. However under certain circumstances (i.e. user has requested to edit an item), the action forwards to the detail data entry screen contentAdminEdit.jsp . The action class can use any of the forwards defined in the XML, in this case Success, Edit and Saved.

If we want more places to go to, we just add more forwards. A forward will either go to a JSP, or to another action.

Typically, there is only one action that calls a specific JSP. In the example, there will be no other actions that go to contentAdminEdit.jsp than /contentAdminAct. If another action wishes to also display contentAdminEdit.jsp, it should forward to /do/contentAdminAct. This means it has to call the controller.

3 Rules for Structuring Action Mappings



Action mappings allow you to dispatch to any action class you want and to forward to any JSP or to any other action URL. This gives you a lot of freedom, and if you are not careful, you could easily end up with a jungle of mappings that follow no discernable pattern. Knowing how to use the controller right makes 80% of a successful Struts application.

Typically, a page should have one principal action mapping, such as /mypageAct, per page. You can consider this action mapping the "controller" of that page.

Then there are three simple, yet important rules to follow:

1. Always call the controller to get to a page.
2. When submitting from a page, submit to its controller.
3. An action forward maps to where a controller can go: to its JSPs or to another controller.

If these rules are not clear to you yet, they will become clearer as you continue reading. In the future, whenever you get confused about your application flow, refer back to these important rules. I sometimes say that if you know this, then you know 80% of Struts.

Writing an Action Class

The following example shows an action class:

```
public class ContentSrchAct extends ... {

public ActionForward execute(ActionMapping mapping, ActionForm
form, HttpServletRequest req, HttpServletResponse resp) throws
Exception {

    log.debug("srch action called");

    return (mapping.findForward("Success"));
} //exec
```

This is what will get executed when the action mapping is called. The most important part is the return of the forward that is mapped.

Also, when we first write an action we have a debug message sent to the console. This way we make sure that the action is really called and that the mapping is correct.

Configuring the Struts Action Servlet

The way the Struts framework gets kicked in is via a servlet call. The Struts action servlet checks the action mapping behind the scenes. The following is an extract of web.xml configuration that registers the Struts controller servlet and maps it to an (action) URL pattern.

```
<servlet>
    <servlet-name>action</servlet-name>
    <servlet-class>org.apache.struts.action.ActionServlet
    </servlet-class>
    <init-param>
          <param-name>config</param-name>
          <param-value>/WEB-INF/config/struts.xml</param-value>
    </init-param>
</servlet>
<servlet-mapping>
    <servlet-name>action</servlet-name>
    <url-pattern>/do/*</url-pattern>
</servlet-mapping>
```

This means that the Struts framework will be invoked when the container receives any request (URL) that matches the /do/* pattern, such as in http://localhost/myapp/**do**/myAct. The servlet will then examine the remainder of the URL and call the appropriate action as defined in struts-config.xml. In this case, the action mapped as /myAct would be called. You do not have to call it "do" you can call it "mo" or whatever.

Review: You know how to structure Struts actions and action mappings

Lab H: Simple Action

So far we have a mock up and design of what the user want. Lets start to turn it into a prototype.

1. In bp\WEB-INF\pgs create a folder called tasks (or a similar name) using Eclipse.

2. Move the task*.html files to there. (Might be easier to do that with a file manager.)

3. Rename *.html to *.jsp.

4. In Eclipse, right click on the project and click refresh.
5. In source folder (where com.bP is), using the IDE, create a package com.myOrg.tasks.

Use a made up name of an organization or a company name instead of myOrg.

6. Edit the struts xml file. (You can open web.xml file and find where struts-xml file is)

7. In the action mappings, create 2 mappings. (You can just clone another mapping 2 times. Towards the top of mappings will make it easer for you to find it)

One to forward to taskList.jsp and one to forward to taskEdit.jsp.
Call the mapping that forwards to taskList.jsp "tasks".
Call the mapping that forwards to taskEdit.jsp "taskEdit"

Look at the other example in there. Do not name a form bean.

Have a forward called "Success" for each that forwards to the respective pages.

8. Create 2 action class files in the "com.myOrg.tasks".
Each will look like this.

```
import org.apache.struts.action.*;
import javax.servlet.http.*;
public class XXAct extends Action {
public ActionForward execute(ActionMapping mapping,
                 ActionForm formBean, HttpServletRequest req,
                 HttpServletResponse resp) throws Exception {
System.out.println("I am in action for XYZ"); // or use log.info if
you are expert
   return (mapping.findForward("Success"));
    }//exec
}
```

Call the one that lists TaskAct and the one that Edits TaskEditAct. Make sure action mappings point to these actions.

9. Now start the container.

10. Using a browser, navigate to localhost/do/tasks and then do/taskEdit.

Look at the output of the console, did you get the message: "I am in action . . ." That means you called the action right.

Always have a message that you called the action right first, so you know you are calling the right action.

11. Did it return the expected page?

If you change the struts-config, you will have to restart the container.

12. Once you have it working on Tomcat, shut it down. Start up Resin and browse to the 2 actions.

13. Optional: Modify log4j properties in source properties to include your log output.

Ex:

log4j.category.com.myOrg.tasks=DEBUG

The com.myOrg.tasks should map to your package name that you want to set a debug level to.

Note that you should change the log4j file in source folder. Any changes that you might make in the classes folder will get overwriten the next time you do a build.

Changing the log4j file in /classes will have it overridden next time you compile.

System.out is a bad practice and you should use log4j ASAP.

Chapter 5: Using JSPs and View

Integrating JSP Tags

There are a lot of tag libraries available that we can use. The most important one is the Java Standard Tag Library (JSTL). This tag library has several groups of tags. The tags from the core © tags group help us to display data. The tags from the formatting and internationalization group (FMT) help us to localize output. For data entry on HTML forms, we use tags from the Struts-HTML tag library.

In the following fragment of a typical JSP with form input, see how the different tag libraries are declared and used to display a combination of text labels and input fields.

```
<%@ taglib prefix="fmt"     uri="/WEB-INF/lib/fmt.tld" %>

<%@ taglib prefix="c"       uri="/WEB-INF/lib/c.tld" %>

<%@ taglib prefix="html"    uri="/WEB-INF/lib/struts-html-el.tld"
%>

<%@ taglib prefix="display" uri="/WEB-INF/lib/displaytag.tld" %>

<fmt:setBundle basename='properties.messages_en-US' scope =
'session' var  = 'loc' />

<html:errors />

<html:form action="/userJoinAct" >

<table border="0">

<tr>

    <td class="text_label" ><fmt:message      key="user.lastName"
bundle="${loc}"/>: </td>

    <td class="form"><html:text              property="lastName"
size="25"/> </td>


    <td class="text_label"><fmt:message       key="user.firstName"
bundle="${loc}"/>: </td>

    <td class="form"><html:text              property="firstName"
size="10"/> </td>

</tr>

</table>
```

```
<html:submit property="Dispatch" value="Save"/>
</html:form>
```

JSP 2.0 and JSTL1.1

JSP 2.0, which runs on application servers that support Servlet 2.4, such as Tomcat 5 and Resin 3, comes with a number of new features or improvement versus the previous version. A notable one that can help you simplify your JSPs it that you can simply write the following:

```
${fb['title']} or
${fb.title}

instead of in Servlet 2.3 JSPs:

<c:out value="${fb['title']}"/> or
${fb.title}
```

The authors recommend that you read the specifications document for the JSTL tag library "JavaServer Pages Standard Tag Library, Version 1.1" available at the www.sun.com The above examples declares the taglibs according to JSTL 1.0. To use JSTL 1.1, web.xml and the taglib declaration URLs have to be modified; see the specifications document for details.

Localization

The above JSP fragment used FMT tags to obtain the content of data labels from a properties file. Using the FMT tags allows you to avoid hardcoding text content in JSPs.

You can specify which properties file to draw from with FMT tags as follows:

```
<a href ='?lang=hr'>Hrvatski</a><br>
<a href ='?lang=en'>English</a><br>
```

```
<fmt:setBundle basename='properties.messages_en-US' scope =
'session' var  = 'loc' />


<c:if  test="${param.lang == 'hr'}">
    <fmt:setLocale value = 'hr'  />
</c:if>
<c:if  test="${param.lang == 'en'}">
    <fmt:setLocale value = 'en-US'  />
</c:if>


<fmt:message key="user.lastName"  bundle="${loc}"/>
```

This assumes that your project contains a properties file at WEB-INF/classes/properties/messages_en-US.properties . which will likely have been created in src/properties/ .

This way, if we want to change a text message that is to be displayed throughout several pages, we have one place only where to make the change. With this mechanism an application can be maintained much more easily.

Using resource bundles also allows you to localize your application easily. You could add another file such as messages_**de**.properties for labels in German. By specifying the locale as **de**, the labels will display the labels from that file instead. Rather than specifying the locale in each JSP, you would likely set and modify the locale variable (which is available anywhere in the session) on a central page.

Your page should not have any text labels on it, it should be completely localized. This is sometimes called word oriented programming.

In the file properties/messages_xx.properties, you would have the following:

```
#user
user.firstName=First Name
user.lastName=Last Name
```

In order to localize some Asian languages, you also have to set the encoding using JSTL.

To localize messages you want to set in action you can do this:

```
//import org.apache.commons.resources.*;
  public void execute() {
  …
  Locale locale = new Locale("en","US");
  Messages messages = Messages.getMessages( "properties.messages");
  String msg = messages.getMessage(locale,"user.firstname");
  …
  }
```

Both view localization and action localization require properties files such as this:

```
user.firstName=First Name
```

All the properties for each language are duplicated in files that end in en-US, etc.

Layout With Tiles

Struts includes a feature that assists with the consistent and centralize layout of the application. This feature is called Tiles. It allows you to create template layouts that you can reuse on multiple screens. A template layout defines the structure of a page, with placeholder variable tags for dynamic “tile” content. Any JSP that can reuse content from other JSPs can be turned into a layout.

This is an example of a layout JSP:

```
<% taglib prefix=”tiles”
uri=”http://jakarta.apache.org/struts/tags-tiles” %>
<table width=”100%” border=”0”>
<tr>
    <td> align=”center” colspan=”2”>
          <tiles:insert attribute=”header”/>
    </td>
    <td> width=”220” align=”center”>
          <tiles:insert attribute=”header_right”/>
    </td>
</tr>
<tr valign=”top”>
    <td width=”150” align=”center”>
          <tiles:insert attribute=”leftBar”/>
    </td>
    <td> align=”center”>
          <tiles:insert attribute=”body”/>
    </td>
    <td width=”220” align=”center”>
          <tiles:insert attribute=”adBar”/>
    </td>
</tr>
</table>
```

Note how the <tiles:insert ...> tag is used to create placeholders. HTML tables, columns and rows are used to define the structure of such a template page.
The combinations of runtime values for the attribute key of each placeholder are specified in layout definitions. Definitions specify which actual “sub-”pages to insert when building a page on the basis of a layout. Not unlike struts-config.xml, these definitions are stored in an XML file, such as /WEB-INF/config/layouts.xml.
You activate the tiles feature by configuring it as a plug-in in struts-config.xml, and adding the tiles tag library to /WEB-INF/web.xml. The following is an excerpt of struts-config.xml:

```
<struts-config>
...
    <plug-in className=”org.apache.struts.tiles.TilesPlugin”>
          <set-property property=”definitions-config”
                value=”/WEB-INF/config/layouts.xml”/>
          <set-property property=”moduleAware” value=”true”/>
    </plug-in>
</struts-config>
```

As usual, make sure that the tag library definition file (.tld) exists at the taglib-location you specify.

Creating Tiles Definitions

To use a tiles layout, you need one or several tiles definitions which specify the runtime values for the tiles. The following is an excerpt from the definitions XML file /WEB-INF/config/layouts.xml which the above tiles plug-in in *struts-config* referred to:

```
<definition name=”baseDef”
     template=”/WEB-INF/pgs/common/myLayout.jsp”>
<put name=”header” value=”/WEB-INF/pgs/common/myHeader.jsp”/>
<put name=”header_right”
     value=”/WEB-INF/portlets/common/myHeaderR.jsp”/>
<put name=”leftBar” value=”/WEB-INF/pgs/common/myMenu.jsp”/>
<put name=”body” value=”/WEB-INF/pgs/Home.jsp”/>
<put name=”adBar” value=”/WEB-INF/portlets/pgs/myAdBar.jsp”/>
</definition>

<definition name=”youPageDef” extends=”baseDef”>
<put name=”body” value=”/WEB-INF/pgs/myPage2.jsp”/>
</definition>
```

Note how the definition named mypage2Def extends baseDef. This means that it will have the same header, leftBar etc. and will use the same layout template as baseDef; the only difference is that it will have a different body.

Using Tiles Forwarding

Following MVC, in most cases you will not call the content JSP directly from the browser, but go through an action which forwards to the JSP. Struts allows you to forward directly to a definition rather than to a page URL.

So in struts-config.xml, instead of specifying a forward as:

```
<forward name=”Success” path=”/Homepage.jsp”>
```

you would specify the forward as follows:

```
<forward name=”Success” path=” yourPageDef”>
```

Note that there is no leading slash (/) when calling the definition directly. Following the above definitions example, another forward would use page2Def as path.

You are free in how you name definitions. Since each definition represents an actual, if dynamic, screen page, the authors often use the pattern mypageDef when naming definitions. So, definition names would be baseDef, or page2Def.

CSS

One way to use CSS is to use name a style name for each cell like this:

```
<td class="text">
```

You can then declare a skin1.css as:

```
.text            { font-family: Verdana,Arial; font-size: 8pt;  valign: top; color: black;  }
.text_label { font-family: Verdana,Arial; font-size: 8pt;  valign: top; color: black;
                                                 font-weight: bold; }
```

You could create a skin2.css with different look and feel.

In general, most people tend to support only the newer browser standards, to reduce complexity.

One way to write up a good style sheet is to manually develop one web page, and based on it create a CSS. The rest of the pages would then use class ids from the style sheet. All look and feel should be in a style sheet and each type of a td cell should have a CSS class associated with it.

Sometimes, people add CSS styles for JavaScript browser events to give it a more dynamic look, by you change the properties tag style. You should spend time making you site look nice.

There were previously no standards for naming CSS classes; however, the OASIS Web Services for Remote Portlets (WSRP) standard 1.0 proposes a naming standard for components of portal applications (“portlets). The JSR-168 Portlet standard reuses the OASIS naming proposals.

You may already want to follow the OASIS/Portlet CSS naming conventions to avoid unnecessary rewriting of JSPs should in the future you wish to make application modules fully JSR-168- or WSRP-compliant. The WSRP specification is available for free and without registering at the WSRP TC website of oasis-open.org. The Portlet specification is available at ***www.sun.com***

Skin-able Style and Branding

The look and feel of a web application should be centralized as much as possible using Cascading Style Sheets (CSS). If not writing your own stylesheet, possibly following the Portlet standards, , you may be able to reuse existing style sheets that have been created by a graphic designer according to a corporate standard. When you have access to a graphic designer, he or she should be able to provide a CSS style sheet with his or her designs and mock ups.

To reuse a style sheet across pages, you would add the CSS file to your application somewhere outside /WEB-INF, such as /myapproot/scripts/mystylesheet.css. Then, in your tiles layout JSP, as part of the HTML head tag, add the reference to this external style sheet as follows:

```
<%@ taglib uri=”tiles” prefix=”tiles” %>
<html>
<head>
...
<link rel=”stylesheet” href=”../scripts/mystylesheet.css” type=”text/css”>
<title><tiles:getAsString name=”title”/></title>
</head>
<body>
...
```

If you want to offer users the possibility to choose between different style sheets or “skins”, you could store the name of the preferred style sheet in the user session and retrieve it from the session using the <c: tags. Your layout would then look as follows:

```
<%@ taglib uri=”tiles” prefix=”tiles” %>
<%@ taglib uri=”jstl/c” prefix=”c” %>
<html>
<head>
...
<link rel=”stylesheet” href=”../scripts/<c:choose><c:when test=”${!empty
stylesheetkey}><c:out
value=”${stylesheetkey}”/></c:when><c:otherwise>defaultstylesheet.css</
c:otherwise></c:choose>” type=”text/css”>
...
```

On a personalized settings page, you could offer the list of stylesheet choices in a Struts select tag as follows:

```
<html:select onchange=”document.forms[0].submit();” property=”userCss”>
<option value=”choose”>Choose your skin</option>
<option value=”skin1.css”>Classic</option>
<option value=”skin2.css”>Modern</option>
<option value=”skin3.css”>Funky</option>
<option value=”defaultstylesheet.css”>Default</option>
</html:select>
```

The form action would then have to retrieve the userCss property from the form bean and store it in the session.

Look and feel of the application is very important. When the client says the application is good, they usually mean that it looks good, and not that you used Struts.

Menu Navigation

In many web applications, navigation code is found all over the place, and thus very difficult to maintain. The Struts-Menu plug-in with struts-menu.jar and struts-menu.tld offers a standardized and easily maintainable way of creating a menu navigation.

Struts menu tags allow different presentation styles for menus, such as horizontal, vertical, based on security, drill-down, using Javascript, etc.

Like Tiles, the Struts Menu is a plug-in defined in /struts-config.xml, with the TLD in /WEB-INF/web.xml.

The following is an excerpt from struts-config.xml:

```
<struts-config>
...
        <plug-in className=”com.fgm.web.menu.MenuPlugIn”>
                <set-property property=”menuConfig”
                        value=”/WEB-INF/config/navigation.xml”/>
        </plug-in>
</struts-config>
```

Displaying Menus

A JSP to display a struts menu could look as follows:

```
<% taglib prefix=”menu” uri=”menu” %>

<menu:useMenuDisplayer name=”Simple” >
        <menu:displayMenu name=”myMenu1”/>
</menu:useMenuDisplayer>
```

According to the plug-in set-property, the menu structure would be defined in /WEB-INF/config/navigation.xml. This would be an excerpt of navigation.xml:

```
<Menu name=”myMenu1” title=”Resources”>
        <Item name=”ref” title=”Cheat Sheet” toolTip=”Cheat Sheet”
                page=”/do/cmsPg?content=CHEAT_SHEET”/>
        <Item name=”downloads” title=”Downloads” toolTip=”Downloads”
                page=”/do/cmsPg?content=DOWNLOAD”/>
</Menu>
```

See how it contains a list of items that we want to display, the items, names and what URL to go to. The name of the menu corresponds to what we use with the menu:displayMenu tag to get the items rendered.

A number of versatile menu dis players are provided with the struts menu library. In the above example, the menu displayer Simple has been used.

So the steps are:

1. User navigates the menu
2. Menu calls the action
3. Action forwards to tile definition

Note: Struts logic and bean tag are planed for deprecation, so we do not use them, they are replaced by JSTL tags. We do use display tag later in the book.

Example of a struts tab menu:

JavaScript

The following is an example for using JavaScript for client-side procession of the User Interface_

```
<FORM NAME="aForm">
    <INPUT TYPE = "button" NAME="aButton" onClick='myinfo()'>
</FORM>
<SCRIPT LANGAGE ="JavaScript">
function myinfo() {
alert("Browser is " + navigator.appVersion + " " + navigator.appName )
```

```
    document.getElementById("aButton").value="Done!";
  }</SCRIPT>
```

In the event of a user click, it pops up an alert. You can access the one form element like this: document.getElementById("aButton"). This is known as a browser DOM.

You can do a lot of sophisticated things using DOM. Use the current DOM 1 or DOM 2, which is not the same as Level 0 DOM or the intermediate proprietary DOMs that should not be used.
You should make your pages look nice and dynamic and use the client side browser's CPU for UI. Using the browser DOM allows you to reduce the load of the server and makes your application more scalable.

Review

You should now know how to use JSTL, Struts Menu, Tiles, Localization and encoding. You should know that Look and Feel are important.

Lab I: Layouts & Navigation

You currently have 2 JSPs.

1. Replace all text labels with word oriented programming.

Ex:

<fmt:message **key=”task.name”** bundle=”${loc}”/>

2. Make sure you declare all needed tags that you use at the top of the jsp.

Ex:

<%@ taglib prefix=”fmt” uri=”/WEB-INF/lib/fmt.tld” %>

and in this case:

<fmt:setBundle basename=’properties.messages_en-US’ scope = ‘session’ var = ‘loc’ />

3. Add the text in source folder of resource message properties.

Ex:

task.name= Task Name

When you build, they will be placed in classes location.

You should not have any text labels left on the page.

Test to see it work. (Start the container and navigate using a browser to the pages)

4. Edit the tiles layouts definitions file in config folder.

Create 2 new definitions, ex: taskLst.def and taskEdit.def but any name will do. Start by cloning another entry (towards the top of layouts)

Ex:

<definition name= “taskList.def”

extends =”baseDef”>

<put name= “body” value=”/WEB-INF/pgs/tasks/TasksLst.jsp”/>

</definition>

Make the definitions extend baseDef, like others.

5. In action mapping, the forwards currently go to JSPs.

Change them to go layout definitions. (taskList.def in one case for example)

6. Test. (Start the container and navigate using a browser to the pages)

Note: In later labs, if you have problems with the JSP (ex: forward error), you will have to change the mapping back to jsp until the bugs in JSP are fixed; and after it works, revert the mapping back to layout definition).

7. In navigation file in config, edit mMain1 menu.
8. Add a menu items to go to /do/tasks Ex:

```
<Menu   name=”mMain1”                   title=”baseBeans”>


<Item  name=”tasks”   title=”Task List”
 toolTip=”Task list”  page=”/do/tasks”/>
```

9. Test.

Lab J: Skin-able Branding Style

1. Change all cells in JSP to have Style.
<td class=”text”>
…
</td>
2. Create or download some nice narrow (small) jpg or gif images, or find some nice logos on the web. Copy them to the images folder.

3. In portlets/common, edit the header and footer jsp files.

Replace the images of baseBeans, etc. and place your image instead.

You should have no original images left on the page.

4. Demo the app.

5. Optional: Edit the CSS file in CSS folder.

Chapter 6: Using Simple Beans

What is a Java Bean?

A Java Bean is simply any class that has private properties and public getters and setters. The following is an example of a simple bean:

```
public class MyUserBean {
        private String userName = null;

        public void setUserName(String _userName)          {
                userName = _userName;
        }
        public String getUserName()  {
                return userName;
        }
        ...
}
```

This means that when you want to retrieve a value from a bean instance, you would do it as follows:
String myuserName = myjavabean.getUserName();

Beans are a very popular means to encapsulate data. We will use Java beans and Java Collections to encapsulate data.

Form Example

The following is an HTML form with Struts JSP tags:

```
<html:form action=″/userEdit″ >
<tr>
    <td class=‶text_label″>
          <fmt:message key=″user.id″/>:
    </td>
    <td class=″form_input″>
          <html:text property=″userId″ size=″10″ />
    </td>
</tr>
<tr>
    <td class=‶text_label″>
          <fmt:message key=″user.userName″/>:
```

```
        </td>
        <td class="form_input">
              <html:text property="userName" size="40" />
        </td>
    </tr>
    </table>
      <html:submit property="Dispatch" value="Save"/>
    </html:form>
```

What is a Form Bean?

What makes a Java bean a form bean in Struts terminology? A form bean is a Java bean whose properties match with the properties of an HTML form. A valid Struts form bean for the above HTML form would be the following:

```
public Class MyUserBean extends org.apache.struts.action.ActionForm
{
        private String userName = null;

        public void setUserName(String userName)          {
                this.userName = userName;
        }
        public String getUserName()          {
                return userName;
        }

}
```

A form bean *does not* have anything to do with how the data is stored or where it comes from, it is based purely on the html form.

Note first that this bean extends from org.apache.struts.action.ActionForm.
The property userName on the HTML form corresponds to getUserName() and setUserName(...) methods on the form bean. This allows Struts to automatically populate the HTML from the form bean, and likewise transport user input in an HTML Form field to the form bean. You could say that this is a binding between HTML form and bean. The mechanism allows to maintain the values of the HTML Form fields (in the bean) across browser requests.

Since html form processing works over HTTP, a String protocol, all getters and setters work with String only, regardless of the native type it is in.

As we will see later, the same binding mechanism can be made to work for multi-row forms, that is forms which repeat the same field name several times. An example for a multi-row form would be a tax input form with a variable-length list of expenses.

You may have noted that the form beans' signature for get/setUserId uses String rather than Integer. Since HTML forms always return a String to the server, Struts expects getters and setters using String. A conversion to other types that may be required in the database (Such as Integer or BigDecimal for ID fields) must be done inside the getters and setters. This type requirement is specific to Struts input tags.
A special case are check box tags which use booleans in the signatures.
Some tags in other tag libraries which are used for display can handle other types, such as <fmt:dateFormat, which can have getters which return Date or Timestamp.
If your form bean is shared between different pages (one that displays data and one that updates data), you may have two getters and setters for the same field, one for display binding, and one for update able form binding. This would be an prototype excerpt of such a bean:

```
public Date getStartDDate() {
//..
}


public String getStartDate() {
//..
}


public void setStartDate(String arg) {
//..
}
```

See how there are 2 getters?

You may note that since the display-only JSP is not an input form, the bean does not need a setter method for this property.

“Realistic” Bean Prototype

The following is an example of a bean that works with tabular data, such as rows of columns, or, as implemented here, a List (rows) of Maps (columns) objects.

```
public class TasksBean extends ValidatorForm {
private Map _rowMap = null; // a fake row of data, name/value
(column / column date)
private List _rowsList = null; // a list of rows (could be 1 row)
public TasksBean()    {
populate();
}
public void populate()      {
_rowsList = new ArrayList();
_rowMap = new HashMap();
_rowMap.put(“id”, “1”);
_rowMap.put(“Task_Name”, “Clone an Army “);
}
private String getValue(String a) {
return (String) _rowMap.get(a);}
public void setValue(String n, String p) {
_rowMap.put(n, p); }

public String getTaskName() {
    return (String) getValue(“Task_Name”); }
public void setTask(String arg) {
    setValue(“Task_Name”, arg); }
// ...
```

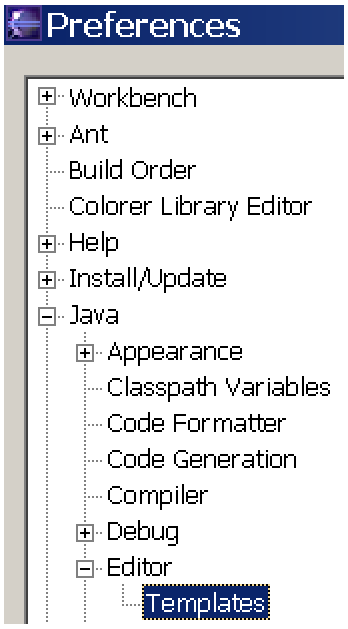
Writing IDE Macros

Writing getters and setters that bind to an HTML form can be a repetitive task with large applications. If your application has a hundred forms, you are likely to have 100 form beans. If each form has about 20 properties, that makes 2000 getters and 2000 setters.

Fortunately, modern IDEs allow you to create macros for code generation. Using macros speed up the task and can make you a lot more productive.

In Eclipse, a macro is defined in Preferences-Editor, and invoked via CTRL-SPACE.

Whatever editor you are using, you should master it.



When to Map a JavaBean

If the bean only displays data and does not have an input form, you do not map the bean in struts-config.xml. You rather create it in the action and put it in scope as shown above. Since a lot of pages in an application are usually display-only, it may well be that in the majority of cases you do not map the bean.

It is also important that when you map a form bean, you do not have to “create” a bean in action, like this:

MyBean mb = new MyBean();

Because the bean is created for you by Struts. If however you need to display read only fields, you should create the bean and put it in request scope.

There are two types of beans

- (RO) beans that only display data, and is used for read only
- (R/W) beans that are used to edit, insert, or update data on a form

Depending on how you use a bean, you have to do things a bit differently. When using a bean for updates (formBean), you must map it in the action.

Here is R/W example:

As step 1, you declare the bean in the <form-beans> section of struts-config.xml.

```
<form-bean name=”myForUpdateBean”
class=”com.basebeans.mypath.MyForUpdateBean”/>
```

As step 2, in the <action-mappings> section of struts-config.xml, you declare that Struts should automatically put the bean into scope of the JSP (and action).

```
<action path=”/contentAdminPg”
      type=”com.basebeans.basicPortal.cms.ContentAdminAct”
      name=”myForUpdateBean” scope=”session” validate=”false”>
      ...
</action>
```

Then you get a handle in to the bean like this:

```
public ActionForward execute(ActionMapping mapping, ActionForm form,
HttpServletRequest request, HttpServletResponse response) throws
Exception {

      MyForUpdateBean bean = (MyForUpdateBean) form;
      //bean.populate();

      ..


   } //exec
```

See how Struts passes it to you as an argument? You just cast it as your type and you can use it. It is a mistake to initialize a bean that is mapped, since struts creates it for you.

Of course, you do not have to create separate bean classes for display and for update. If you want to use the identical class for both purposes, but want to keep them instantiated and scoped separately, you would just use different keys for the session.

Majority of beans we use are used for RO. For example: displaying a user name, display some content. Those beans do not get mapped.

Here is a RO example.

```
public ActionForward execute(ActionMapping mapping, ActionForm form,
HttpServletRequest request, HttpServletResponse response) throws Exception {
        MyForDisplayBeanInPageScope b = new MyForDisplayBeanInPageScope );
        //b.populate(); //or other code to populate the bean

        request.setAttribute("myForDisplayBeanInPageScope", b);

        //return (mapping.findForward("Success"));
} //exec
```

The above puts the bean in request scope, thus making it accessible for JSP tags on any JSP page..The author use the decapitalized bean class name as key, such as "myForDisplayBeanInPageScope" for the class MyForDisplayBeanInPageScope, but any name will do.

When using same bean class for both, you may have more than one getter for a property, as we illustrated earlier: one for String getters and setters; The other getter is for the native type, such as long or Date.

If values for Struts R/W tags (such as text input fields) need to be localized, this is done in the getter method of the bean.

A R/O bean is localized in the JSP using JSTL, such as FMT tag, as shown earlier.

Logging

You should never use System.outs They cannot be turned off for production, and even one System.out can cause performance problems, since it is a I/O operation to a device (console), and with 100's of concurrent users, it gets executed a lot.

This is how you can use logging:

```
import org.apache.commons.logging.*;
public class SomeClass extends SomeThing {
    public final static Log log =
            LogFactory.getLog(SomeClass.class.getName());
public void SomeMethod() {
..
    log.debug("I am in here and value is" +
_someVariable.toString());
..
}
```

You can now turn on debugging by adding your package to log4j.properties:

```
log4j.category.com.bP=DEBUG
log4j.category.com.myOrg.myPackage=DEBUG
```

In production we can now centrally turn of all debugging. Any exceptions could be logged at the error level, log.error(e), and they would still be logged to a file and displayed to console.

Review

You should know how to write a Struts bean, and how to use it for R/W and RO.

Lab K: Simple Form Beans and Macro

1. In you com.myOrg.tasks, create a new class, TaskBean.
2. Ex:

```
import java.util.*;
import org.apache.struts.validator.*;
public class TasksBean extends ValidatorForm {
private Map _rowMap = null; // a fake row of data, name/value
(column / column date)
private List _rowsList = null; // a list of rows (could be 1 row)
public TasksBean()          {
populate();
}
public void populate()      {
_rowsList = new ArrayList();
_rowMap = new HashMap();
_rowMap.put("id", "1");
_rowMap.put("Task_Name", "Clone an Army ");
_rowMap.put("Task_Desc", "...");
_rowMap.put("Task_Status", "2");
_rowsList.add(_rowMap); //row 1
_rowMap = new HashMap();
_rowMap.put("id", "1");
_rowMap.put("Task_Name", "Train the Army ");
_rowMap.put("Task_Desc", "...")
_rowMap.put("Task_Status", "2");
_rowsList.add(_rowMap); //row 2
    }
private String getValue(String a) {
return (String) _rowMap.get(a);}
public void setValue(String n, String p) {
_rowMap.put(n, p); }
public String getTaskName() {
return (String) getValue("Task_Name"); }
public void setTask(String arg) {
```

```
setValue("Task_Name", arg); }
// ...
}
```

Write a getter and setter that matches the forms (jsps) properties.

Use a macro ("bgs" cntrl-space) for get/set. Keep in mind that later, when we work with db, if you have an integer coming from DB, you will not be able to cast it to a String (but will have to receive an Integer, and convert that to String).

Using Windows / Preferences / Java / Editor /Templates, review macros.

7. Optional: Write a String testIt method that returns a getTaskName() in Task Bean. Ex:

```
String testIt() {
{ return getTaskName(); }
```

Using Test Servlet in Core Bean package, instantiate the Bean, and output the testIt.

Ex:

```
TaskBean tb = new TaskBean();
Out.println(tb.testIt());
```

You can add a few more messages to output.

Chapter 7: Development Process Tips

Can You Build a Bridge (or a Web Application)?

Anyone can build a bridge. You can get 500 peasants to pile up rocks on a river during five years and you have a bridge. Probably boats will not be able to pass under the bridge, and the bridge will get washed out during a flood. Still, anyone can build a bridge.

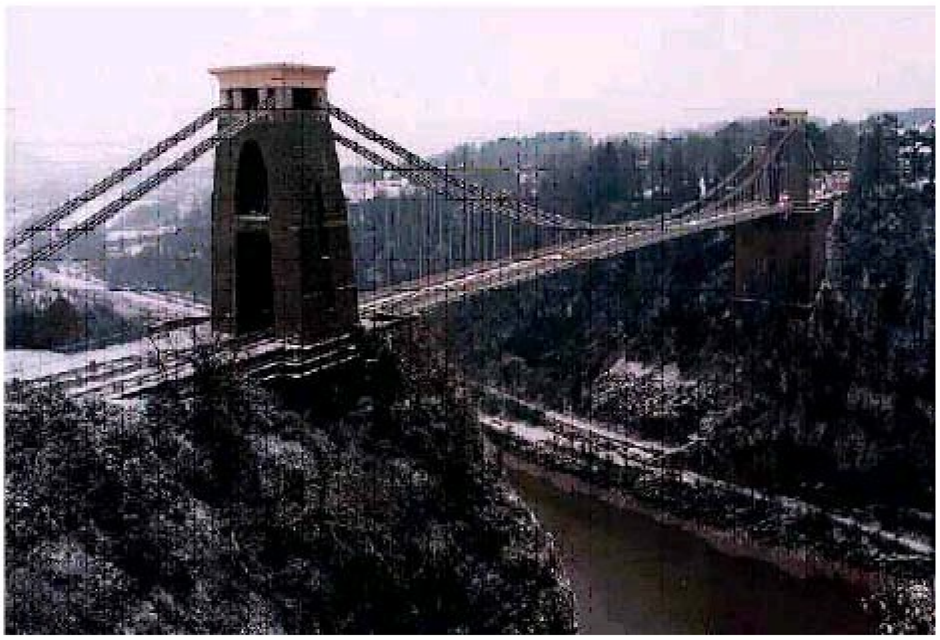
A bridge engineer, however, can tell you how thin the cable can be to save you costs. A car designer can make the engine more fuel-efficient and the silhouette less wind-resistant.

So can any developer build a web application even if he or she does not use an optimal design. But a software engineer can build it optimally: faster, more efficiently, more user-friendly, cheaper, and highly performing.

Not every developer may agree with this. Some will claim that they did not have the time to do it right, they didn't get the right requirements specification or they made it complex in order to do it right.

Think of professional and amateur basketball players One of the 2 will not miss a basket in a key situation. Are you doing development as a hobby or as a pro?

Read on and you won't need an excuse why your application failed—because it won't.



What is Project Planning?

Is Project Planning the same as Project Management? Definitely not. The former provides the plan; the latter provides the monitoring of the plan's execution. An analogy is financial budgeting (planning) versus monittroing by bookkeeping reporting on actual against budget (managing).

One of the best ways to manage a project is to track actual time to task, versus planned time to task. While you might think that your developers had two weeks of keyboard time, they may have actually spent time in meetings, fixing the LAN, or helping end users. They may have spent only two instead of 80 hours on the task.

There are many areas to master in project management, and one can obtain a certification in project management by an institution such as Project Managment Institute (PMI. Viewing a lecture by PMI does not make you a certified mamanger, but it takes years to obtain a certification, just like we take years to get Java or SQL certified). One of the authors has previously participated in the management of multi-million dollar real estate contruction projects. The experience helps.

In addition, managers of software development projects should have experience in software development. People who have never developed software are not likely to do well as project manager on a software development project. Nor are organizations likely to be successful in the long run if they put the wrong people in the role of project manager. For example, a manager of one department, being transferred to IT would not giving him skills to lead software engineers on a project; because he or she knows nothing about the software development process.

In the Capability Maturity Model (CMM), the SEI defines five capability stages for organizations that develop software. When they did an analysis on a large number of organizations, including professional software development companies, they found that the vast majority were only at the first and lowest stage of the CMM.

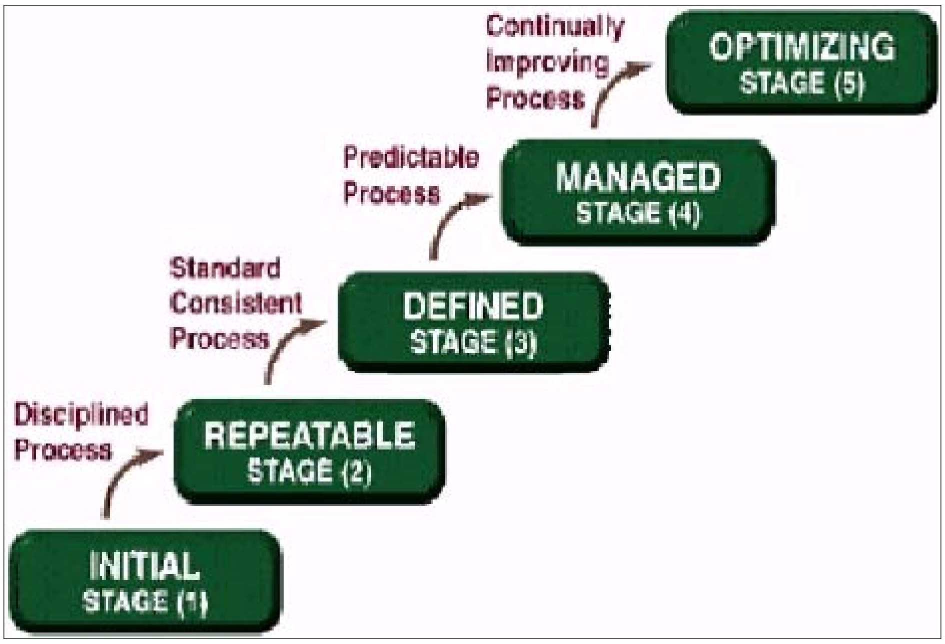


Figure 7-1: The Capability Maturity Model (CMM) © SEI

An organization passes from an initial stage via process improvements to the higher level stages. One of the improvements is to make the project methodology scientific rather than artistic. This means that others can repeat the same steps and get the same result.

There are many methodologies out there, and each consulting company claims to use at least one. A classic methodology is the Waterfall, already used on COBOL projects. Other methodologies are something controversial, such as Rapid Application Development (RAD).

The only methodology that works better than others is Iterative Development. You will read more about Iterative Development in the section after the next.

Lets define some concepts that will help you.

What are tasks?

A task has all of the following:

- a due date
- a single resource responsible; a senior if more than one resource is involved
- a deliverable
- is planned

If even one is missing, it is not a task, but an activity.

A task may also have the following:

- a Quality Assurance level, potentially with expected productivity
- objective monitors and/or reports, such as hours to task versus budget

For a successful project, beware of Activities. An activity has at least one of the following characteristics:

- no due date
- it is unknown who is responsible
- there is no deliverable
- it is a meeting about progress or about anything else
- it is not planned
- it is subjective.

Examples for activities are: the network was down so we fixed it; a client called with a problem so we worked on it; I stopped by to explain something to Juan; and then I had a staff meeting.

When the authors do project recovery, they mostly care about the due date, and who is responsible for the completion. It is up to the task owner to manage his time, days off, vacation, etc. As long as the task owner meets the date, he can take all the vacation he wants.

How do you react if a task slips? In project recovery, the only thing that has worked for the authors is to immediately reassign the task as soon as the first sign appears that a task will be late by the person it was originally assigned to.

Only someone with long development experience can know how long something will take to implement. Furthermore, the task owner must participate in deciding the due date. So if he says, “I should get that done by March 15”, the project manager says “OK, let’s write down March 19 on the plan”. Sincere buy-in is very import.

Iterative process steps

“Do it once and do it right” may be a good rule for many things. However, this is not the most efficient way to develop software.

The authors have seen that an iterative process works better. Iterative means that you do the whole entire project over several times. Some people think that iterative means modular, where you just add further modules. This is not iterative, it is not a known process.

You do each screen and each report several times! Yes, in an iterative process, you take the entire project, all the pages, reports, etc. through several iterations. The steps can vary, but this is a good example:

1. HTML Mock-up
2. JSP
3. Tiles and Navigation and Style development
4. Beans Prototype
5. Read-only version
6. Create-Read-Update-Delete (CRUD) version
7. Validation
8. Security
9. Embellishments such as drop downs

You develop an HTML prototype and PDF mock-ups for the entire application which you can demonstrate to the client for approval. Then you add simple tiles, navigation, and CSS style sheets. You then demonstrate the prototype again to obtain approval for the look and feel.

At this point, you have not yet written any beans. Now you write beans and get the entire application to display data (no updates). Then you add updates to the application; then inserts. Then you add validation; then security; then further embellishments such as drop downs.

People sometimes ask when should the help screens and user documentation be written: in Step one, mock up. If we do not have enough information to write users docs that is how the users needs to use this application, we sure do not have enough information how to implement it.

The key to optimal development is that you do the entire application each time in order to complete the iteration. You get a continual feedback from the client on the which ensures that you meet the requirements.

Coding to Contract

It means that you write code to implement the API or interfaces. In our methodology, we created HTML mockups and prototypes and PDF report mock-ups.

Since the properties of a form bean must map to the properties of the HTML form, you can derive the bean properties directly from the HTML mockup/prototype. Coding to contract then just means that you have to get the beans to work. This involves getting the form beans to create, insert, update and delete data (CRUD).

Prototype and mock-ups can be considered as the contract or public Application Program Interface *(API)*, since each follows.

Whom to hire?

Sometimes the authors get asked to be involved in the hiring process. Manager can have a hard time selecting a qualified candidate, how to tell? The key question to ask a candidate is: “How many years of professional experience do you have with developing applications that went into production/operation?” This will tell you who to hire.

Some developers have never delivered a web application to production. It is difficult to judge in advance whether it is something they did that prevented it from happening. However, we do know that some people are just able to deliver. This is who we look for. Every successful organization has a few of these individuals on their projects.

Interview questions about candidate personality do not help. A person even with the best personality will not add value to the organization if he or she lacks experience and cannot deliver. If you cannot find people to hire that have production delivery experience, then make sure that the people you do hire are cheap and will not be allowed to touch the source code. They might be of some use after a few years of sandbox code maintenance.

On projects, you need a lot of people whom you continue to need in operations, after the construction phase has ramped down: database administrator (DBA), Unix admin, Tech Support, Report writer, etc. Once the application has gone into production, if you have do deliver a new version of the application, you may also need developers. Else you mostly need operations people.

The operations people also do most of their work weekends and off-hours, so they work in shifts. There is no need to have two DBAs and two Linux administrators during the day when most of the work is done on the weekend. The senior of them do the major work during daytime, such as restore exercises, and the junior of them cover the nights and weekends for normal operations.

Exit Strategy

Exiting the development and handing off to production can be tricky. There tend to be a lot of little tweaks that people want to slide in as the project evolves. It sometimes takes weeks to exit the development stage, even after the development has been officially completed and the requirements specification been met.

Or... the staging and production environment was not ready. The management was betting against you, that you would be late. Once you are done, we will order the hardware. What?

To avoid this, ensure maximum detail for the requirements specification and involve the client. Get him or her to approve your iterations and provide you with feedback, and buy in, such as on the look-and-feel iterations. If you try to get these kinds of approvals late in the project you are bound to have difficulties.

Make sure that your staging and production areas are ready plenty of time in advance. Sometimes organizations plan for failure, and order the production hardware once the development is completed. That is not the time to issue a purchase order. Not only will you have unnecessary delays, but you will also massively frustrate your developers who have labored very hard to make their deadlines, and now have to wait until they can see their work in production. Yes, there are bugs, but we sometimes see the rate of new bug (vs solved ones) decline and still no production.

Exiting development to production can be tricky and can eat up a lot of budget.

Fixed Bid

Clients would do well to do outsource development to a design studio on a fixed bid. Once development is done, they may not need the developers around. In this way, they can outsource a lot of the risk. They can just have admin to operate the system or to write custom reports.

Project Politics

Some good developers are not well educated on office politics, and they should be. One definition of the word politics is incompetence. This helps you translate: Q: Why did you do it this way? A: Because of politics. Translation: Because of incompetence.

Some people are successful in organizations purely by playing the politics. They take credit for the work done by competent people and shift the blame for their work to the competent people.

This “design pattern” can be hacked. It is naïve not to play politics with the incompetent people. Solution: Do not let people take credit for your implemented designs.

Also, do not drop your work, to help the “politicians”. If they claimed that they would do something for someone, let them do it, maybe they can. If they can’t get it done, they may blame the competent, but this is a better outcome. Next time they will learn and ask the competent. Don’t be naïve, you will be taken advantage of, and the project promotions to leads for future projects will go to the wrong people.
Then junior developers end up working for the politician.

Do not be naïve, you hurt the rest of the software engineers.

Review

An experienced software development project manager using a proven methodology process can provide stability to a software project…. Or fail it, regardless of how hard developers work. In review labs, you now have gone from a mock up (html) to a Struts prototype. The prototype does not talk to the database.

We have created a Struts action that creates a form bean and we display in tiles/JSP, via navigation and we have a look and feel approved.

Preview

This part of the book has covered the development environment set-up, database model design, mock up and prototypes and process, as well as basic MVC with JSP, actions, mappings and beans.

In the next part of the book, you will build on this foundation to do advanced MVC, crud, drill down, and more advacanced way of doing beans and actions, as well as some OO.

Lab L: HTML text tag

1. Important: Browse HTML tag docs. (Docs are in the war files we looked at earlier. Or on the web at cheat sheet, or from Struts home page). You should for example know how to look up what an indexed property of “text” is.

2. On the editTask form (JSP), declare the Struts HTML Tag.

3.Type in something like this:

```
<html:form action=”/taskEdPg” >
<tr>
        <td class=”text_label” ><fmt:message
        key=”task.taskName”/>: </td>
              <td class=”form”><html:text property=”taskName”
    size=”25”/> </td>
..
<html:submit value=”Save”/>
</html:form>
```

Note: taskName property on a form (jsp) needs a getter called getTaskName in formBean. By convention, get/set “TaskName” must be upper case in bean which is converted to lowercase in properties “taskName”

4. Edit Struts-config.xml

5. Declare a form bean.

Ex:

```
<form-beans >
   <form-bean name=”taskBean”
   type=”com.yourOrg.TasksBean” />
```

7. Modify the taskEdit mapping to use the named (declared) form bean:

```
<action path =”/userEditPg”        type=”com.yourOrg.TaskEditAct”
  name=”taskBean”     scope=”session”  validate=”false”>
   <forward …
```

Remember that when declare a bean in a action mapping, Struts instantiates if for you.

8. Test to see if you are getting values from the bean.

9. Optional: Place a debug (“I am here”) in setTaskName() to see when it executes. (Test)

The difference between a good and a poor architect is that the poor architect succumbs to every temptation and the good one resists it.

■ Ludwig Wittgenstein

Part II

The Bricks and Mortar

Chapter 8: Doing Data Access Right

Introduction

As a result of the tutorials in the first part of the book, you should have reviewed a working prototype of your application at this point. The prototype contains a navigation system, the GUI (look and feel). It has a running database with sample data and you can connect to it via JDBC.

If you are a developer, but do not intend to complete the labs, the authors recommend that you return the book to the place you purchased it from.

Struts does not cover DAO, CRUD or displaying data grids (i.e. list views of data) but we will. Struts is meant to be data agnostic, but we will apply and demonstrate a practice.

Data Layer Access Approach – The M in MVC

Organizations spend —and waste—man-years of valuable development brains in writing unnecessarily complex data access. Good choices can be made, however, and the authors advocate them in this book. Choosing a solid data access implementation is crucial. One of the worst things you can do in a project is to let someone who has not brought any applications to production choose the data access implementation.

Layered MVC means that even form beans or actions should not contain any SQL or database-related code. Such code should be properly encapsulated.

A design pattern Data Access Object (DAO) comes to the rescue. One of the primary forces or reasons to use this pattern is to hide or encapsulate the data access implementation from the business logic that uses it.

There are various benefits to using the DAO pattern. One is that the data access methods can be standardized, irrespective of the data source (such as an SQL database, a flat file, XML, *MQ* or legacy systems). Developers can maintain data access more easily if it is standardized. This holds true also for situations where all data resides in a relational database.

At one point you may need to migrate your application to work with another relational database product than the application was designed for. This invariably means having to adapt the SQL statements used. If you have properly encapsulated your data access, and not have SELECT statements all over the place in your code, your migration task becomes much easier.

Hiding the data access implementation allows you to change the implementation later without having to touch the business logic. For example, if someone forces you to use EJB Entity Beans for data access, and you manage to encapsulate this access using the DAO pattern, you can later replace the Entity Beans with another data access implementation that proves better performance—without having to rewrite your business logic. If you choose JDO for your implementation, and something better than JDO comes along over time, you can switch it out easily—as long as you have properly wrapped it with a DAO.

As mentioned earlier, form beans should not contain database-related code, and should delegate all CRUD work to DAOs. Any bean should be agnostic as to how it obtains its data. Looking at the bean, one should not be able to tell if it received its data from a message queue or an SQL database. This is done by having a separate DAO class that the bean talks to. A bean only has public getters and setters, not much else.

It may be worth noting that Sun's more recent reference implementation of a web application ("Adventure Builder"), unlike the earlier Petstore, does not use EJBs but follows the DAO pattern.

Sample DAO Interface

You hide the implementation of the DAO from the bean by using an interface. Every time a bean talks to a DAO, it should do so via an interface. Any class that implements a DAO interface can be used to service your beans.

The official DAO pattern is foremost a design pattern, and thus does not specify the methods a DAO interface has to use. This means that you can define the interface name and methods yourself. A typical DAO interface class would look as follows:

```
public interface DAO  {
    public abstract void populate(Object obj) throws Exception;
    public abstract void populate(long l) throws Exception;
    public abstract void update(List list) throws Exception;
    public abstract List getResList() throws Exception;
         public Document getResXML() throws Exception;

    public void updateXML(Document xmlArg) throws Exception;

    public abstract void tBegin() throws Exception;
    public abstract boolean tCommit() throws Exception;
    public abstract void tRollback() throws Exception;
}
```

DAO Implementation Choices: Entity-relational or Object-relational?

In most cases, a DAO talks to a relational database. While you can hide the implementation using a DAO interface, you still need to decide how you actually implement the database access.
There are two major types of DAO implementations. The first type is a tabular entity-relational (E/R) implementation. These implementations are based on traditional entity-relationship diagrams, and return rows of columns, like a dataset. Sometimes, they can return a single row, but they return lists or sets of multiple rows. They are based on SQL, and can handle any SQL command, no matter how complex. Since they wrap around SQL queries, they can implement joins and can also be tuned for performance. They can do correlated and nested stored procedures and they can tap into the full power of the SQL database(s) that you happen to use.
Examples for entity-relational DAO implementations are:

- Cached Rowset (such as from Oracle, pgSQL)
- iBatis
- Scaffolding
- Jakarta SQL Commons

The other type is an object-relational (O/R) implementation. Such implementations follow object-relational models such as from the *Object Management Group* (OMG). They do not use ANSI SQL but languages such as *Hybrid Query Language* (HQL), *Embedded Query Language* (EQL) or *Object Query Language* (OQL), which usually have limitations with regard to multi-row handling and lack versatility, especially compared to long-established SQL. They tend to not perform as well in operation since they cannot fully leverage the propitiatory power of the SQL engine.

Examples for object-relational DAO implementations are:

- Hibernate
- EJB
- Java Data Objects (JDO)

The authors advocate to use DAO pattern implementations which follow the SQL and tabular entity-relationship paradigms. When developing an application on top of an SQL database, SQL needs to be used somewhere, often in conjunction with stored procedures for optimization. In real life, when O/R layers are chosen, many times the O/R functions wind up being bypassed.

We will implement our DAO using iBatis DB layer. Read more about IBatis at www.ibatis.com

Mapping a DAO to the Database With SQL

All database access should be done from the DAO. There are four basic types of database access: Create, Read, Update and Delete; commonly referred to as CRUD. One DAO implementation object chould handle all of these. You will not want to complicate your life with a data layer that uses different objects for inserts than for read-only.

For this book, the authors have chosen the iBatis database layer implementation. It is important, however, to note that the DAO layer allows you to switch to another implementation without having to change a single line of code in your beans.

The iBatis implementation uses an SQL mapping file in XML format to provide the information to the DAO which table and columns to access. The following is an example of such a mapping file:

```
<sql-map name='UserSQL'>
<cache-model name ='user_cache' reference-type='WEAK'>
    <flush-interval seconds ='30'/>
    <flush-on-execute statement = 'UserUpdate' />
</cache-model>
<mapped-statement name='UserSelectOne' result
          cache-model='user_cache' inline-parameters='true'>
    SELECT ID, NAME_LAST, NAME_FIRST FROM usr
    WHERE ID = #value#
</mapped-statement>
<mapped-statement name='UserUpdate' inline-parameters='true'>
    UPDATE usr SET
    NAME_LAST= #NameLast#,
    NAME_FIRST = #NameFirst #
    WHERE ID = #Id#
</mapped-statement>
</sql-map>
```

You see that this XML file is where the SQL statements go. When the mapped statement UserSelectOne is called, its query is executed and the result returned in the format specified in the result map: a list of java.util.HashMaps, one for each row, with property names id, name_last, name_first. Also note that DAO provides automatic distributed data caching and flushing at the Model layer.
SQL statements are thus outside your Java code; this allows you to work on your application when optimizing your SQL. A SQL expert can tune the SQL without having to touch Java.

The column has to map the db field name in select, but the property name could match the bean getter, for example “FIRST_NAME” column maps to “FirstName” property. is a

Connecting a DAO With the Data Source Pool

With iBatis, SQL mapping files are registered with the application through a central SQL-Map-Config configuration file which also specifies the data source. Typically, the data source is the connection pool specified for the container. The following is an example of SQL-Map-Config content:

```
<sql-map-config>
<datasource name = "ids"    factory-
class="com.ibatis.db.sqlmap.datasource.JndiDataSourceFactory"
default = "true" >
    <property name="DBInitialContext" value= "java:comp/env" />
    <property name="DBLookup" value= "bppool" />
</datasource>
<sql-map resource =
     "com/baseBeans/scaffoldingXpres/coreBeans/UserSQL.xml" />
...[more sql mapping files here]...
<sql-map-config>
```

The above example assumes that the container is aware of a J2EE connection pool named bppool and the SQL mapping XML file was stored in the application source code tree at the shown location under the name UserSQL.xml.

Your DAO implementation could then read this sql-map-config.xml into a sqlMap object in a static initializer as follows:

```
static
{
    log = LogFactory.getLog(DAOIBaseHelper.class.getName());
    _sqlMap = null;
    try
    {
          java.io.Reader reader = Resources.getResourceAsReader(
                "properties/sql-map-config.xml");
          _sqlMap = XmlSqlMapBuilder.buildSqlMap(reader);
          _sqlMap.setCurrentDataSourceName("ids");
    }
    catch(Exception e)
    {
          log.fatal("XX no iBatis config found " + e.toString());
          e.printStackTrace();
    }
}
```

This assumes that the sql-map-config.xml is accessible in the /properties directory (like resources/properties for internationalization resources). By using static intalizer, we get the handle to the JNDI lookup only once for the entire web application.

Using the DAO

Once registered with a SQL map, you can access and use the mapped statements in your DAO implementation. You would typically have a UserDAO class that encapsulates persistent data about one or several users. Following the above examples, to execute a query by a primary key to the user table, in UserDAO, you could use the iBatis methods of the SQLMap object as follows:

```
List _oList; //to holding the list of rows retrieved
Map _current; //to hold a Hashmap of a row of data
_sqlMap.startTransaction();
_oList = _sqlMap.executeQueryForList("UserSelectOne",
myprimarykey);
_sqlMap.commitTransaction();
if(_olist.size() > 0)
_current = (Map) _olist.get(0);
```

iBatis would identify the mapped statement UserSelectOne in the registered SQL maps, execute the query based on the #value# of myprimarykey passed in, and return the query result according to the specification of the ResultMap (a list of HashMaps).

Next we talk about techniques in Java to make you more productive.

What Makes a Good Programmer?

Is a good programmer someone who can crank out a lot of good code? Does it take more than fast typing to be a good programmer?

Like any good employee, a good programmer can do a lot of high quality work quickly. The secret of a good programmer's productivity is that he or she practices re usability; via object-oriented programming (OO).

Object-disoriented Programming

Many people understand object orientation only academically. OO programming is especially difficult for people who are trained to think in terms of structural analysis and design, and top-down or bottom-up function decomposition. These approaches focus on differences rather than common things. Function-oriented people see that apples are different from oranges and have to be treated differently. OO people see that both apples and oranges are fruit, grow on trees, can be eaten, have seeds, and so on. They will try to share or reuse the same treatment as much as possible.

It takes months of discipline to switch your brain from the old procedural approach to OO. Usually, it also takes serious mentoring. If you are under pressure to deliver, it is by no means easy to implement productive OO. However, once you have been able to switch over your synapses and have completed an OO project, you will notice a tremendous jump in productivity. Chances are that you will be to an order of magnitude (10x) more productive.

OO does not mean a change of programming language but a change in the way of using that language. Object-disoriented programmers use the verb(noun) paradigm. For example they write code like the following: Print(pdf); Print(text); Print(doc); Print(gif). The code that does the work on an object is not a part of the object. It follows that the print code has to handle error conditions for various object types. The code can be almost anywhere; it is hard to decide where something should be done, and it is hard to maximize reuse.

Object-oriented programmers use the noun.verb() paradigm instead. Their code will look as follows: PDF.print(); DOC.print();. The behavior of an object goes with the object. It is not necessary to know how printing happens; the print work goes on inside that object. We also know where to put the code and where to find it.

Just because you code in Java does not mean that you do OO. Nothing in Java will tell you that you are not doing OO. The code will compile and execute with no warning.

A bad project is one where they say "We did not have time to do it in OO". The truth is that if they had done it in OO, they would have had the time to do it in a relaxed manner.

The Benefits of Object-oriented Programming

Object-oriented programming enables code reuse. Re usability means less code, and less code means less bugs and faster programs.

If you want or need to be more productive you can achieve it by using Java with good OO practices. An object-oriented developer is usually more productive than ten object-disoriented developers. An experienced programmer can write several MVC modules per day.

One pragmatic reason that code needs to be written quickly is so that the project can be completed before the client changes the requirements. Requirements always change over time, and if the project is not completed quickly, it will never end. Coding fast is your best way of not getting stuck in a never ending project.

A single person can outcode entire programming teams. This is excellent news at times where bigger software companies try to play the card of "bigger is better". You do not need to be afraid of their competition. These development companies can often be outcoded just with good software design. Smart clients realize that and will hire you; some, of course, having previously been burnt by a big development outfit.

Since a good OO programmer is faster than ten non-OO programmers, a good way to get a pay raise is to learn object-oriented programming. While you may not get a tenfold salary increase (you will be ten times more productive) , chances are that you are able to increase your job security.

On one project recovery, one of the authors was called into a project that was two months behind schedule. This was for a major consulting company in Dallas, Texas. Within one month on site, we were one month ahead of schedule. How did we do four months of work in one month with the same staff and no overtime? The answer is: reuse. We have to be able to look at things and not see apples and oranges, but see one base apple-orange fruit, from which we can make a pear easily. A big part of advanced Struts is to maximize reuse.

Making DAOs Reusable

You would typically have at least one DAO class per database entity: one UserDAO class for persisting information about users, one ContentDAO to access CMS content and so on.

You normally start by creating implementing DAOs to serve your beans. It is likely, however, that you will be able to reuse this DAO mapping for other screens and business logic as well, just by adding another mapped statement.

Any decent DAO comes with inherent capability to treat multi-row. Therefore, you could, for example, have a page with a table that lists the items you can edit, and a detail page with input fields to edit a single row.

Using DAO Helper Objects

Of course, you think of OO re usability as well, and will bring code that all DAOs use into a base class, such as BaseDAO or DAOIBaseHelper. This base class would have the standardized implementation for operations like retrieving data and updating data, to be reused by all the classes that extend from it.

The following is an excerpt of such a DAO helper base class:

```
protected List _olist = new ResList(); // orginal native result
list, for the copy option, fix retrieve

    public List getResList() {  // return the data from db

        return _olist;

    }


protected void doRetrieve(String sa, long i) throws Exception {

try {

_olist = _sqlMap.executeQueryForList(sa, new Long(i));

if (_olist.size() > 0) _current = (Map) _olist.get(0);

}// try

catch (Exception e) {

log.error("X sql DAO doRetrieve" + sa + i + e.toString());

throw new Exception(e);

} //catch

return;

} // retrieve


protected void doUpdate(List alist, String sUp, String sIn) throws
Exception {

try {

_s = alist.size();

for (int i = 0; i < _s; i++) {

_current = (HashMap) alist.get(i);

String newFlag = (String)

_current.get(BConst.INSETNEWFLAG);

if ( BConst.YES.equals(newFlag) )

    _sqlMap.executeUpdate(sIn, _current);

else

    _sqlMap.executeUpdate(sUp, _current);
} //for
} catch (Exception e) { log.error("X sql DAOdoUpdate " +
e.toString()); throw new Exception(e); } //catch
}//update
```

This DAO can support multi-row updates with just a few lines of code. A DAO that extends from this base class would have public wrapper methods that call the base class methods. Finally, your UserDAO would look somewhat similar to the following:

```
public class UserDAO extends DAOIBaseHelper implements DAO
{
    public void populate(long i) throws Exception
    {
          doRetrieve("UserSelectOne", i);
    }
    public void update(List list) throws Exception
    {
          doUpdate(list, "UserUpdate", "UserInsert");
    }
}
```

You could have one DAO per entity or formbean; the result map in its SQL map would have a long list of property mappings (all columns of the table, if necessary), and several mapped statements that share the same result map. Most of the times a DAO does a lot of joins.

For example, one mapped statement would only query for the columns that are required for the first screen, another mapped statement would query for all columns if needed as such elsewhere. You can use the same result map without necessarily always filling all its properties with values.

Once you have that DAO working, you will query it from an action rather than in the bean constructor. Since querying the database could fail (db could be down etc.), it is better to query from a place where such exceptions can be caught, so querying from an action is better than querying from the bean constructor. You also have more flexibility if your query parameters change between requests.

You will reuse the UserDAO from the previous chapter. This time, you need to do a query that returns a list of all users, rather than just one, so you add a method that uses a different mapped statement that you will create.

It is very easy to create a unit test for DAOs. DAOs can have a testIt() method that returns the result of the test. The implementation of a testIt() method on a DAO can be as simple as the following:

The testIt() method on a DAO:

```
public String testIt() throws Exception
{
    populate();
    return getResList().toString();
}
```

The testIt() method on a bean:

How do you call this test? You would have a servlet that you run in the browser and would return the test result String as a response. The servlet could look as follows:

```
public class MyTestServlet extends HttpServlet  {
    protected void service(HttpServletRequest req,
                            HttpServletResponse res)      {
          PrintWriter out=null;
          try
{
          out = res.getWriter();
          res.setContentType("text/html");
          MyDao d = new MyDao();
                out.println(d.testIt());
}  catch (Exception e)      {
    log.debug("Exception in TestServlet:"+e);
          }
    }
}
```

Review

We should access data and persistence via a DAO interface that we implement. A productive way to implement anything in Java is to apply OO.

LAB M: DAO

PART 1: Create and register a SQL mapping file

1. Important: Review the iBatis DAO documentation. (In docs or from iBatis web site)
2. Clone a UserSQL.xml , .. as TaskSQL.xml, copying it to com.yourOrg coreBeans package.
3. Inspect **TaskSQL.xml**. See how it maps different SQL queries to the user table right now.
4. Make sure that there are no references to “user” in the TaskSQL.xml, (search/replace with “task)
5. Edit the User specific SQL to Task specific SQL. Test SQL in pgAdmin first. You need to write these in you XML DAO.:
    1. TaskSelectOne
    2. TaskSelectAll
    3. A DAO column name mapping of the results
    4. TaskUpdate
    5. TaskInsert
6. Modify the sql-map-config in source properties to add the new DAO mapping.
7. To the list of <sql-map resource...> tags, add the following:
   <sql-map resource=”com/myorg/web/TaskSQL.xml/>
8. Save sql-map-config.xml.

Result: Created and registered SQL mapping file with working SQL CRUD.

PART 2: Create a DAO that uses the SQL map.

1. In src/com/myorg/web, create a new class **TaskDAO.java**. Make it extend from **fx base.util DAOIBaseHelper**, and implement **fx base DAO** interface. (the IDE should import it if you right click)

2. Implement methods as follows:

```
public void populateAll() throws Exception{
   doRetrieve("TaskSelectAll");
}
public void populate(long i) throws Exception {
   doRetrieve("TaskSelectOne", i); //by primary key
}
public void update(List list) throws Exception {
   doUpdate(list, "TaskUpdate", "TaskInsert");
}

//String should match DAO names.

public String testIt() throws Exception
{
   populateAll();
   return _olist.toString(); //public property of parent class, query results
}
```

Peak at UserDAO.java to see an example. Note that all the "Task" variations must exist as a mapped statement in the sqlmap.

3. Save TaskDAO.java.

4. In Eclipse, open the definition **DAOIBaseHelper.java,** by right clicking on the word (when inside the TaskDAO). See how it implements the methods we use from TaskDAO. Note that instead of extending from DAOIBaseHelper, we could implement the database access in TaskDAO. This is example of code reuse, since method are generic and could be used by any DAO. Ex: doRetrieve(), doUpdate().

5. Test TaskDAO.java.
In **com/myorg/test/TestServlet.java**, edit the code so it looks like this:

```
d = new com.myorg.task.TaskDAO();
out.prtinln(d.testIt());
```

Note that the servlet has been registered in Web.XML.

6. Save TestServlet.java. Rebuild the project.
Run http://localhost/test You should see a list of results in the browser.

Chapter 9: Reusable Beans

Unit Testing

One way to speed up development is to leverage OO programming features to improve reuse. The second way is to use Unit Testing and Quality Assurance (Q/A) procedures. Development is cheaper and takes less time and resources if problems can be fixed earlier rather than later in the process. Therefore, unit testing is important for projects with limited resources. Managers that are new to development are sometimes tempted to remove unit testing from the time line in order to make the project look like it will require less time. This can increase the time and cost of a project.

New developers often spend a lot of time debugging and looking through the beans, actions, JSP and SQL to find where the problems are. Their modules tend to be buggy. Experienced developers produce higher quality code, and when they deliver a module, it tends to work.

Their trick is that they unit test. Anytime you are overconfident and skip testing, it is very likely that you will have to go back and do it later. A bean should be unit tested before you connect it to JSP and Struts actions. If the bean does not work outside Struts, it will not somehow magically work inside Struts.

So we will should write a String testIt() method in the bean, and test CRUD.

Once mapped to a URL in web.xml, this servlet will give you the rapid answer to the question if your bean is working, does queries as expected, returns the proper data, etc. A bean should be unit tested before you connect it to JSP and Struts actions. If the bean does not work outside Struts, it will not somehow magically work inside Struts.

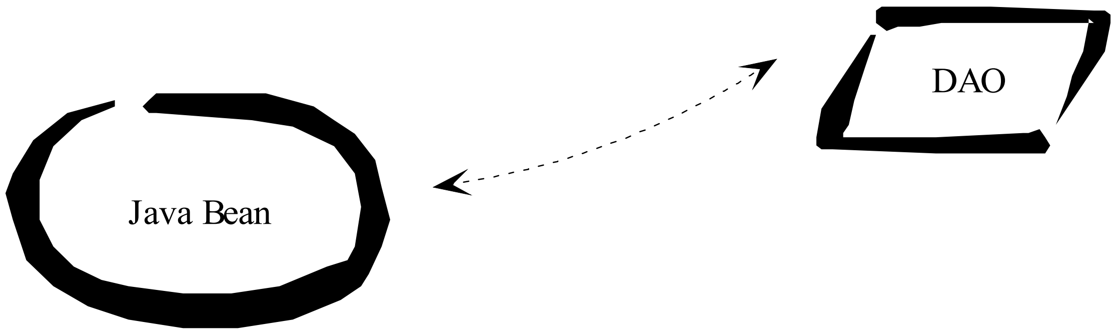
This approach is a short-circuit to Junit. Junit in most cases would require you to create and maintain separate classes for testing that extend from junit-specific classes. If you still want to use Junit as testing framework, the above would simplify its use: All test classes could call the testIt() method of the class to be tested and assert that the returned values are correct. Struts-specific test cases that call the execute() method also exist.

A unit test with testIt() is also a building block for regression tests of the entire application. For every new bean you write, you add it to the test servlet, and just comment out the test on the previous bean. At the end, just comment the all previous tests back in, and you have a complete test program already there to be executed.

Populating the Bean

Your bean sould not access the data, but it can use a helper, a DAO, to get the data for it. Ex:

```
public class MyUserListBean extends ActionForm
{
protected UserDao _dao = null;

protected ResList _rlist = null; // results list, if one row, then
one Map (current

protected Map _current = null; // current row


    public MyUserListBean()        { // cons
          _dao = new UserDAO();
    }

    public void populateAll() throws Exception //delegate to dao
    {
          _dao.populateAll();

          _rlist = _dao.getResList();

    }//populateAll


private String getValue(String a) {

    return (String) _current.get(a);}

public void setValue(String n, String p) {

    _current.put(n, p); }


public String getTaskName() {

    return (String) getValue("Task_Name"); }

public void setTask(String arg) {

    setValue("Task_Name", arg); }


public String testIt() {

   populateAll();

   return getTaskName();

} // testIt
}
```



Remember in Struts, you should not access the data source data directly, but it should utilize a DAO for that. The bean does not query the database until its populate() method is called. Since the goal is to call the populate method from an action, and the action has ready access to the bean, we add a helper method populateAll() to the bean that delegates to the DAO.

In the execute(...) method of com.myorg.web.MyUserListAct, you would do the following:

```
execute(…

    MyUserListBean b = new MyUserListBean();
    b.populateAll();

    request.setAttribute("myUserListBean", b);

return mapping.findForward("Success");
```

Or

```
execute(…


    MyUserListBean b = (MyUserListBean) formBean;
    b.populateAll();


return mapping.findForward("Success");
```

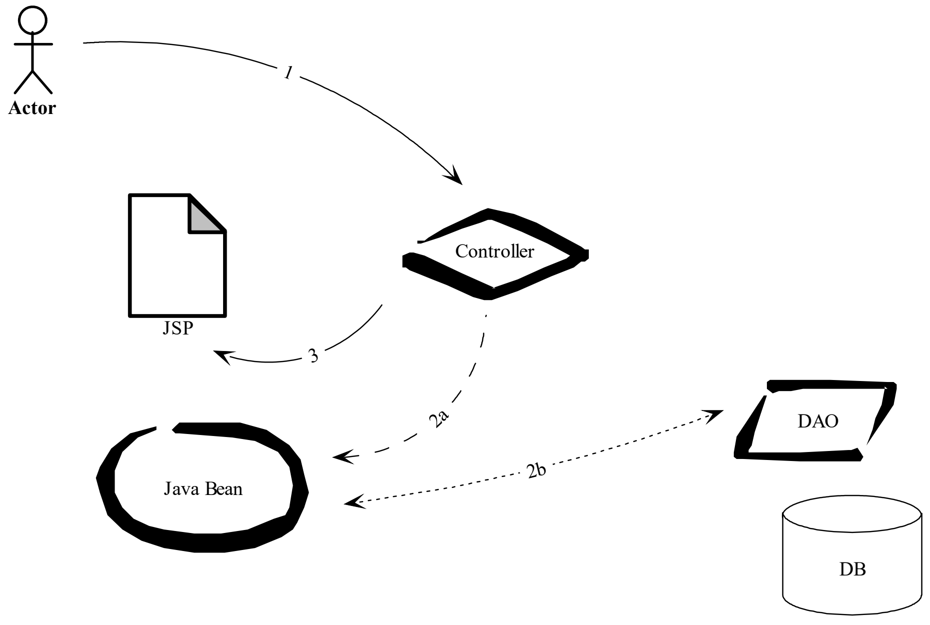
JSP and Bean

You should never use JSP use bean or Bean define in Struts, but should prepopulate the bean in the action as we have done. Here is a snippet of JSP code:

```
<%@ taglib uri="jstl/c" prefix="c" %>
...
<table>
<c:forEach id="row" name="myUserListBeanName"
property="displayList">
<tr>
    <td><c:out value="${row.id}"/></td>
    <td><c:out value="${row.name_last}"/></td>
    <td><c:out value="${row.name_first_init}"/></td>
</tr>
</c:forEach>
</table>
```

This uses the bean in scope from the action(controller).

When you call /do/myTaskListAct in the browser, the action will be executed, the bean will be instantiated and added to the session if not already there. In this example, the DAO will be populated with data on bean instantiation. Next, the action will forward to the JSP; now the JSP has the populated bean available and can access it using the c tags. The processed JSP will be returned as a response to the browser.



MVC For Form Beans With DAO

This is probably a good moment to present the interactions between action, bean, DAO and JSP in the context of the MVC (Model 2) implementation.

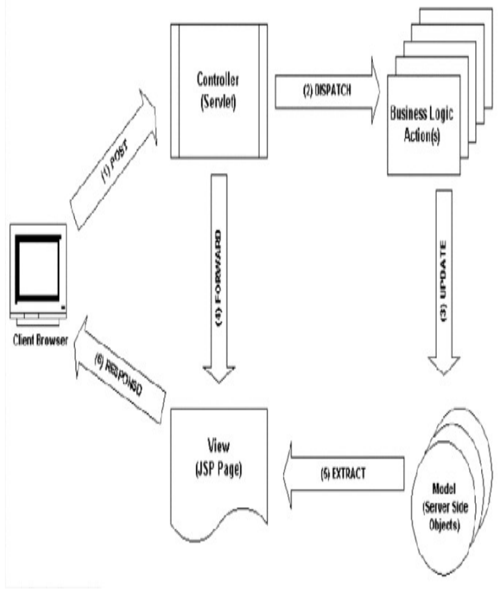


Figure 9-1: MVC2 Architecture

Figure 9-1 presents us with the MVC2 flow again. Applied to our example of the list of users to be displayed on a JSP page, the following happens:

(1) The Client Browser posts a request to the Controller servlet with the URL /do/myUserListAct, such as by clicking on a link, typing http://localhost/do/myUserList in the browser, or by calling the action through a tile layout which hides the request to this action URL.
(2) The Controller servlet dispatches to the Action class that is mapped to /myUserListAct in struts-config.xml, in our case com.myorg.web.MyUserListAct.
(3) The code in the action checks whether the bean com.myorg.web.MyUserListBean exists in the session, under the key myUserListBean. If not, it will create an instance and put it into the session. It will also ask the bean to query the DAO and populate its List of HashMaps.
(4) The Controller then forwards to the JSP page at /WEB-INF/portlets/memberSec/userList.jsp
(5) The JSP processor finds the struts:iterate tag. It accesses the bean myUserListBean in the session and calls its method getDisplayList(). This bean method returns the list of HashMaps from the pre-populated DAO instance on the bean. While iterating over the list, the JSTL c:out tags obtain the maps' values for each row.
(6) The Controller returns the JSP processing result (an HTML page with a table of users with id, last name and first initial) as a response to the browser.

Editing Data

In the previous example, the JSP accessed data from the DAO for HTML display. This illustrated the interaction between all the key elements. The only property the bean had was the DAO, with a getter named getDisplayList() that returned the whole content of the DAO in a List.

As you know, a form bean to edit data has one pair of getters/setters for each editable field on the JSP. Then we have to take care of transporting the values of those fields to the proper row of the DAO since the DAO may contain multiple rows of data. There may also be cases where the HTML form is multi-row editable, displaying input fields for several rows of data at the same time, much like an Excel spreadsheet does. This more complex situation is discussed later on.

Reusable FormBean

So far, you have used inheritance on the DAOs to reuse common methods and properties assembled in the DAOIBaseHelper class. The example beans shown have been extended from org.apache.struts.action.ActionForm. In the next step, you will further increase reuse by extending all your beans from a base class that you will write yourself. We will use Struts in an object oriented way.

To maintain the basic function of the Struts ActionForm class, your base class will extend from it. Struts already provides some classes that extend from its base class ActionForm, such as ValidatorForm. The ValidatorForm has methods that automate the validation of input fields, both client- and server-side.

You may call your base class BaseBean. BaseBean itself will extend from ValidatorForm so that validation functionality is available in all your beans.

Common Bean Methods

What should go into BaseBean? First of all, you will put there a holder for the instance of the DAO that you will use for that bean. Since all your DAOs implement the DAO interface, for this your BaseBean would have a property as follows:

protected DAO _dao = null;

But this is only the beginning. The following is a list of properties and methods on the BaseBean that have been found useful over time:

```
protected Map _current = null;
protected boolean _autocommit = true;
public void populate(Object o) throws Exception
public void populate(long i) throws Exception
public void save() throws Exception
public Object getValue(String a)
public void setValue(String n, Object p)
public boolean getAutoCommit()
pbulic BaseBean getIndex(int index) trhows Exception
public List getDisplayList()
public Map returnMap() thows Exception
public org.w3c.dom.Document getXDoc()
public void insertNew() throws Exception
public Locale getLocale()
public BaseBean gotoRow(int index)
public void beforeFirst()
public Iterator iterator()
private class IteratorAdapter implements Iterator

public abstract String testIt() throws Exception
```

You may have noticed that the testIt() method has been made abstract which politely forces you to implement the method on all your individual beans.

Iterator Adapter

Sometimes you will need the bean to implement the Iterator interface. The previously shown JSP that uses the iterate tag takes the whole list of data as one, and iterates over it. We used the capacity of the iterate tag to iterate over a List. The iterator interface provides this facility with the next() method. Inside the implementation of the next() method, you can do callbacks, totals, etc. So you can implement the Iterator interface on the DAO, or on the bean.

You can implement the iterator adapter like by just saying this:

```
public abstract class BaseBean extends ValidatorForm
   implements Iterator {
```

Review

We can develop a bean using object orientation. It could also have a helper DAO class; and we know importance of unit testing.

LAB N: Advanced Beans and Unit Test

We have a prototype Task bean that we need to modify to extend BaseBean, and to make it use a tested TaskDAO helper for CRUD.
We already did a simple bean, this is a more advanced bean, w/code reuse.

Examine the UserBean.

```
public String getRole() {
return (String) getValue("role_code");    }
public void setRole(String arg) {
     setValue("role_code", arg);
```

1. We will now modify TaskBean.java in src/com/myorg/web to read live detail data from the database (for eventual update). Open **TaskBean.java**.
2. In TaskBean.java, **remove** the instance variables _rowMap and _rowsList and get and set value (they are not in base) code.
3. Change the Task bean to "**extends** BaseBean".
This is a more sophisticated FormBean, and it has in base a current Map and rList. It also has a _dao.
4. Add an instance of the dao as property of TaskBean.java as follows:
protected TaskDAO taskDao = null;

5. Add a constructor to TaskBean.java as follows:

```
public TaskBean()
{
   taskDao = new TaskDAO();
   _dao = taskDao; // register w/ base

     }
```

6. Review the **getters** as follows:

```
public String getTaskName() { // what you call it in jsp; lower case
     return (String) getValue("Task_Name"); // what you called it in DAO
                          }
     public void setX(String arg) {
     setValue("Y", arg);  }
```

You should not have a setter for PK.

If you have an integer value in DB, you can use getValueIS and setValueIS to help with cast/conversion.

8. Create 2 populate methods:

```
public void populateAll() throws Exception {

taskDAO.populateAll();
setupList();          }
public void populate(long l) throws Exception {

taskDAO.populate(l);
setupList();          }
```

Create a module unit test:

Ex:

```
public String testIt() throws Exception {

StringBuffer sb = new StringBuffer();
populateAll();
sb.append("TaskName is " +getTaskName());

etc…

return sb.toString();

} // testIt
```

Modify Test Servlet to do

```
b = new TaskBean();
out.println(b.testIt());
```

Start container and navigate to localhost/test

Change the unit test (testIt) to modify and save:

```
populateAll();
setTaskName("Clone the smart army);
save();
return null;
} // testIt
```

Execute localhost/test

Bring up the db admin tool (pgAdmin)
Browse the table tasks. Did the data change?

Note: If a bean does not CRUD outside of Struts… placing it in Struts will not make it work.

Chapter 10: Reusable Actions and Events

Messages From Page to Action

Imagine that a user sees a list of lots of rows on the page. When he clicks on a single row, he would like to see details of that row. How do you communicate from the JSP to the action which row to display and query?

You can do this by adding a URL parameter to the action URLs on the JSP. The HTML on the list page would have to end up looking similar to the following in the browser:

```
<table>
<tr>
        <td>1</td>
        <td>Cekvenich</td>
        <td>V.</td>
        <td><a href=”/do/myUserDetailAct?id=1”>Edit</a></td>
</tr>
<tr>
        <td>2</td>
        <td>Gehner</td>
        <td>W.</td>
        <td><a href=”/do/myUserDetailAct?id=2”>Edit</a></td>
</tr>
</table>
```

This would allow the user to click on the Edit link and instruct the action which user data to edit. The id parameter would correspond to the primary key by which a user can be retrieved from the database.

What would a JSP that will dynamically generate this list have to look like?

```
<c:forEach var="row" items="${beanNameInJSPscope}">
<tr><td>
    <c:url value="myUserDetailAct" var ="url">
              <c:param name="ID" value="${row.id}"/>c:param
name="Dispatch" value="Edit"/>
    </c:url>
      or
     <a href ='myUserDetailAct?ID=<c:out value="${row.id}"/><' >
           c:out value="${url}"/>
      </a>
</td>
```

You see how the primary key for each row is dynamically added to the Edit link using the c (Core) JSTL tag. This requires a bean that has a getRow method

You create a new action myUserDetailAct that can retrieve and receive the message, by making use of the request parameter id. This action should forward to a detail page. The execute(..) method of this action would look as follows:

```
public ActionForward execute(…){
MyUserDetailBean b = new MyUserDetailBean();
request.setParameter("myUserDetailBean", b); //put in scope

String id = request.getParameter("ID"); //retrieve parameter
    b.populate(id);
    return mapping.findForward("Success");
}
```

When this action finds the ID parameter in the request, it uses it to populate the bean with a populate(parm) method that passes the primary key for querying through the bean to the DAO.

There is another way to send a message, using a display tag. Ex:

```
<display:table width="95%"  name="fb" property="displayList">
  <display:column property="title"            width="50%"
    title="Title"            styleClass="text"
                          href="/do/myUserDetailAct " paramId="ID"
    paramProperty="id" />
```

They both end up looking same in the browser, and now you can receive a message in the action.

DISPLAY TAG

The Display tag library is a most popular and advanced tag library library for data grids. It allows you todo multi-page, sortable data grids. You should review the display tag documentation and examples available on the Web.

Debugging JSP Request and Session

Now that you are starting to pass around request parameters, you may need to debug. The Sevlet API allows you to enumerate over all request parameters and session attributes. This is debugging code that you could use in the execute method of an action to output session attributes to the console:

```
Enumeration enum = request.getAttributeNames();
    while (enum.hasMoreElements())      {
           String paramName = enum.nextElement();
           log.info("session attr: "+ paramName + "=" +
                  request.getSession().getAttribute(paramName));
}
```

This can help you to find out what is in scope for a JSP. Commonly people are not sure what is in the session or request scope of their JSP, or they want to snoop other browser information. Any Servlet book has more information on snooping.

First Cut Action Dispatching

The action receives a query parameter ID, and uses it to pre-populate the bean. (Since the action does not have access to the DAO, the method populate(arg) has been added on the bean.)

The action myUserDetailAct will forward to display an individual row when called from a JSP with an edit link like this:

<a href="/do/myUserDetailAct?ID=1">Edit</a>

Now you have to start thinking about more actions that could be done on this data, such as save, delete, insert new or just redisplay. Should you have a separate action class for each operation?

You may, but this would need a lot of code, and quickly create a jungle of action classes. Better would be if you had a way to group these actions together as they relate to one module such as user administration. This would also give you the opportunity to merge the actions for List and Detail.

The way we can group actions is by doing if-then dispatching with action parameters. We should have one action to deal with displaying and editing. We can use a Dispatch parameter to do if-then dispatching. We define that a parameter named Dispatch will be used to decide what to do. This is what our first version of the execute(...) method of userAdminAct could look like:

```
String parm = request.getParameter("Dispatch");
try
{
    if (parm == null)       {
          //do nothing, just forward
          return mapping.findForward("Success");
    }
    else if (parm.equals("Display"))      {
          b.populateAll();
          return mapping.findForward("Success"); //forward to List
    }
    else if (parm.equals("Edit"))       {
          String id = request.getParameter("ID");
          b.populate(id);
          b.rowid=0;
          return mapping.findForward("Edit"); //forward to Edit
    }
    else if (parm.equals("Save"))       {
          b.save();
          return mapping.findForward("Saved");
    }
}
```

The JSP could send messages to this action with a URL like the following:

/do/userAdminAct?Dispatch=Edit&ID=2

The Save button on the edit page could also use this action with a dispatch parameter. The JSP for the edit page could look as follows:

```
<%@ taglib uri="struts/html" prefix="html" %>
<%@ taglib uri="jstl/c" prefix="c" %>
...
<html:form action="userAdminAct">
<table><tr>
  <td>ID: </td>
  <td><c:out value="userAdminBean.id"/></td>
</tr><tr>
  <td>Last name: </td>
  <td><html:text property="name_last" size="70"/></td>
</tr><tr>
  <td>First initial: </td>
  <td><html:text property="name_first_init" size="20"/></td>
</tr>
</table>
<html:submit property="Dispatch" value="Save"/>
</html:form>
```

...

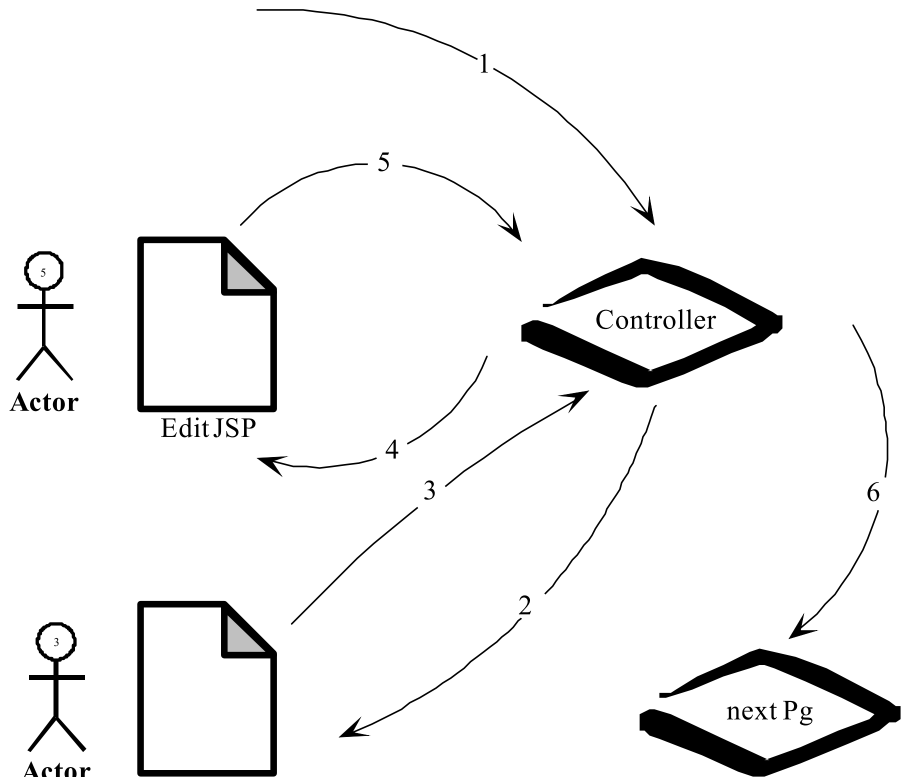
Clicking on the submit button will trigger the action user AdminAct with a request parameter named Dispatch and a value of Save. The action will ask the bean to save the data in the HTML form, and forward to the location defined in the forwards section of the action mapping in struts-config.xml.

Drill down

The action mapping for this action could look as follows:

```
<action path=”/userAdminAct”
        type=”com.basebeans.basicPortal.cms.UserAdminAct”
        name=”userAdminBean” scope=”session” validate=”false”>
    <forward name=”Success”
        path=”/WEB-INF/pgs/memberSec/userList.jsp”/>
    <forward name=”Edit”
        path=”/WEB-INF/pgs/memberSec/userDetail.jsp” />
    <forward name=”Saved”
        path=”/do/userAdminAct” redirect = “true”/>
</action>
```

The Success forward just displays the list page after the action has been executed. The Edit forward just shows the detail page after the bean has been positioned on the current row. The Saved forward goes though the action again to force a requery or refresh of the list before the list page is shown again.



A user calls the action, and defaults to a list or rows page. He clicks on a row, and a page is displayed that shows that one row, and he is able to edit the page. On a submit, the data is saved.

However, the code of the action class can further be improved upon. You will see in the following section how.

Events

In this section we use events to create simpler dispatching with action parameters.

Most developers are used to procedural programming where the programmer can control the flow. You have a green screen menu, with choices from 1-12. In this case, the program can keep track of the user's location in the flow fairly easily.

A user might decide to go to different sites and then come back to your site at an entirely different location. To adequately deal with this, the web application developer has to discard procedural programming in favor of event-oriented programming and code more defensively. Events need to be dealt with as soon as they occur. An example of an event is a handling a save dispatch.

Common Events

Over the course of doing a years of MVC development using Struts, the following events are commonly needed and could be reused, but you may think of more:

```
public String onDefaultExec(ActEvent ae) throws Exception;
public String onDisplayExec(ActEvent ae) throws Exception;
public String onEditExec(ActEvent ae) throws Exception;
public String onSaveExec(ActEvent ae) throws Exception;
public String onZoomExec(ActEvent ae) throws Exception;
public String onNewExec(ActEvent ae) throws Exception;
public String onInsertExec(ActEvent ae) throws Exception;
public String onLogonExec(ActEvent ae) throws Exception;
public String onNewSessionExec(ActEvent ae) throws Exception;
```

With events, you can bring reuse to another level. Since many events such as Save, Display are common to a lot of screens, a base action class can be set up to deal with them. Of course, the event-handling process itself is common to all such actions and is coded in the base action class.

Events mean that your concrete action class has handlers for actions. Rather than having to deal with the dispatching within the execute(...) method of the concrete action, there would be event-handling methods, such as the following:

```
public String onSaveExec(ActEvent ae) throws Exception {
    //do custom handling on save handling here
    return "Saved";
}
```

Using reflection, you can set up the base class to run this method when the action receives a Dispatch=Save parameter.

Action Event Object using Context

You can examine the execute signature for Struts Tiles action, and see that it is different than Struts action. This makes it a bit harder to reuse for tile action.

Also a Struts action is a singleton, so any properties in the action are not thread safe. All the methods are of course. So we could pass any parameters from method to method. We can create such an object, the action context, aka event object.

We could then simplify the log execute signature from this, to something simpler and easier to type.

```
public ActionForward execute(ActionMapping mapping,
             ActionForm formBean, HttpServletRequest req,
             HttpServletResponse resp) throws Exception {
```

We will instead only have:

public String onSaveExec(ActEvent ae) throws Exception {

You note that the method above receives an ActEvent which is also created in the base class and is simply a container for useful parameters, such as the current request, the bean instance and the ActionMapping. The following is a fragment of the ActEvent class:

```
public class ActEvent extends ContextBase
{
    protected HttpServletRequest _req = null;
    protected BaseBean _bean = null;
    protected ActionMapping _mapping = null;

 public void setReq(HttpServletRequest req) {
        this._req = req;
    }
 public HttpServletRequest getReq() {
        return _req;
        }
        ...
```

The way to access the ActEvent properties is with the usual getters and setters. At the least, the ActEvent will have the same four properties the original bean.execute() method has: request, response, form bean and action mapping. If you need to pass other information, you can just add more properties to the ActEvent class.

To do a save on the current bean instance, you would now just write the following:

```
public String onSaveExec(ActEvent ae) throws Exception
{
        ae.getBean().save();
        return “Saved”;
}
```

The ContextBase class is a HashMap from Struts Chains. It allows you to put properties in there in a thread safe way.

Extending the Action

By now, you will be curious to see the implementation of the base class:

```
public abstract class BaseAct extends Action implements Command
{
     public ActionForward execute(ActionMapping mapping, ActionForm
      formBean, HttpServletRequest req, HttpServletResponse resp)
          throws Exception
     {
ActEvent ae = new ActEvent();
try
          {
ae.setReq(req);
ae.setBean((BaseBean) formBean);
ae.setMapping(mapping);
processActScope(ae);
String parm = ae.getDispatch();
String forward = dispatch(ae);
if(forward == null)
forward = “Success”;
return mapping.findForward(forward);
          }
catch(Exception e)
        {
procE("B.exec", e, ae);
throw new Exception(e.getMessage());
        }
    }

protected final String dispatch(ActEvent ae)
throws Exception
    {
String parm = ae.getDispatch();
String forward = null;
String methodName = "on" + parm + "Exec";
try
        {
Class args[] = {ActEvent.class};
Method eventMethod = this.getClass().getMethod(methodName, args);
Object objs[] = {ae};
forward = (String) eventMethod.invoke(this, objs);
        }
catch(Exception e)
        {
procE("dispatch ", e, ae);
throw new Exception(e.getMessage());
        }
return forward;
    }
...
}
```

The interface command is a future part of Struts that allows for CoR pattern.

In addition we coded onFinalize(ActEvent ae) that executes when the user changes actions. This allows for clean up of the object in session.

Action Class With Events

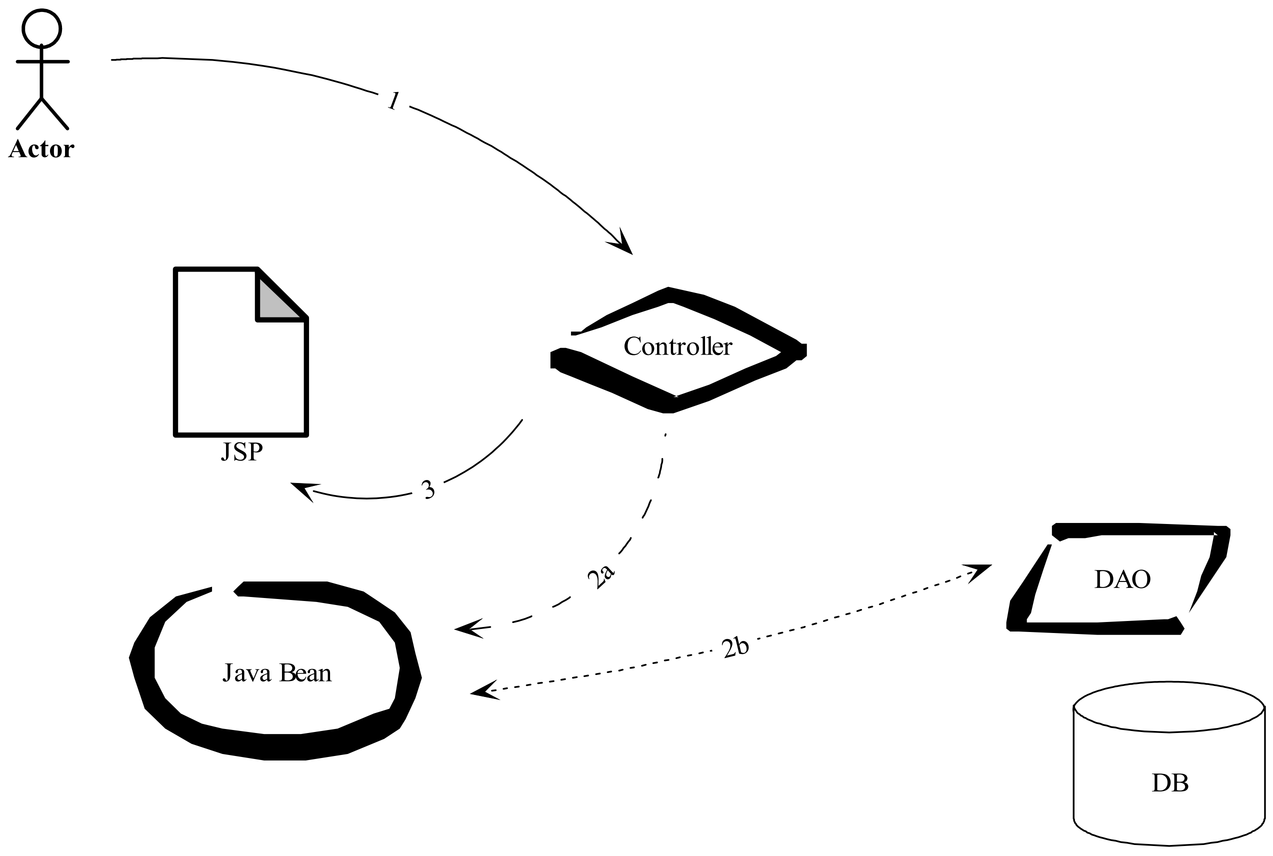
Using the base class BaseAct, we encapsulate the if-then dispatching and, therefore, simplify the code of our concrete action class. Our UserAdminAct class could now be rewritten, as follows:

```
public class UserAdminAct extends BaseAct{

public String onDisplayExec(ActEvent ae) throws Exception {

MyBean mb = new MyBean();

    mb.populateAll();
    ae.getReq().setAttribute("whatIwhatToCallItInJSP",mb);

    return "Success";

    }

public String onEditExec(ActEvent ae) throws Exception    {
          String id = ae.getRequest().getParameter("ID";
          ae.getBean().populate(id);
          return "Edit";

    }

public String onSaveExec(ActEvent ae) throws Exception    {

ae.getBean().save();

return "Saved";

    }

}
```

You notice that this is very clean code. You know where events are handled. The dispatching mechanism is hidden in the base class. Also, default code for each of the events is already in the base, so if you just need a plain vanilla onSaveExec, it is already in the base, you do not need to write it.

Typing mistakes in the forward names (success or Sucess instead of Success) are a common source for errors and debugging work. Therefore, you may find it useful to have String constants for them as shown above.

The entire flow is illustrated here:



Review

We can now the Struts action to make it more reusable and we can easily write a handler for a Save dispatch.

LAB O: More Simple Actions

Review the display tag possibilities and example display tag pages, in the display tag war.

Make sure Struts.xml has a form bean mapped for the edit action.

We will now integrate modules.

We will now move the populating of the DAO to an action, where we populate the query based on a URL parameter. Open **TaskEditAct.java**.

2. Modify as follows

```
public ActionForward execute(ActionMapping mapping,
                             ActionForm formBean,
                             HttpServletRequest req,
                             HttpServletResponse resp) throws
                             Exception {
String parm = req.getParameter("ID");
long l =  BUtil.longString(parm);


TaskBean tB = (TaskBean) formBean; // get a handle on Struts
tB.populate(l);


    log.debug("task edit action called:" + parm + tb.getTaskName());
return (mapping.findForward("Success"));

}//exec
```

3. In the browser, test the following URLs:
http://localhost/do/taskEditAct?ID=1
http://localhost/do/taskEditAct?ID=2

for the result.

4. Open TaskAct.java

5. Edit like this:

```
public ActionForward execute(ActionMapping mapping,
ActionForm formBean, HttpServletRequest req,
HttpServletResponse resp) throws Exception {
    TaskBean tB =   new TaskBean();
tB.populateAll();

    log.debug("task  action called:" + tb.getTaskName());
    req.setAttribute("myTaskBean",tB); // the name you will use in display tag
return (mapping.findForward("Success"));
    }//exec
```

Examine WEB-INF/portlets/cms/ContentAdminLst.jsp example.

Copy the display:table section to TaskLst.jsp.

Modify property names in display tag to match TaskBean getters.

(do not worry about href tag yet, delete it until next lab)

Make sure you "name" in display matches the name you gave in anction (setAttribute).

Navigate to /do/task

10. Test

We have completed Struts integration of modules.

LAB P: Messages

We will now send a URL message from a JSP to the action. Open the TaskLst.jsp

Modify it to say:

```
<display:column
property=”taskName”
width=”50%”
title=”Task Name”
styleClass=”text”
href=”/do/taskEdit?Dispatch=Edit”
paramId=”ID”
paramProperty=”id” />
```

This assumes you have a property called getId() in bean that works. (unit tested).

Test.
You should be able to navigate to the task list page from the menu.
You should be able to click on one of the names on taskList and zoom the detail, for editing.

LAB Q: Advanced Action /w Save Event

We could do manual handling of Parm in each action w/o base code reuse or OO, and some Struts projects do. This is a more advanced action. We will have one action w/ more than one forward, with all if then logic in base.

1. Open the TasksAct.
Modify edit action to Extend BaseAct.

Examine the code in UserAdminAct in memberSec package. Examine the code in NewsBlgCmntsLstAct in cms.newsBlg package.

Examine BaseAct code (by right clicking on BaseAct in IDE)

(BaseAct takes the parm argument and generates an event)

study at onEditExec
study at onSaveExec
study at onDisplayExec

Note how req, resp and formBean are in ActEvent object now, making a cleaner execute signature.

If we do not have a method in a concrete class that extends BaseAct, the default is base method.

We will combine 2 actions into one action with more than one forward . Open struts-config.

Remove the entire mapping to TaskEdit. Now we only have Task Action.
Also rename (refactor) the TaskEditAct.java (to TaskEditOLD.java) so as to not be confused.
In the mapping task, add a forward for edit, and save.
Ex, this should be the only task action:

```
<action path =”/tasks”          type=”com.yourOrg.TaskAct”
  name=”taskBean”       scope=”session”               validate=”false”>
   <forward name=”Success” path=”taskLst.pg” />
   <forward name=”Edit”      path=”taskEdit.pg”  />
   <forward name=”Saved”    path=”/do/tasks “  redirect = “true” />
</action>
```

Remove the execute method in TaskAct().

Add code for:

    On Display event, forward to list.
    On Edit, forward to the edit page.
    On Save, save.
Ex:

```
public String onDisplayExec(ActEvent ae) throws Exception {
    XBean XBean = (XBean) ae.getBean();
    xBean.populateAll();
System.out.println("we have list data:" + xBean.getName()); //debug
ae.getReq().setAttribute("myBeanInScope",xBean); //match display tag name
    return "Success";
     }

    public String onEditExec(ActEvent ae) throws Exception {
    XBean xbean = (XBean) ae.getBean();
    xbean.populate(ae.getParmID());
    // no attribute! And not create!
System.out.println("we have zoom data:" + xBean.getName());
    return "Edit";
     }


public String onSaveExec(ActEvent ae) throws Exception {
    xBean bean = (xBean) ae.getBean();
    System.out.println("saving");  // should work if we unit tested bean

    xbean.save(); // bean asks it's DAO helper to save
    return "Saved";
      }


<html:submit property="Dispatch" value="Save"/>
```

// again, make sure that old action and action mapping are not referenced anywhere, and are deleted.

6. Test a save. Did the data change in DB?

This would complete the next iteration of a project. We have taken it from a prototype, to a integrated read only application. (In real life, we would have to do entire app., all the screens /reports, as read only 1st, before any fancy work)

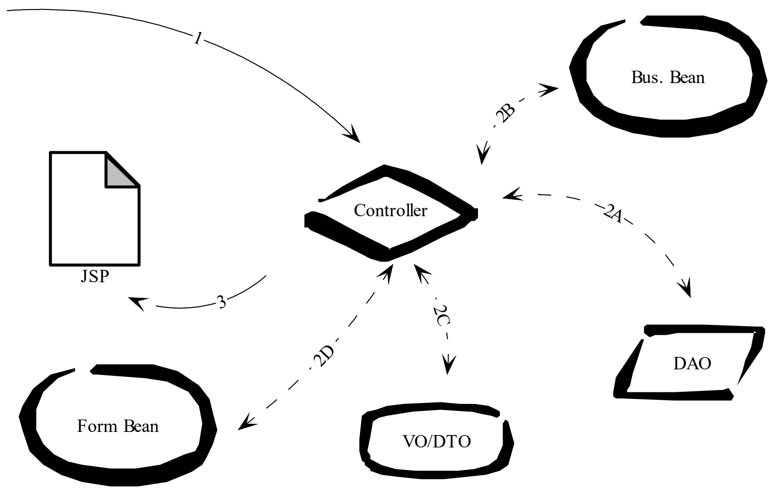
Chapter 11: More Form Handling Action

Alternative MVC Flow Options

There are other approaches which merit being mentioned in comparison.

Some approaches work with JSTL, others do not. Some work with multiple rows, others do not. Some approaches are easier to unit test than others.

Some use Value Objects (VO) and Business Object Beans (BO).



These are the other approaches:

A. Simple VO
Going to the JSP, a helper class copies from the VO to the form bean.
Going back, the action copies from the form bean to VO with a helper factory.

B. Form bean implementing VO
Going to the JSP, a BO populates the VO-form bean.
Going back, the action passes VO-form bean to the BO.

C. Form bean with VO facade
Bean exposes a single-row VO with a getter/setter. The VO has getters and setters for properties.

Going to the JSP, an action passes the VO from BO to form bean.
Going back, an action passes the VO from form bean to BO.

D. Form bean with DAO helper
Going to the JSP, action asks bean to populate from DAO.
Going back, action asks bean to save to DAO.

This book recommends approach D. It works with JSTL, is easy to unit test, has little code and few objects, which is good for performance (less VM garbage collection).

The approaches which use Value Objects are an older way of using Struts; it was the original design created by Struts developers. It was used to be able to keep EJBs outside the model, but had scalability issues. There are variations, but the flow ususally includes these following steps:

1. The controller passes a connection to the DAO (which is usually an O/R implementation). The DAO returns data and the controller passes it to a BO bean.

2. The controller or a controller helper passes the data via a VO to the form bean.

You are likely to hit issues with multi-row processing. Since multi-row form beans work with arrays, you are likely to need a second set of beans.

These architectures have at least two beans and a VO, which creates a lot of work for the JVM garbage collector. Also, there is a lot of non-controller code in the action. Good MVC does not have model code in the action; the action should not work with data. These architectures are also a lot more complex than the recommended architecture “Bean with DAO helper”. It is almost always a good idea to follow the “keep it simple” or KISS approach.

Forms are sufficiently complex by themselves so you do not want to make your life more difficult than absolutely necessary. Taking the example of a tax form: the corresponding bean will have getters and setters for a person’s name. It may also contain two nested beans, one for a list of incomes and one for a list of expenses. If you were using the VO approach, you would quickly have a very complex and confusing structure at your hands.

A quick note on a variation of the VO approach which uses DynaBeans: this architecture does not have a form bean, just a business bean. XML mappings of property names are created within struts-config.xml. This approach is hard to unit test since DynaBeans cannot be used without Struts.

If you follow the recommended approach, you develop code so that it is most easily understood by the people who read your code. Readability leads to reuse and reuse leads to higher quality and higher reliability.

*** Forward**

A new syntax that is available since Struts 1.2 release it the * forward lke this:

```
<action   path="/*Edit"
        type="myorg.example.{1}EditAct"
        name="{1}FormBean" >
  <forward name="success"   path="/WEB-INF/jsp/{1}.jsp"/>
</action>
```

It will look at any request’s mapping and match it to the *, so for example /do/customerEdit, will have the {1} replaced by “Customer”.

This does allow for less mapping. You may have to put a use case in a same package as other uses cases, and doing mapping is not that hard.

Insert Event

We can also do insert events in our design from prior chapter by adding the following to the JSP which has so far been for display only:

```
<html:form action=”/userAdminAct”>
    <html:submit property=”Dispatch” value=”New”/>
</html:form>
```

You now have access to the full power of events. You would code your event as follows:

```
public String onNewExec(ActEvent ae) throws Exception
{
    UserBean b = ae.getBean();
    b.insertNew();

    return BConst.EDITD;
}
```

In the code above, you just add a blank row in the bean. The JSP will display a page with the blank row. The action needs to set all default values, such as foreign keys.

This insert event adds a row in memory only. Once the user has filled the form, he would use the submit button with Dispatch=Save to save the inserted data to the database. This is the same action used for updating modified data on an existing row of data.

Exception Handling

Before exceptions, code looked like this:

```
int x;
x = doA();
if (x < 0) {
doError1();
}
x=doB();
if (x < 0) {
doError2();
}
```

Notice how the returned values are checked to see if doA and doB return negative results. This results in code that is hard to read. Good paths (error-free flows) and bad paths (error checking and handling) are mixed.

With the advent of exceptions as a means to deal with errors, code can be made to look like this:

```
try{
doA();
doB();
}catch (Exception e){
if (e==x){...}
}
```

The good paths are separated from the bad paths, and the code becomes more maintainable. Some people tend to create their own series of Exception objects (MyCompanyException1, MyCompanyException2, etc). The authors believe that the standard exceptions provided by Java are fully sufficient for most purposes since they can be adapted by providing custom messages with exception.setMessage(”My Error message”). The code will be more readable.

Exception Flow

All the exceptions bubble up to the container, via the action.

All DAO methods should log and re-throw exceptions. It is impossible to guarantee network and database connectivity to 100 per cent. DAOs are likely to generate the most of all exceptions, both in development and in production. The DAO should log any exception and re throw it to its bean, so that the respective action can find out about it. This is an example for this mechanism in the DAO:

```
public populate(long i) throws Exception{

    try {...}
    catch (Exception e)     {

           log.error(e);

           throw new Exception(e);

    }

}
```

Beans might not need to handle exceptions. Beans delegate work to DAOs and should not do much work with data results other than reformatting and remapping. In the bean, throw the exceptions up to the actions as follows:

```
public save(long i) throws Exception{

    dao.update(i);
}
```

An exception thrown by the DAO is automatically passed on to the action which has called the save(...) method on the bean.

Actions receive the exceptions passed on to them by beans. They will handle the exceptions and sometimes throw them to the container so that an error screen will be displayed to the user. The following is an example of an exception thrown by an action to the caller. The caller in this case is BaseAct, which will provide some code to display the error to the user. The action could have code as follows:

```
public String onDisplayExec(ActEvent ae) throws Exception{

try

    {

UserBean b = new UserBean();

b.populate(ae.getRequest().getParameter("ID"));

           return BConst.SUCCESSF;
    }      catch (Exception e)     {

           log.error(e);

           throw new Exception(e);
```

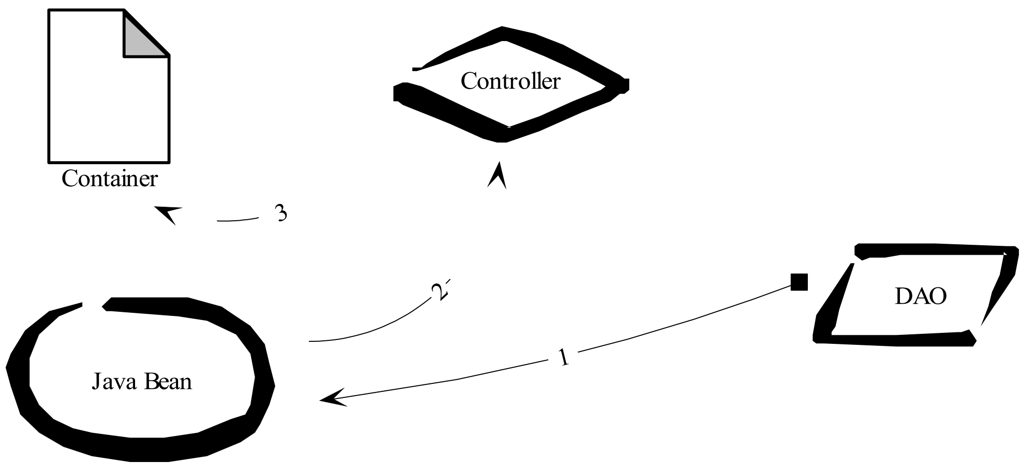
We also set up web.xml to display the normal errors in a user friendly way.

```
<error-page>

    <exception-type>java.sql.SQLException</exception-type>
```

```
        <location>/error.jsp</location>
    </error-page>
    <error-page>
        <exception-type>java.lang.Exception</exception-type>
        <location>/error.jsp</location>
    </error-page>
```

Note that the action exception exposes it to the container, so that in production we can check the application server's log to see any issues.

These exceptions indicate a hardware, network, or software problem. You must check the log file of production systems regularly to be able to remedy such problems.



Multi-row Update

Struts has built in support for multi row updates, an update where you change more than one row and submit. Lets say you have firstName in row 1 and firstName in row 2, etc. What happens on a submit? The data entry field name_last may be repeated 10 times on a page, and each time the name of the entry field is the same. So how does the server know which row of data maps to which fields?

Struts has an indexed property available for this. A JSP that is multi-row updatable might look like this:

```
<c:forEach var="row" items="${userBean}">
<tr>
<td>
          <html:text property="name_last" name="row" indexed="true"
                size="50"/>
    </td>
</tr>
</c:forEach>
```

For the above to work, the source of multi-row data (taskBean) has to implement Collection. Magically, the rows present in the bean are mapped to the fields, and values transported forth and back. You could use the indexed property in more complex ways, for example referring to a property of the bean which implements Collection, but the easiest to handle is when you implement the Collection interface on the bean itself.

Transactions

The DAO has tBegin() and tCommit() methods to indicate the boundaries of a transaction.

Sometimes transactions span several database engines and we need a two-phased commit. The DAO can do it, and we just have to write a custom save method on the bean which should receive the instruction to the complex update from the action. The code in the bean could look as follows:

```
public void save() throws Exception
{
    _dao.tBegin();
    this.save();
    bean2.save();
    bean3.save();
    _dao.tCommit();
}
```

Master-Detail Processing

One-to-many relationships are omnipresent in database designs. The following figure shows a master-detail relationship between a project and its tasks.

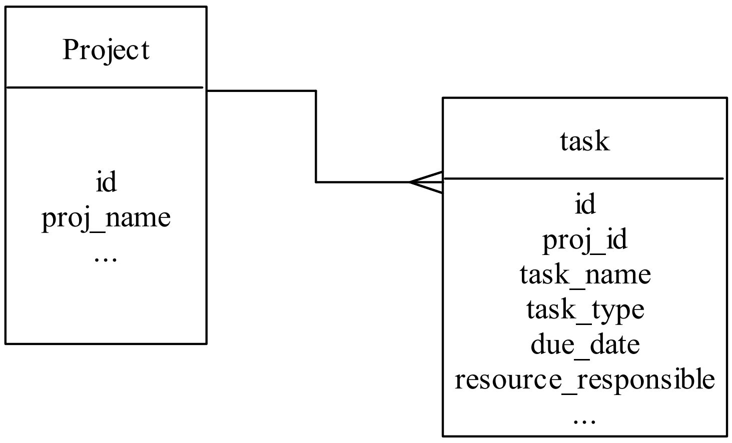


Figure 11.1: Master-detail relationship

The project is the master, the tasks are the details. The code should join the tables with project.id and task.proj_id, where task.proj_id is the foreign key.

The screens could have a list of projects, with the ability to add a task to a certain project. The primary key project.id needs to be carried over to the detail table when generating new detail rows. This should be taken care of in the action that prepopulates the task bean. If this step is omitted, new tasks would have a blank project_id, and the relationship of a task to project would be unclear.

The code in the action could look as follows:

```
public String onNewExec(ActEvent ae) throws Exception
{
TaskBean b = ae.getBean();
b.insertNew();


//Get the foreign key from the request
    Integer project_id =
          ae.getRequest().getParameter("project_id");
    b.setProjectId(project_id);


    return BConst.EDITD;
}
```

The above code assumes that the project_id was passed to the action as a request parameter as in /myTaskAct?project_id=XX.

This may have been generated from a list of projects with a JSP similar to the following:

```
<%@ taglib uri="struts/logic" prefix="logic" %>
<%@ taglib uri="jstl/c" prefix="c" %>
...
<table>
<logic:iterate id="row" name="projectBean" property="displayList">
<tr>
      <td><c:out value="${row.id}"/></td>
      <td><c:out value="${row.proj_name}"/></td>
      <td> <a href="/do/taskAct?Dispatch=New&project_id=<c:out
                              value="${row.id}"/>">Add task</a> </td>
</tr>
</logic:iterate>
</table>
...
```

Bookmarkable action

One nice feature of the Web: to bookmark or place a Favorite on a specific page of the application and come back to it in one step, without having to go through a series of operations and without having to click oneself through to the requested location.

Any solid web application should provide “bookmarkable URLs” that allow the user to go back to the preferred place, even after weeks have passed, by going to one specific, reliable URL. This means that a maximum amount of the state of the application should be reflected in the URLs. The web application should be able to recover with the information provided in the URL and lead the user to the same screen even after long interruptions (possibly after having logged in again).

One way to do this is to go via a tile definition mapping like this:

```
      <definition name="userDetailDef" extends="admDef">

            <put name="title" content="User Detail"/>

            <put name="left" content="../bpadmin/leftPage.jsp"/>

            <put name="main" content="/do/userDetailAct"/>

      </definition>
```

Review

We covered inserting, multirow updates and exeptions.

Optional LAB R: sql.date

Examine the javascripttoolbox tag in war samples.

Review conversion from SQL date to Calendar in BaseBeans.

Review conversion from text to SQL date in BaseBeans.

Modify the edit task jsp to allow inputting of the task due date and to use the pop up calendar.

LAB S: Multi Row Update

Review the JSTL document. In docs or linked on cheat sheet page.

Create a new action mapping, a new JSP (TaskMR.jsp)

Create an action (TaskMRAct) that extends the working TaskAct. You should not need to change anything.

3. Link to the new action from the ListTask JSP use href (/do/taskMR)

4. Create a new tiles layout mapping, Ex: TaskMR.def.

5. See if you can navigate to a blank page, simple action.

6. Look at example (ContentAprvMR.jsp, run and view source)

Edit the MRTask.

Add a form tag, forEach, and indexed text tag. Ex:

```
<c:forEach var=”row” items=”${ myBeanInScope }”>
<td>

    <html:text styleClass=”form” property=”taskName” size=”20” name=”row”
    indexed=”true”/>
    </td>
```

We use getRow variable (in baseBean) and indexed tag to enable Multi Row.

8.Test Submit changes to 2 rows. View browser source

9. Check DB.

LAB U (Optional): On New

Add code to allow users to request a new task be added, when they are on the list page.

In task list jsp, add a page to dispatch a New event. Ex:

<html:form action=”/taskPg”>

<html:submit property=”Dispatch” value=”New”/></html:form>

2. Make sure DAO XML can do insert of SQL. Test that you can insert a row in pgSQL. (You can get SQL help from postgresql.org site or cheat sheet page)

3. Unit test bean to do insert, setters and then save.

Ex:

```
testIt() {
newRow();
setTaskName(“Feed the Army”);
save();

}
```

Did the data change in the database?

You might need to make sure that users have right to generate primary key in DB, such as:

GRANT ALL ON tsk_id_seq TO bpuser;

Add a onNewExec method in TaskAct()

It should insert a blank row to the “list”.

It should forward to the edit page. Look at examples of onNew.

5. Note that once user populates this blank list, and clicks save, the same update event handles the new data.

6. Test the web app. Did the data change in the database?

You now have a simple CRUD iteration, your previous iteration was read only.

Chapter 12: Validation

You will complete the picture of flow handling with input validation. Struts provides a validation framework that allows server-side validation as well as client-side validation using JavaScript. Struts allows declarative validation, outside Java and configured in one or several XML files. The following is a sample validation.xml:

```
<form-validation>
<formset>
<form name="userBean">
<field property="nameLast" depends="required,minlength">
<arg0 key="user.name_last"/>
<arg1 name="minlength" key="${var:minlength}"
resource="false"/>
<var>
<var-name>minlength</var-name>
<var-value>2</var-value>
</var>
</field>
<field property="nameFirst" depends="required,minlength">
<arg0 key="user.name_first"/>
<arg1 name="minlength" key="${var:minlength}"
resource="false"/><var>
<var-name>minlength</var-name>
<var-value>1</var-value>
</var>
</field>
</form>
```

...

This assumes that the Struts form to be validated (userBean) has the properties getNameLast() and getNameFirst(). With a listing in the depends property, several validation rules can be combined in the order of preference, such as required, minlength. If a field is empty, a message to this effect is generated. If it is not empty, but the length is not sufficient, a different message is generated.

The key property is combined with the rule names to generate the respective message. For easy localization and customization, message strings are located in the properties file at properties/messages_xx.properties.

By default, Struts comes with properties file containing entries as follows:

```
# Errors

errors.footer=

errors.header=<h3><font color="red">Validation Error</font></h3>You
must correct the following error(s) before proceeding:

errors.ioException=I/O exception rendering error messages: {0}

errors.required={0} is required.

errors.minlength={0} cannot be less than {1} characters.

errors.maxlength={0} cannot be greater than {1} characters.

errors.invalid={0} is invalid.

errors.byte={0} must be an byte.

errors.short={0} must be an short.

errors.integer={0} must be an integer.

errors.long={0} must be an long.

errors.float={0} must be an float.

errors.double={0} must be an double.

errors.date={0} is not a date.

errors.range={0} is not in the range {1} through {2}.

errors.creditcard={0} is not a valid credit card number.

errors.email={0} is an invalid e-mail address.
```

A message generated by the validation framework would then be
“The last name is required.” or “The last name cannot be less than 2 characters.” You are free to customize the messages and provide a degree of politeness customary in your culture.

To activate Struts validation, and register the above validation.xml, you add an entry to struts-config.xml similar to the following:

```
<plug-in className="org.apache.struts.validator.ValidatorPlugIn">
                <set-property property="pathnames"
                      value="/WEB-INF/lib/validator-rules.xml,
                            /WEB-INF/config/validation.xml"/>
</plug-in>
```

Several validation files can be added with the comma separator. Default validations provided by Struts (such as the basic rules the custom validation depends on like required, minlength, maxlength, etc.) are found in validator-rules.xml.

Validation Messages

How are validation messages generated and put in scope? Struts provides the methods validate(mapping, request) and saveErrors(request, actionerrors) for this. A typical implementation may look similar to the following action code:

```
public String onSaveExec(ActEvent ae) throws Exception {
        ActionMessages errors;
              BaseBean bean = (BaseBean) ae.getBean();
              errors = bean.validate(ae.getMapping(), ae.getReq());

             if (errors.isEmpty()) {
                    bean.save();
                    return "Saved";
             } else {
                    this.saveErrors(ae.getReq(), errors);
                    return "Edit";
            } // else
   }
```

Of course, this implementation should be a part of BaseAct for reuse.

The validate(..) method can produce one or more validation errors at the same time (Last name could be too short and First initial could be empty). If any action error is returned, the action forward should bring the user back to the page which produced the error, after putting the errors in scope and thus make them available for display. However, you could also forward to a separate error page.

Displaying Validation Errors

One Struts tag can take care of displaying all validation errors. This is the <html:errors/> tag. A JSP that will display error messages at the top of the input page may look as follows:

```
<%@ taglib uri="struts/html" prefix="html" %>
<%@ taglib uri="jstl/c" prefix="c" %>

...

<html:errors/>
<html:form action="userAdminAct">
<table><tr>
  <td>ID: </td>
  <td><c:out value="userAdminBean.id"/></td>
</tr><tr>
  <td>Last name: </td>
  <td><html:text property="nameLast" size="70"/></td>
</tr><tr>
  <td>First initial: </td>
  <td><html:text property="nameFirst" size="20"/></td>
</tr>
</table>
<html:submit property="Dispatch" value="Save"/>
</html:form>
...
```

The errors tag will display the action errors which the bean method saveErrors(...) put in scope earlier. Default entries in messages.properties provide formatting prefixes and postfixes, such as:

```
errors.header=<h3><font color="red">Validation Error</font></h3>You
must correct the following error(s) before proceeding:
errors.prefix=<b>

errors.postifx=</b>
errors.footer=
```

If your validation messages themselves contain raw HTML, you may use the html:messages tag for full control, such as in the following:

```
<html:messages id="error" locale="en">

<span class="validation"><font face="Helvetica">

<c:out value="${error}" escapeXml="false"/>

</font></span><br>

</html:messages>
```

The messages tag can be used to display any messages, not limited to action errors. The implementation shown above iterates over the action errors, which are put in scope under the key id error.

Furthermore, you can create your own action errors in addition to those provided by Struts validation. This could be achieved in the execute method of the BaseAct as follows:

```
catch (Exception e)
{
ActionMessages errors = new ActionErrors();
errors.add(ActionErrors.GLOBAL_ERROR,
new ActionMessage("errors.formError",
e.getMessage()==null?e.toString():e.getMessage()));
saveErrors(request, errors);
return getActionMapping().findForward("FormError");
}
```

This would display all error messages on exceptions caught (such as database connectivity problems) on a separate page under the forward FormError. As required for the html:errors tag, errors are stored in request scope under the key "error", or `ActionErrors.GLOBAL_ERROR` . In conjunction with the messages tag shown above, the file messages_en.properties would have to contain an entry like the following:

```
errors.formError=The following error was encountered when
processing this page: <br>{0}
```

Client Side Validation

Struts validation can dynamically generate JavaScript that performs client-side validation based on the same validation.xml declaration used for server-side validation. In the JSP, validation errors can be displayed as javascript alerts by adding the script and intercepting the form submit. The corresponding JSP form would look as follows:

```
<%@ taglib uri="struts/html" prefix="html" %>
<%@ taglib uri="jstl/c" prefix="c" %>

...

<html:errors/>
<html:form action="userAdminAct"
    onsubmit="return validateUserAdminBean(this);">
<table><tr>
  <td>ID: </td>
  <td><c:out value="userAdminBean.id"/></td>
</tr><tr>
  <td>Last name: </td>
  <td><html:text property="nameLast" size="70"/></td>
```

```
</tr><tr>
  <td>First initial: </td>
  <td><html:text property="nameFirst" size="20"/></td>
</tr>
</table>
<html:submit property="Dispatch" value="Save"/>
</html:form>
<html:javascript formName="userAdminBean"/>
...
```

You would add this kind of client-side validation after the server-side validation works.

Business Rules

An example of a complex validation is not to allow an order if a client exceeds the permitted credit limit.

You might tag your business rules with an interface such as this:

```
public interface BusinessRule   {
   public ActionMessages validate (BaseBean b, String
rules_name) ;}
```

By tagging the rules with an interface you know where to find them. It the business rule, typically you create DAO's to check tables, using a Business Rules SQL mappings you create.

To use this rule, you should override the Struts validate(…) method in baseBean/validatorBean. First call super.validate to check any simple declared rules in XML. Then create your business rules objects, and append any Messages to return. As always you can unit test the bean.

Then to use rules, check them from your action before a save as in:

ActionMessages errors = formBean.validate(…);

If there are messages, that means you want to put them in scope and forward back to the input page:

this.saveErrors(ae.getReq(), errors);

You could also do a CoR pattern to call other business rules.

Error page

Any exceptions thrown by action are visible by the container.

<error-page>
<exception-type>java.lang.Exception</exception-type>
<location>/error.jsp</location>
</error-page>

Duplicate submit

During QA or production, sometimes people do multiple submits. This can confuse any application server. Struts has built in handling for submits, using an autogenerated request token. Here is an example of preventing duplicate submits, even when user spawns another browser duplicating the session:

```
public String onEditExec(ActEvent ae) throws Exception {
        saveToken(ae.getReq());
        BaseBean bean = (BaseBean) ae.getBean();
        bean.populate(ae.getParmID());
        return "Edit";
}
public String onSaveExec(ActEvent ae) throws Exception {
        ActionMessages errors;
        if (isTokenValid(ae.getReq())) {
              resetToken(ae.getReq()); //method on Struts base
class
              saveToken(ae.getReq());//method on Struts base class


              BaseBean bean = (BaseBean) ae.getBean();
              errors = bean.validate(ae.getMapping(), ae.getReq());
             if (errors.isEmpty()) {
                    bean.save();
                    return "Saved";
             } else {
                    this.saveErrors(ae.getReq(), errors);
                    return "Edit";
            } // else
            } // if token
```

```
          else {
          return onTokenError(ae); //submit the duplicate entry event
  } // on save
```

Review

We have now reviewed validation.

Preview

At the end of part two of this book, you have created a solid and extremely efficient application infrastructure, which allows you to benefit from a high degree of reuse. You have the building blocks to create a high performance application. With this infrastructure, you can also build modules rather quickly. The next part will focus on providing richer user interfaces and ensuring that an application will scale well in a production environment, and a look into the future.

LAB V : Validation

Edit validation XML in WEB-INF/config. (location is set in struts config xml plugin configuration).

Note it has validation for a form bean userBean (name must match Struts-config form bean declaration), and a few properties are declared as "required,minlength".

Run the app. Navigate to join as a user, but leave all the fields blank, and click save. See the error message. This is what we will add to tasks.

Add validation for the tasks form bean.

Make some fields required and minimum length.

Edit TaskEdit.jsp. Add the errors tag to display messages:

<html:errors />

Edit task Act. On Save Exec, call the validate method.

Put error message in scope (look at example code in base)

Ex:

```
public String onSaveExec(ActEvent ae) throws Exception {
ActionMessages errors;
BaseBean bean = (BaseBean) ae.getBean();
errors = bean.validate(ae.getMapping(), ae.getReq());
if (errors.isEmpty()) {
bean.save(); // no errors
return BConst.SAVEF;
} else {
this.saveErrors(ae.getReq(), errors); // put it in scope
return BConst.VALIDERRORF;
 } // if
 } // saveExec
```

6. Test server side validation.
7. Add client side code to emit and call JavaScript (like User.jsp in membersSec)

```
<html:form action=”/TaskPg” onsubmit=”return validateTaskBean(this);”>

. .
. .

<html:javascript formName = “taskBean”/>
```

7. Test client side validation

Controlling complexity is the essence of computer programming.

-- Brian Kernighan

Part III

Perspectives

Chapter 13: Dynamic Site Content

Dynamic Content

The problem with static HTML and static HTML servers is that the content is hard to manage. A dynamic content site allows for CMS and portals, served by an application server. When developing your content application, you should of course apply MVC.

All dynamic content of a site (such as, but not limited to, text on the site) can be delivered by a generic action which uses a query parameter that identifies the content in the database. The search module above used the database column page_url as key for locating the content. This key would not be a physical URL, but rather a unique telling string. Accordingly, an action URL could look as follows:

http://www.basebeans.com/do/cmsAct?content=TRAINING.

In this example, TRAINING is the content key. The action cmsAct would pass the content parameter to the DAO that would use it to find rows in the database containing the desired content, an HTML document that talks about training services in this case.

The sql-map contentSrchSQL shown previously would have an additional mapped statement as follows:

```
<sql-map name='contentSrchSQL'>
<cache-model name ='content_cache' reference-type='WEAK'>
<flush-interval minutes='10'/>
<flush-on-execute statement='ContentInsert' />
<flush-on-execute statement='ContentUpdate' />
</cache-model>
<result-map name='result' class='java.util.HashMap'>
    <property name='page_url' column='PAGE_URL' />
    <property name='title' column='TITLE' />
    <property name='author' column='AUTHOR' />
    <property name='content' column='CONTENT' />
    <property name='content_type' column='CONTENT_TYPE' />
    <property name='topic' column='TOPIC' />
    <property name='id' column='ID' />

</result-map>
<mapped-statement name='ContentSrch' result-map='result'
          inline-parameters='true'>
    SELECT page_url, title, author FROM CONTENT
          WHERE title LIKE '#stitle#' AND
          CONTENT LIKE '#scontent#'
</mapped-statement>
<mapped-statement name='ContentSelectPage' result-map='result'
          cache-model='content_cache' inline-parameters='true'>
    SELECT page_url, title, author, content, content_type, topic,
          id
    FROM CONTENT
    WHERE page_url = #value#
    AND content_type = 'PAGE'
</mapped-statement>
```

Note that a content cache is also used here: the content returned by the query is maintained in a cache for ten minutes or until a new item is inserted or an existing item is modified. Further, you verify that the result-map contains all the columns of the SELECT statement of the mapped statement ContentSelectPage.
The ContentDAO will use the newly added mapped statement in a method that could look as follows:

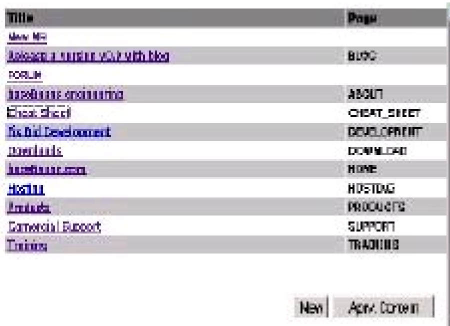
```
public void populate(Object o) throws Exception

{

    doRetrieve("ContentSelectPage", (String) o);
}
```

A c:out tag can emit this content on the JSP, which could look as follows:

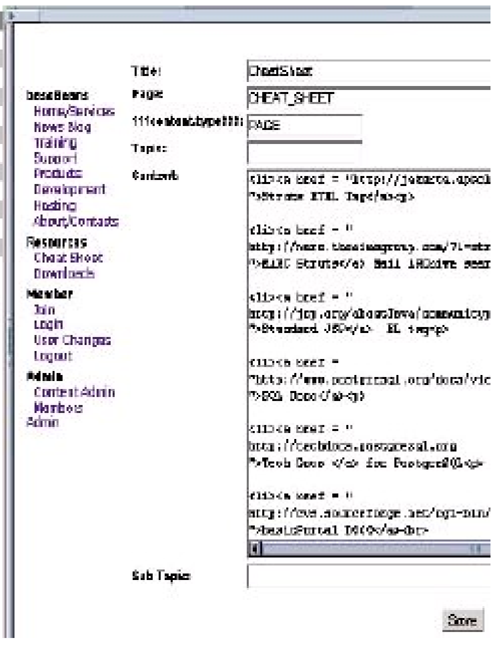
```
<c:out value="${contentBean.content}" escapeXml="false"/>
```

Since the content is likely to contain HTML tags, the excapeXml parameter with value of false will avoid an automatic conversion to visible characters (such as the conversion from < to &lt; in order that the browser will display < as in X<Y).

Content Administration



Title	Page
New MR	
Released version v0.2 with blog	BLOG
FORUM	
baseBeans engineering	ABOUT
Cheat Sheet	CHEAT_SHEET
Fix Bid Development	DEVELOPMENT
Downloads	DOWNLOAD
baseBeans.com	HOME
Hosting	HOSTING
Products	PRODUCTS
Commercial Support	SUPPORT
Training	TRAINING



Title: CheatSheet
Page: CHEAT_SHEET
[illegible]: PAGE
Topic:
Content:

```
<li><a href = "http://jakarta.apach
">Struts HTML Tags</a><p>

<li><a href = "
http://[illegible]
">[illegible] Struts</a> [illegible]

<li><a href = "
http://jcp.org/aboutJava/communityp
">Standard JSP</a>  EL tags<p>

<li><a href = "
"http://www.postgresql.org/docs/vie
">SQL Docs</a><p>

<li><a href = "
http://techdocs.postgresql.org
">Tech Docs </a> for PostgreSQL<p>

<li><a href = "
http://sve.sourceforge.net/cgi-bin/
">basicPortal [illegible]</a><br>
```

Sub Topic:

The screenshots illustrate what a content administration module could look like. A list page would provide a list of all content items available, with links to the detail page for editing.

Implementing Content Search

One of the key functionalities that a dynamic site should be able to provide is content search. A typical search screen would look as follows:

```
<%@ taglib uri="struts/html" prefix="html" %>
...
<html:form action="/contentAdminAct">
<table><tr>
  <td>Find title: </td>
  <td><html:text property="title" size="40"/> </td>
</tr><tr>
  <td>Find text: </td>
  <td><html:text property="content" size="80"/> </td>
</tr>
</table>
  <html:submit property="Dispatch" value="Click to searchs"/>
</html:form>
```

...

A typical result screen would look as follows:

```
<%@ taglib uri="jstl/c" prefix="c" %>
...
<table>
<c:forEach id="row" name="contentBean" property="displayList">
<tr>
    <td>

          <a href="/do/cmsPg?content=<c:out
                      value="${row.page_url}"/>">
                <c:out value="${row.title}"/>
          </a>
    </td>

</tr>
<tr>

<td>

          Author: <c:out value="${row.author}"/>
    </td>

</tr>
</c:forEach>
</table>
...
```

The above will display a list of search results, each with a link to the actual content, and the name of the author.

Mapping Search SQL

You will have to provide two query parameters to the content DAO: sTitle and sContent. When writing the SQL map for the DAO, you would first test the SQL command to use in an SQL console, such as:

```
SELECT page_url, title, author FROM content
    WHERE title LIKE ‘%a%’ AND
    content LIKE ‘%a%’
```

If the SQL works in the SQL console, you can be sure that it works in the DAO mapping. The quick test in the SQL console will avoid having to debug later.

The DAO mapping can then be set up as follows:

```
<sql-map name=’contentSrchSQL’>
<result-map name=’result’ class=’java.util.HashMap’>
    <property name=’page_url’ column=’PAGE_URL’ />
    <property name=’title’ column=’TITLE’ />
    <property name=’author’ column=’AUTHOR’ />
    <property name=’content’ column=’CONTENT’ />
    <property name=’content_type’ column=’CONTENT_TYPE’ />

    <property name=’topic’ column=’TOPIC’ />

    <property name=’id’ column=’ID’ />

</result-map>
<mapped-statement name=’ContentSrch’ result-map=’result’
          inline-parameters=’true’>
    SELECT page_url, title, author FROM CONTENT
          WHERE title LIKE ‘#stitle#’ AND
          CONTENT LIKE ‘#scontent#’
</mapped-statement>
```

Note that the mapped statement in the SQL map expects to receive stitle and scontent as parameters. Also, the mapped statement ContentSrch does not return the content itself since the content is not displayed in the result list (only title, author etc. is shown). The result map, however, contains all columns since we want use them in more detailed queries later. It is not required to fill all columns of a result map with every mapped statement.

Form Bean for Search Results

The form bean that passes the search parameters to the DAO would look as follows:

```
public class ContentBean extends BaseBean {
protected ContentDAO _contentDAO = null;
public ContentBean()  {
_contentDAO = new ContentDAO();
_dao = _contentDAO();
    }
public void populateSrch(String sTitle, String sContent)
throws Exception      {
Map parm = new HashMap();
          sTitle = "%"+sTitle+"%";
          sContent = "%"+sContent+"%";


          parm.put("stitle", sTitle);
          parm.put("scontent", sContent);

          _contentDAO.populateSrch(parm);


          _rlist = new ArrayList();
          _rlist.addAll(_dao.getResList());
          setupList();
    }
public String testIt() throws Exception {
populateSrch("a", "a");
return _dao.getResList().toString();
    }
}
```

ContentDAO will reuse generic query functionality in its base class as follows:

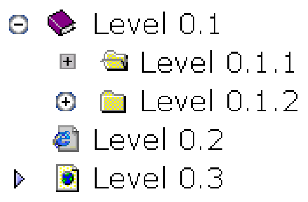
```
public void populateSrch(Map parm) throws Exception {
    doRetrieve("ContentSrch", parm); //method in baseClass
}
```

When the contentSrchAct is called, the above code will pass the search parameters to the DAO that will retrieve the items using the mapped statement defined there.

ContentBean is populated with the list of query results, to be displayed by the result JSP. What is the most important step in the above? Using the testIt() method of the bean!

JavaScript tree

Here is an example of a javascript tree tag:



And here is the JSP that emits the javascript.

```
<davisjsp:tree level="-1" text="" script="true" browser="<%=browser
%>" />

<% String browser = request.getHeader("User-Agent");  %>

<davisjsp:tree level="0" text="Level 0.1" icon="book_close.gif"
leaf="false">

   <davisjsp:tree level="1" text="Level 0.1.1"
icon="folder_open.gif" leaf="false" style="WIN">

      <davisjsp:tree level="2" text="Level 0.1.1.1"
icon="file_selected.gif" leaf="true"  />

   </davisjsp:tree>

   <davisjsp:tree level="1" text="Level 0.1.2"
icon="folder_closed.gif" leaf="false">

      <davisjsp:tree level="2" text="Level 0.1.2.1"
icon="file_unselected.gif" leaf="true"  />

   </davisjsp:tree>

</davisjsp:tree>

<davisjsp:tree level="0" text="Level 0.2" icon="msie_page.gif"
leaf="true"  />
```

</davisjsp:tree>

Frequently, you want to display content hierarchy in a tree that expands in a browser with out a submit. You can also do a server side tree via the Struts menu.

The content hierarchy is usually stored in a SQL table via a self join.

Examples

As a part of basic portal there are the following working production examples that you should use when needing a similar need:

- CMS
- Ads *
- Portal
- Membership management
- Forums
- Mail
- Shopping Cart
- Credit Card Processing
- Survey
- Reporting
- Graping
- Security
- Blog

* For example, a great way to server out Ads is as CMS. You just need to display, track and report. This way there are no delays for external sites.

These are done using Struts and demonstrating best practices, with the main guiding goal of Simplicity. Because of Struts, these modules are easy to customize and maintain. The big point of basicPortal is to provide 80% of code common to 80% of projects. Since each project is different, students can then develop the last 20% that is custom and that deals with specialized requirements. Anything that is common would be found in bP as an example. Anything that is an exception or specific to a few cases would be up to you.

BasicPortal has integrated IDE, Aplication Server, SQL Ending and other things so that you can unzip and hit the ground running. Care has been taken to only utilize accepted best practices.

Inversion of Control Pattern (IoC)

There are several IoC service microkernals out there, such as Nano and HiveMind. IoC is a design pattern that is also sometimes called the Hollywood Pattern ("Don't call us we'll call you") . Simply put, an IoC component does not go off and get other components that it needs on instantiation. Instead ts boss supplies them. Hivemind can be obtained at SourceForge (sf.net). To use HiveMind, one begins by declaring and interface, such as this:

```
package bPe.async1.charge;
public interface Charge {
    public boolean send();
}
```

And to make it work in HiveMind, you then map the interface, with a file that must be in src/META-INF/hivemodule.xml:

```
<?xml version="1.0"?>
<module id="asyncB" version="1.0.0">
<service-point id="charge" interface="bPe.async1.charge.Charge">
    <create-instance class ="bPe.async1.charge.ChargeImpl"/>
</service-point>
 <service-point id="news" interface="bPe.async1.news.NewsSync">
          <create-instance class ="bPe.async1.news.NewsSyncImpl"/>
    </service-point>
</module>
```

You can see how by changing the mapping file, you can change what the implementation will be.

You could then utilize the services like this:

```
public void exec() {
NewsSync news = (NewsSync)
_reg.getService("asyncB.news",NewsSync.class);
Charge commerce = (Charge)
_reg.getService("asyncB.charge",Charge.class);
commerce.send();
news.receive(); }
```

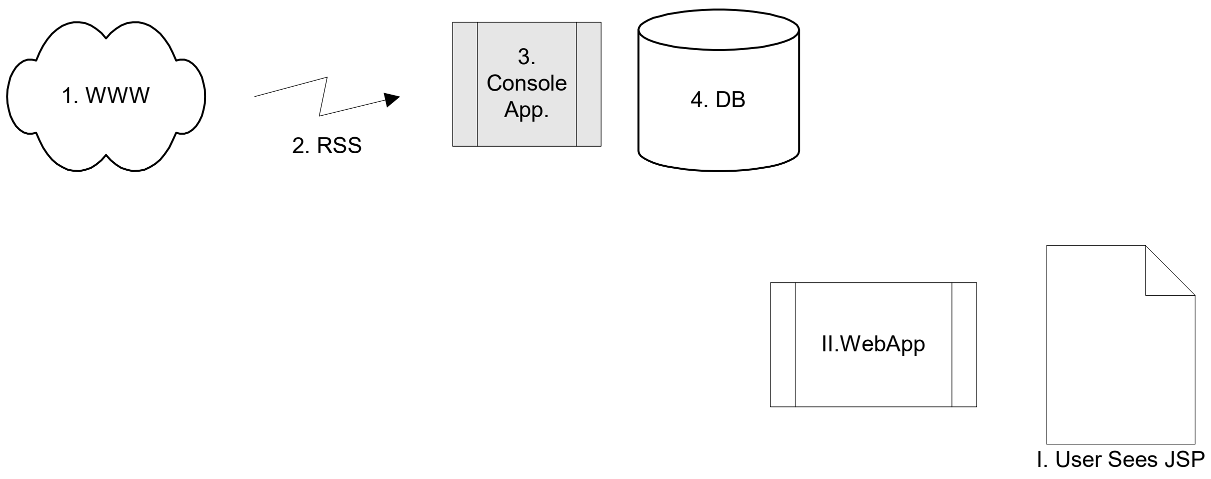
Asynchronous Processing

Any task that could potentially be time consuming should be done asynchronously so as to not to hold up the application server. Also any tasks that that involve talking to another data source, for example talking to RSS, Mail, credit cards services, etc.   All these tasks should be done as asynchronous console applications, outside the application server. It is a mistake to try to send Mail or talk to RSS from JSP.

Imagine a user joins a site and you want to confirm the e-mail. Above process can run as a cron loop job, checking the database for new users and sending out an e-mail. Even when we write console apps, we can keep using the same beans. We just change the SQL Config.xml like this:

```
<datasource name = “ids”
factory-
class=”com.ibatis.db.sqlmap.datasource.SimpleDataSourceFactory”
default = “true” >
   <property name=”JDBC.Driver”                    value=
“org.postgresql.Driver” />
   <property name=”JDBC.ConnectionURL”       value=
“jdbc:postgresql://basebeans.com:5432/bp2?autoReconnect=true” />
   <property name=”JDBC.Username”            value= “bpuser” />
   <property name=”JDBC.Password”            value= “password” />
</datasource>
```

Containers are best at running many quick tasks, such as JSP and servlets that take nanoseconds, not a task that may take a second or more. Maybe even the server is down.


1. WWW
2. RSS
3. Console App.
4. DB
II.WebApp
I. User Sees JSP

e-Commerce and XML

Assume that you wrote a JSP shopping cart that add items to an order and then saves it to a DB. Here is an example of an async. process:

```
public boolean send()  {
OrderBean orderBean = new OrderBean();
orderBean.populateAll(); // this will find all the orders that were not
charged/processed

 while (orderBean.hasNext()) {

    BigDecimal totalPay = orderBean.getTotalAmount();
    String ccNum= orderBean.getCCNum()
    String adres1 =orderBean.getUserBean().getAdres1();
    String postal =orderBean.getUserBean().getZipCode();

    Element orderEl = new Element("order");
    Element merchantEl = new Element("merchantinfo");
    merchantEl.addContent(new Element("configfile").addContent(store));
    orderEl.addContent(merchantEl);

    Element orderTypeEl = new Element("orderoptions");
    orderTypeEl.addContent(new Element("ordertype").addContent("SALE"));
    orderTypeEl.addContent(new Element("result").addContent("LIVE"));
    orderEl.addContent(orderTypeEl);

    Element payTotEl = new Element("payment");
    payTotEl.addContent(new
Element("chargetotal").addContent(totalPay.toString()));
    payTotEl.addContent(new Element("tax").addContent("0"));
    orderEl.addContent(payTotEl);

    Element ccEl = new Element("creditcard");
    ccEl.addContent(new Element("cardnumber").addContent(ccNum));
    ccEl.addContent(new Element("cardexpmonth").addContent(expMon));
    ccEl.addContent(new Element("cardexpyear").addContent(expYr));
    orderEl.addContent(ccEl);
    Element billAdrsEl = new Element("billing");
    billAdrsEl.addContent(new Element("addrnum").addContent(streetNum));
    billAdrsEl.addContent(new Element("zip").addContent(postal));
    billAdrsEl.addContent(new Element("name").addContent(fullName));
    billAdrsEl.addContent(new Element("address1").addContent(adres1));
    billAdrsEl.addContent(new Element("state").addContent(state));
    orderEl.addContent(billAdrsEl);

    Document order = new Document(orderEl);
     ChargeHelper cHelper = new ChargeHelper(); // here we use a helper
class that send the XML via internet
      Document result = cHelper.charge(order); // here we receve an XML of
the response, that is mostly the aproval code

    } // next
    }// send()
```

You can see how we make an XML Document for sending. Shopping card is an important way to build a relationship with your customer.

Generating Ad-hoc Reports

On most projects, reports are a major part of the requirements. We created some mock ups of reports earlier in iReport, now you can use JasperReports to emit the live reports for users. You could write a servlet that creates and returns a report, as follows:

```
public class ReportHandler extends HttpServlet{
   public void service(HttpServletRequest request,
                       HttpServletResponse response)
        throws IOException, ServletException      {
        String rep = req.getParameter("report");
        String args = req.getParameter("args");
        PrintWriter out = response.getWriter();
        if (rep == null)        {
              out.println("no report argument");
              out.close();
              return;
        }
        ReportHelper helper = new ReportHelper();
        String repXML = "/WEB-INF/reports/"+rep+".xml;
```

```
            helper.reportPrepare(repXML, this);
            java.util.Map argm = new HashMap();
            String repJasper = rep + ".jasper";
      helper.reportFill(repJasper, this, argm);
      String repOut = helper.reportEmitHTML(req);
      out.println(repOut);

  out.println(e.getMessage());
  } }
```

This servlet relies on a ReportHelper class that prepares, fills and returns a Jasper report. Note that we are using a helper, following our OO ideals. Jasper is a very popular open source report-generating package, available at http://www.sourceforge.net/projects/jasperreports. Jasper needs a bit of help to get it to work with iBatis DAO and Servlet engine, which is what helper does. The servlet receives the report name as argument, and optionally, further arguments that the ReportHelper may need. Any component that passes these arguments to the servlet will receive a report.

In *Jasic*, the report servlet shown above is registered in web.xml at /rep/handler. This means that a report can be returned to the browser with a URL similar to the following:

http://localhost/rep/handler?report=UsrReport1

[TODO: maybe add Screen shot of iReport designer here]

The previous section used a report form in XML format available at /WEB-INF/reports/UsrReport1.xml. Report forms for Jasper are standard XML and may be created with any text editor. However, you may consider using iReport, a powerful open source tool for creating report forms. It can set up reports in HTML, PDF, XML and other formats. It is a visual designer allows you to visually insert fields and map them to columns in tables of your database.

iReport is available for download at /http://www.sourceforge.net/projects/ireport.

Optional LAB AA : Dynamic Content

Select * from usr to find the password.
You can also select * from content, to view content.

2. Run the app and login as admin.
3. Go to content management.
4. Change the about page, add a few words and save.
5. Now click on the “about” menu item.

LAB AX Opt: Report Servlet

1. Place the XML report definition generated by iReport, in the 1st reports to WEB-INF/reports.

2. Use the report helper Servlet. Look at web.xml to find out more about the report servlet.

3. Start app server, and login in.

4.Navigate to: http://localhost/rep/handler?rep=name

   where name is the name of your XML file.

This should generate an html report.

Review docs on JasperReports.sf.net and edit xml file in a text editor in case of issues.

Chapter 14: Security

Login Security with Container-managed authorization

J2EE has a built-in security mechanism via container managed autorization. By using these mechanisms, you may well be able to leverage your own security.

This security allows to protect URLs from unauthenticated access. When a protected URL is accessed, it verifies if the user credentials permit access. It is capable of presenting a login screen or dialog if the user is not yet logged in. It also verifies if a logged in user has a role (such as ADMIN) that qualifies for access to the specific URL.

This type of security is set up declaratively in web.xml. The following is an example:

```
...
<security-constraint>

<web-resource-collection>

    <url-pattern>/do/userAdminAct/*</url-pattern>

    <url-pattern>/do/contentAdminAct/*</url-pattern>

</web-resource-collection>

<auth-constraint>

    <role-name>ADMIN</role-name>

</auth-constraint>

<user-data-constraint>

    <transport-guarantee>NONE</transport-guarantee>

</user-data-constraint>

</security-constraint>

<login-config>

<auth-method>FORM</auth-method>

<form-login-config>

    <form-login-page>/sLogin/login.jsp</form-login-page>

    <form-error-page>/sLogin/loginBad.jsp</form-error-page>

</form-login-config>

</login-config>

<security-role>
```

```
    <role-name>GUEST</role-name>
</security-role>
<security-role>
    <role-name>ADMIN</role-name>
</security-role>
</web-app>
```

To secure an application that uses Struts actions to access pages, the rights to each page should be set up on the action URLs that are called to access these pages. In the above example, the first security constraint ensures that only logged in users with ADMIN role are permitted to access the administration modules with userAdminAct and contentAdminAct. The second security constraint defines a members-only area of the site, where only logged in users, including guests and administrator can add news or comments.

The login-config section specifies that all unauthenticated requests to actions with a security constraint forward to an HTML form, which will typically have to provide the login form to enter username, password and a login submit button.

The security roles are also declared in WEB.XML as shown above. Of course, there are still some pieces missing for everything to work, as you will see in the next section.

Some developers try to build a home grown security system outside of the J2EE, this is a bad idea when you can extend container-managed authorization.

Configuring Container Security

While web.xml allows us to declare security roles, the actual repository for user and user rights information is maintained elsewhere, through mechanisms provided by the servlet container.

Depending on the business environment, one may wish to authenticate against a Novell NDS security realm, Microsoft security realm, or against a JDBC realm, for example. In a JDBC realm, users and their mapping to roles are stored in the database and accessed via JDBC. With a JDBC realm, your application can provide a web-based userAdmin module that allows you to add, modify and delete users without having to write code that accesses proprietary security systems.

Each container has its own configuration standard for security realms. This is an example of a JDBC realm from Tomcat server.xml:

```
<Realm className=”org.apache.catalina.realm.JDBCRealm” debug=”99”
driverName=”org.postgresql.Driver”
```

```
   connectionURL=”jdbc:postgresql://localhost:5432
                  /basicportal?autoReconnect=true”
connectionName=”bpuser” connectionPassword=”changeme”
       userTable=”usr”
     userNameCol=”real_email_id”
     userCredCol=”passwrd”
   userRoleTable=”usr”
     roleNameCol=”role_code” />

This is the same realm configured in Resin resin.conf:
<web-app id=” app-dir=’c:\jasic\Myagent\myapproot’ >
     <authenticator id=’com.caucho.server.http.JdbcAuthenticator’>
<db-pool>bppool</db-pool>
<password-query>
select passwrd from usr where real_email_id=?
</password-query>
<role-query>
select role_code from usr where real_email_id=?
</role-query>
        </authenticator>
...
</web-app>
```

When moving from testing to a production environment, be aware that the content of your user repository and user rights may need to be modified. You do not want a user with admin rights left over from the test environment mistakenly purge the production database after several days of operation.

Further, the application server and database server should not be run as the UNIX root user. The database account used by the web container should not be an administrative account and should not have write privileges if not needed.

Generally speaking, production servers should be secured by an experienced administrator using a hardening script such as Bastille. All servers should be protected from potentially disgruntled employees with physical security and user account policies.

Membership

Most applications have to develop membership management. You want to store the member information in the database, allow users to add themselves and have an admin screen where you can assign a role.

You could also have a cross reference table that would link many to many as to what content they like.

Or even FOAF (Fried of a Friend) cross reference many to many table that indicates who are the friends of the member are. This give you a way to drill down to FOAF content.

Implementing Row Level Security

Sometimes, certain application users should not see certain rows of tables. In an intranet portal application, a group of STANDARD users may view sales contact information, while GUEST users (such as external consultants) may not. The right to modify contact information may be limited to a group of ENHANCED users. In addition, possibly users should be limited to view the contacts within their unit or organization only, rather than the company-wide list of contacts.

The contact management example is actually a combination of three security mechanisms: read access by role, modify access by role, and access by organization (unit).

To implement this, for each security mechanism required, we add a column to the table of contacts, such as:

contact.roleViewID
contact.roleModID
contact.orgID

A roleViewID value of null can be interpreted as “no security” and all users may view it. A roleModID value of “ENHANCED” would mean that only users with this role may modify this item. An orgID value of “EAST” would mean that only users in the Eastern sales organization can view this item.

Another situation where you will want to limit access by organization is when you deliver your application as *Application Service Provider* (ASP). If you have three clients, A, B, and C, all of them should be able to enter records into your tables. However, when users from Client B retrieve records, they should not be able to see other clients’ (such as Client A or B) data. One approach is to have three different databases with three different connection pools (database users). However, if your number of clients increases, your approach is not scalable and will rapidly become a maintenance nightmare.

Adding Row Level Security to Bean and SQL Map

As usual, before you modify the mapped statement in the SQL Map, you will develop and test the SQL in a SQL console. Initially, a test select statement may look as follows:

SELECT * FROM contact WHERE (name_last LIKE "%e%") AND (roleViewID IS NULL AND roleModID IS NULL AND orgID IS NULL)

This test statement should retrieve all rows that meet our search criteria AND do not have a security setting.

As a second step, use your SQL console to add some values to entries in the usr table, as in:

usr:
id, real_email_id, passwrd, organization, role_code
1, salesrep1@basebeans.com, 111, SALES_US, "ENHANCED"
2, salesrep2@external.com, 222, SALES_US, "STANDARD"
3, salesrep3@infonoia.com, 333, SALES_EU, "ENHANCED"
4, salesrep4@external.com, 444, SALES_EU, "STANDARD"

As a third step, use your SQL console to add some values to entries in your contact table, as in:

```
name_last, name_first_init, roleViewID, roleModID, orgID

Cekvenich, V., STANDARD, ENHANCED, SALES_US
Gehner, W., STANDARD, ENHANCED, SALES_EU
```

Consequently, for the user salesrep2 the following SQL should be generated to view the items:

```
SELECT name_last, name_first_init, tel FROM contact C, usr U WHERE
U.real_email_id="salesrep2@external.com" AND

C.orgId = U.organization AND

(C.roleViewID IS NULL OR C.roleViewId IN (U.role_code))
```

The SQL map that is used to display a list of all items that match the security criteria would have a mapped statement as follows:

```
<mapped-statement name='ContactSecureList' result-map='result'
         inline-parameters='true'>
    SELECT name_last, name_first_init, tel FROM contact C, usr U
    WHERE  U.real_email_id=#value# AND

    C.orgId = U.organization AND
    (C.roleViewID IS NULL OR C.roleViewId IN (U.role_code))
</mapped-statement>
```

The content bean would pass the key for the logged in user to the mapped statement as follows:

```
public void selectSecure(Object key){
doRetrieve("ContactSecureList", (String) key);
}
```

The action that will show the list of contacts could have a method, as follows:

```
public String onDisplayExec(ActEvent ae) throws Exception{
    ContentBean b = (ContentBean) ae.getBean();
    Object userKey =
         ae.getRequest().getSession().getAttribute("user_key");
    b.selectSecure(userKey);

    return "Success";
}
```

This assumes that the lookup key for the user (here: real_email_id) was stored in the session during the login process.

Dealing with Inherited Rights

How can you ensure that ENHANCED role also includes the rights of the STANDARD role (such as view an item accessible to STANDARD users)?

One way to deal with this would be to allow users with several roles, maintained either in a comma-delimited list in usr.role_code, or have a separate table of user_roles that joins to the usr table. However, maintaining several roles per user can be cumbersome.

Secondly, you could adapt the SQL to *view* the items to include the rights critera as follows:

C.roleViewID IS NULL OR
(C.roleViewID="STANDARD" AND U.role_code IN (STANDARD,ADVANCED))
OR (C.roleViewID="ADVANCED" AND U.role_code ="ADVANCED")

However, you may wish to avoid coding security roles and rules in the db layer, while they are really a part of the application logic.

Therefore, thirdly, you could build a role string such as "STANDARD,ENHANCED" when the ENHANCED user logs in. This string could be maintained in the session, to be passed to the secure query along with the user key. A mapped statement for this could look as follows:

```
<mapped-statement name='ContactSecureList' result-map='result'
         inline-parameters='true'>
```

```
    SELECT name_last, name_first_init, tel FROM contact C, usr U
    WHERE  U.real_email_id=#value# AND
    C.orgId = U.organization AND
    (C.roleViewID IS NULL OR C.roleViewId IN (#role_string#))
</mapped-statement>
```

This assumes that the bean passes both user key and role_string to the mapped statement.

Tiles and Menu Security

Struts-menu provides special menu displayers that can read and interpret container-managed roles. Such a menu could be expressed as follows:

```
<% taglib prefix=”menu” uri=”menu” %>
...
<menu:useMenuDisplayer name=”MenuForm” permissions=”rolesAdapter”>
        <menu:displayMenu name=”mySpecialAdminMenu”/>
        <menu:displayMenu name=”myMenuForEverythingElse”/>
</menu:useMenuDisplayer>
...
```

This would work in conjunction with an entry in menu-config.xml that specifies the administration menu, as follows:

```
<Menu  name=”mySpecialAdminMenu” title=”Administration”
        roles=”ADMIN”>
        <Item title=”User Admin” location=”/do/userAdminAct” />
        <Item title=”Content Admin” location=”/do/contentAdminAct” />
</Menu>
```

The result is that the special administration menu is only shown to a logged in user with the role ADMIN.

Struts-menu can provide even more fine-grained control that allows to show or hide individual items of a menu. If you think you need this, feel free to consult the examples provided in *struts-menu.war*. MenuForm is a security menu type.

Tiles also uses the role attribute. You can optional display tiles if the user is in a certain role. This comes in very useful.

SSL

We should be able to encrypt our content as it goes over the wire, so that we can get to the web site like this

https://localhost

We can use the keytool that comes with JDK

```
Keytool –genkey –alias Tomcat –keyalg RSA
```

Then we have to setup the keystoreFile in the server, which is specific to every container. We would go to https mode anytime we issued a password or a credit card number. (One of the example modules is a shoping cart). Built into basicPortal is support for Struts SSLExt, that makes it easier to switch back and fort from http to htts.

Geocode User IP

Sometimes there are denial of service attacks that come from obscure places. We can protect our web application by writing a filter that will ignore all requests that come from blocked regions. One can acquire a database that codes what IP ranges comes from what location.

We can find the IP of the user going to the site, like this:

```
String ip = request.getRemoteAddr();
```

Based on this information, we can write a simple filter, that intercepts the requests, and does a query to find out the geo coded location, to see if you are allowed to get to this site, from your location. Sometimes you also want to capture the users IP, in case you want to block them from your site for any reason.

Other Security

Windows has a reputation as being hard to secure. With Linux, there are hardening packages such as Bastile. It is important not to run your container at root level. Also, in SQL, the creator of the tables should not be the same user that you connect at as a web app db user. So we have a SQL admin that creates a SQL table, and they grant some privileges to “bpuser”. Mostly the router (CISCO et al) is exposed to outside internet, and application servers are in DMZ, while the DB server is behind the fire wall.

Review

Web is full of opportunity for mischief, but we can do some diligence and harden our web site.

LAB AB : Security

1. With the application running, click join to add a user.

2. Modify web.xml to require a user to be a “GUEST” role to view tasks. Examine the urls at end of web.xml for syntax. You can just go to the guest security constraint.

In there add a url pattern for /do/tasksPg/*

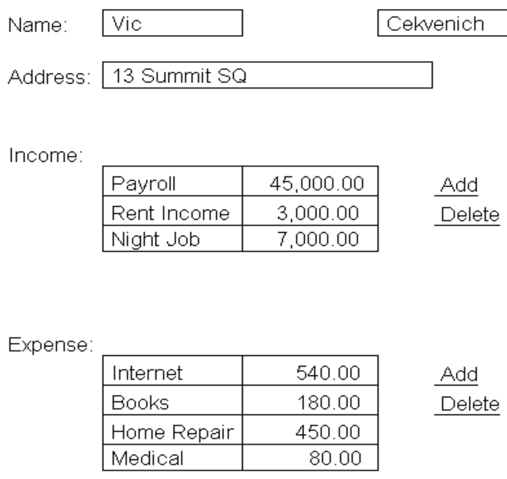
3. Restart web server, navigate to localhost/ (home page) and test to see if you can get to that page via menu, or another way.

4. Select * from usr to find the password, or add your own record to usr table.

Chapter 15: Complex Forms and Dot Notation

Complex Forms

Imagine a complex jsp form, that has multiple lists to display. For example a list of income and a list of expense on the same page. For example in a tax form. You can add expenses to exense list or income to income list.



Name: Vic Cekvenich

Address: 13 Summit SQ

Income:

Payroll	45,000.00
Rent Income	3,000.00
Night Job	7,000.00

Add
Delete

Expense:

Internet	540.00
Books	180.00
Home Repair	450.00
Medical	80.00

Add
Delete

Based on this requirement, how would you implement a formBean that matches it? You end up nesting beans within each other, and using a dot notation to access properties.

Within one form bean you can have “is a/has a” implementation like so:

```
public class CompoundBean extends BeanA {
private BeanB _b = null;
public BeanB getBeanB() {
return _b; }
}
```

Nested Beans

Consider a tax form as shown above. It has address information, a list of income and a list of expenses. One person has many incomes and expenses. Data for income and expense is likely to be stored in a different database table, in a many-to-one relationship to person data. The number of income/expense items will vary, and the form should allow editing of the lists.

At the JSP layer, you would uses Tiles to combine "sub"-JSPs in a "master"-JSP, in a similar fashion as shown above. But what about the form bean?

In this module, you really have three form beans, each drawing data from their own DAO. The beans are fairly independent. For example, when using the page, you could add many expenses without ever updating data on the person. This is a situation where you may consider nesting beans.

How would you construct such a complex page? Each sub-JSP has its own bean. First, you would unit test each sub-JSP with its bean. Then you would create a bean for the master-JSP on which you carry instances of the sub-beans as properties.

The compond-bean could nest beans as follows:

```
public class TaxBean extends BaseBean{
public PersonBean personBean = null;
    protected IncomeBean incomeBean = null;
    protected ExpenseBean expenseBean = null;
    public TaxBean()  {
          personBean = new PersonBean();
          incomeBean = new IncomeBean();
          ExpenseBean = new ExpenseBean();
    }
public PersonBean getPersonBean() //returns instance of bean    {
    return personBean;
}
public IncomeBean getIncomeBean() //returns instance of bean    {
    return incomeBean;
}
public void save();
    super.save()
    incomeBean.save();
    expenseBean.save();
}
public String testIt() {
    super.populate()
    incomeBean.populate ();
    expenseBean. populate ();
    super.setPropX()
    incomeBean.setPropY();
    expenseBean.setProX();
    super.save()
    incomeBean.save();
    expenseBean.save();
    return null; // the db should be updated, and bean maps the
form making it easy.
}
```

Note that in the compound bean, we have to override save() to ask support to save if any, and then any nested beans. The concept of nested beans following complex forms is an important one. Do you see how we can now unit test CRUD for the compound bean, before we try to integrate? Cool.

Rather than having the TaxBean just returning String properties, you return a bean or a collection. In an action, you could then access the properties of PersonBean with something like myTaxBean.getPersonBean().getNameLast();

Dot Notation

Properties of nested beans can be read from the JSP with what is called dot notation. This is the same mechanism we used in part 2 of the book in the iterate tag which used a property=”dao.resList”.

For example, a non-nested property would be addressed with the following:

<c:out value=”${myBeanName.myPropName}”/>

A nested property could be addressed, as follows:

<c:out value=”${myBeanName.**myNestedBeanName**.myPropName}”/>

In the example above, we could write, as follows:

<c:out value=”${taxBean.personBean.name_last}”/>

This assumes that:

(a) the action has put TaxBean in scope using the key taxBean,
(b) TaxBean has a public property getPersonBean() which returns PersonBean, and
(c) PersonBean has a public property getName_last()

Many tag libraries, such as JSTL, support nested properties.

Nested Tag

While JSTL tags support nested properties, the Struts html tag library tags for input fields do not support these nested properties. This creates a potential issue where your form bean is nested inside a master bean and/or you have several form beans.

A Struts extension tag library nested: exists where you could write as follows:

```
<html:form action="/mySub1Act"/>
    <nested:nest property="mySub1Bean" >
          <nested:text property="mySub1Prop"/>
          <nested:submit property="Dispatch" value="Save"/>
    </nested:nest>
</html:form>

<html:form action="/mySub2Act"/>
    <nested:nest property="mySub2Bean" >
          <nested:text property="mySub2Prop"/>
          <nested:submit property="Dispatch" value="Save"/>
    </nested:nest>
</html:form>
```

This assumes that the master bean has been put in scope and has the methods getMySub1Bean() and getMySub2Bean() which return the instances of the respective beans.

Does using the nested tags create an issue if you want to use a JSP that is not nested? That is where mySub2Bean is put in scope rather than the master bean? You could maintain the nested syntax if you add a method to mySub1Bean as follows:

```
public class MySub1Bean extends BaseBean
{
      ...
   public MySub1Bean getMySub1Bean() //getter to return itself
      {
   return this;
      }
}
```

This would ensure that the nested tag can operate on the bean itself rather than looking for a nested bean.

The way you know you have mastered Struts is if you can do above nested module from scratch, in less than half a day, or a few of these type modules per day without strain.

Review

You should know how to create compound beans that contain nested beans and how to unit test crud on the compound beans.

Optional LAB AC: Nested Dot Notation

1. Create a project module:

 Create a SQL table to store project info. (projects have tasks)

 Create a DAO, Bean, JSP to list projects.
 Unit test the Project Bean.

2. Alter table tasks to have a FK proj_id

3. Clicking on a project on list should list tasks for that project
 Change task list to receive a message of what proj_id to display.
 Change the task list populate, to populate based on proj_id, only tasks for that project.
 Change the task onNew, to find the FK (proj_id) and set it before task page displays.

4. Now we have 2 tested core (simple) beans, and 2 modules (tasks and projects)

Create a nested (compound/complex) bean called ProjTaskFBean that extends TasksBean. It is a TaskBean.

In ProjTaskFBean add a property of Project Bean. ProjTaskFBean now "has a" Project Bean. Ex:

```
private ProjectBean _pB= null;

public ProjTaskFBean() { //cons
_pB = new ProejctBean();
  }

public ProjectBean getProject() {

    return _pb;

  }
```

Modify all the CRUD bean methods to re-throw to property. Ex:

```
public void save() throws Exception {
super.save();
_pB.save();
}

public void populate() throws Exception {
super.populate();
_pB.populate();
}
```

.etc.
Make sure you carry the FK at all times.

5. Unit test the nested (compound/complex) bean.

Note: Most forms are nested (compound/complex). Since FormBean needs to map to the From, we often need to create nested (compound/complex) beans.

Chapter 16: Dropdowns and Super Type

Using Option Collections for Drop downs

Drop downs or HTML select tags are an important element of web applications. They should be used in place of text input fields wherever possible. They make it difficult and sometimes impossible to enter invalid input, and thus provide a better user experience.

Often drop downs need to be dynamically populated from the database. Struts provides the html:optionsCollection tag to facilitate this. A simple JSP that uses an options collection to populate a drop down could look as follows:

```
<%@ taglib uri="struts/html" prefix="html" %>
<%@ taglib uri="jstl/c" prefix="c" %>
...
<html:form action="userAdminAct">
<table><tr>
  <td>ID: </td>
  <td><c:out value="userAdminBean.id"/></td>
</tr><tr>
  <td>Last name: </td>
  <td><html:text property="name_last" size="70"/></td>
</tr><tr>
  <td>First initial: </td>
  <td><html:text property="name_first_init" size="20"/></td>
</tr><tr>
  <td>Role: </td>
  <td>

<html:select property="role">

<html:optionsCollection property="roleOptions"/>

</html:select>

</td>

</tr>
</table>
</html:form>
...
```

This would obtain the collection of available roles from the form bean in scope. The form bean could look as follows:

```
public class UserBean extends BaseBean {


    protected String role = null;
     //inner class, but would usually be in a separate class
     class OptionBean //this encapsulates one option label and value
     {
           private String label = null;
           private String value = null;


           public OptionBean(String label, String value)   {
                 this.label = label;
                 this.value = value;
           }


     }


     ...
     public void setRole(String role)    {
           this.role = role;
     }
     public String getRole()       {
           return role;
     }
     public Collection getRoleOptions()  {
           List options = new ArrayList(); //implements collection
           options.add(new OptionBean("Sales", "101"));
           options.add(new OptionBean("Marketing", "202"));
           options.add(new OptionBean("Accounting", "303"));
           return options;
     }
}
```

See how that there is get and set for the current property? And a getCollection to give you a set of choices? This matches the tat select property and options collection.

By default, the optionsCollection tag tries to obtain the display value by accessing the label property of the option bean, and the storage value by accessing the value property of the option bean. Struts will store the selected value in the role property of the form bean, as specified in the surrounding html:select tag.

Before you move to obtaining the values from the database rather than hard coding them on the bean as above, you may wish to add a frequently required feature: a null value at the top of the list. This could look as follows:

```
<html:select property=”role”>

<html:option label=”---Please select one---” value=”-1”/>

<html:optionsCollection property=”roleOptions”/>

</html:select>
```

If the user does not select one of the items in the options collection, the role property of the bean will have the value “-1”. In the Save action, you could validate the role property and throw an error message if the value is outside the valid range.

Obtaining Option Values from the Database

Let us assume that the database has a role lookup table with the columns id and role_name. You could create an SQL-map roleLookupSQL and a DAO named RoleLookupDAO to access this information.

The sql-map could look as follows:

```
<sql-map name=’roleLookupSQL’>
<result-map name=’lookup_result’ class=’com.myorg.web.OptionBean’>
    <property name=label’ column=ROLE_NAME />
    <property name=’value’ column=’ID’ />
</result-map>
<mapped-statement name=’RoleLookup’ result-map=’lookup_result’
          inline-parameters=’true’>
    SELECT label, value FROM ROLE
          ORDER BY label ASC
</mapped-statement>
```

Note how the result map creates an instance of OptionBean for each row returned in the result list, rather than a HashMap as previously shown. If you already have a roleSQL.xml with other mapped statements and result maps, you could just add this mapped statement and result map to it.

Super Type Implementation

Inexperienced database model designers sometimes represent diffent types of option data in different tables, e.g. one table to list countries, one table to list status codes, one table to list roles, etc. Experienced designers create tables with “type types”, or super types.

Super types have a type-value-label structure. A typical super_type table would thus have four columns: id, type, value, and *label*.

The following are examples of rows:

1, country, US, United States
2, country, CH, Switzerland
3, status, open, Open
4, status, closed, Closed
5, role, 101, Sales
6, role, 202, Marketing

Using this approach, one generic table can be used to store and decode options and any other custom-typed data desired. The single table is easier to maintain than a design with one table for each custom type.

SQL-map and DAO for Super Types

The following is an example of an SQL-map that accesses data from a super_type table:

```
<sql-map name=”OptionsSelSQL”>

<cache-model name=”options_sel_cache” reference-type=”WEAK”>

<flush-interval minutes=”88”/>

</cache-model>

<result-map name=”result” class=”com.myorg.web.OptionBean”>

<property name=”value” column=”VALUE”/>

<property name=”label” column=”LABEL”/>

</result-map>

<mapped-statement name=”OptionsByType” result-map=”result” cache-
model=”options_sel_cache” inline-parameters=”true”>

SELECT value, label

FROM super_type

where type = #value#

</mapped-statement>

</sql-map>
```

A generic OptionsSelDAO may use the sql-map in a method as follows:

```
public class OptionsSelDAO extends DAOIBaseHelper {
populateByType(Object sType)      {
doRetrieve(“OptionsByType”, (String) sType);
    }
    ...
}
The DAO would be accessed from a helper bean as follows:

public class OptionsSelBean …{
…
public Collection getOptions(String type){
myOptionsSelDAO.populateByType(string);
return myOptionsSelDAO.getRowList();
}
```

The bean that should serve the roleOptions list of option beans would have a getter method as follows:

```
public Collection getRoleOptions() {
    if (_options == null) _options = new OptionsSelBean();
    return _options.getOptions("ROLE");
  }
```

Options choices are cached at the model layer in Struts. Also note that options collection never joins in SQL to the decoding table, thus the SQL join executes faster, as we will learn later. Benefits of Options Select are:

- Less validation
- Less typing
- Higher performance (no SQL join, browser decodes)
- Data driven Site (adding choices in DB, and the flow of the application changes)

Single maintenance screen

Review

With this chapter, you have created an infrastructure for creating efficient access to data for drop downs.

* (Required) LAB AD : Options Select

1. First make sure task status field is updatable without options selection. (You might need to alter table in case you do not have task status field.)

2. In task jsp, make task_status a option selection. Ex:

```
<html:select property=”taskStatus” size=”1”  >
        <html:optionsCollection property=”taskStatusOptions” />
</html:select> </td>
```

3. Add sample data using DB Tool to the super type table

Type filed should be “TASK_STATUS”

Ex: 1, Open, 2, Closed, 3, Approved

4. Add a method to the task bean:

```
public Collection getTaskStatusOptions() {
if (_options == null) _options = new OptionsSelBean();
return options.getOptions(“TASK_STATUS”);

}
```

Unit test the UserBean getTaskStatusOptions() in test Servlet

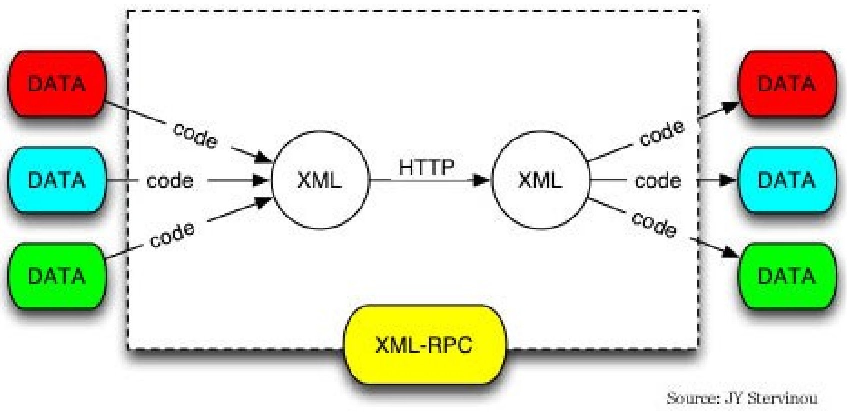
See if the JSP lets you update status of task via a drop down.
After testing, add a new choice via db, and see if it shows up.
Optional: Review the options sql.xml.

Chapter 17: Rich UI

XML-RPC

Every few years, a new approach or technique is developed that enables leaps in productivity. Such as was the case with Object Oriented capability. Recently, Web Services (WS) have allowed developers to produce more high quality work with less resources.

The underlying concept of WS is that with one simple protocol, multiple remote services can be easily connected and integrated even across company and firewall boundaries.



It is simple (built like HTTP servlets that do request-response in XML).

You can do browser-side XML-RPC and activate an XML RPC implementation in your project like this:

<script type="text/javascript" src="./scripts/init.js"></script>
Server-side, once you have working formbeans, you can expose them to XML-RPC.

Serving Content as XML

To transmit data in XML format, you do not necessarily have to wrap the XML data in an envelope that corresponds to the SOAP standard. You can also just write a servlet that emits XML rather than HTML.

This is what such a servlet could look like:

```
public void service(HttpServletRequest req, HttpServletResponse
res)
    throws IOException, ServletException{
ContentBean b = new ContentBean();
b.populateAll();
Document docOut = c.getXDoc();
XMLSerializer ser = new XMLSerializer(
res.getOutputStream(),
new OutputFormat(“xml”, “UTF-8”, true));
ser.serialize(docOut);
res.getOutputStream().close();
}
```

To make it real XML RPC we can use an xml.apache.org package like this:

```
XmlRpcServer xmlrpc = new XmlRpcServer ();
xmlrpc.addHandler ("examples", new ExampleHandler ());
...
byte[] result = xmlrpc.execute (request.getInputStream ());
response.setContentType ("text/xml");
response.setContentLength (result.length());
OutputStream out = response.getOutputStream();
out.write (result);
out.flush ();
```

You can access this from a JavaScript in a browser

```
function xmlhttp() {
xh = new XMLHttpfRequest();
xh.open("POST", "test.txt", true);
xh.onreadystatechange = function() {
        if (xh.readSate==4) { alert(xh.responseText()) }}
xh.send(null);
xh.setRequestHeader("MessageType", "CALL");
xml.send("some html");
}
```

It allows you to send and receive messages to servlets without a submit. You should start the advanced stuff by using an xml-rpc JavaScript library.

Establishing a xml rpc in javascript can be as simple as:

```
var server = new xmlrpc.ServerProxy(serverURL, methods);
or
service = XMLRPC.getService("xml rpcs service url");
```

depending on what .js library you use. More on XML RPC home page.

Case: Accessing Struts Beans from Excel

A client had an application for financial loan processing. As a successful application in production, the number of users and the scope of the application continued to increase. Some of the users had data on their PCs and laptops in Excel spreadsheets that should be captured in the application. Inversely, users should also be able to capture application information in their Excel spreadsheets.

Before the advent of SOA, one may have considered writing a custom client software that provides Excel-like functionality, a task of several man years. With SOAP, suddenly everything became much simpler.

With the form beans already tested and working, a SOA layer was added to the Struts application. For this, the Apache Axis SOAP WAR file was added, and a SOAP service created which obtained its data from the form beans.

The SOA web service was called from a PocketSoap module on each PC and laptop. PocketSoap was chosen over MS Soap because it did not require administrator privileges on the PC and laptops to install.

Excel used its macro language Visual Basic for Applications (VBA) to talk to PocketSoap.

The outcome was that when a user “loads” a spreadsheet, via SOA the getters of the form bean would be fired. “Saving” a spreadsheet meant that the setters of the form bean would be fired.

The users could take their laptop with them, work remotely, and enter their data and do their analysis as normal. Once they had Internet connectivity, they could send their data to the application.

Of course, the application continued to be accessible in parallel via the Web and JSP, and a browser interface. However, the Excel user interface is much richer and provides for a more comfortable user experience. An added benefit is that processing is offloaded from the server to the user PC.

With Axis, any Struts application can be made to provide SOAP web services very quickly. I now tend to look at light weigh WS such as XML-RPC over SOAP.

Flex

An interesting tag library from MacroMedia Flash is Flex, for example:

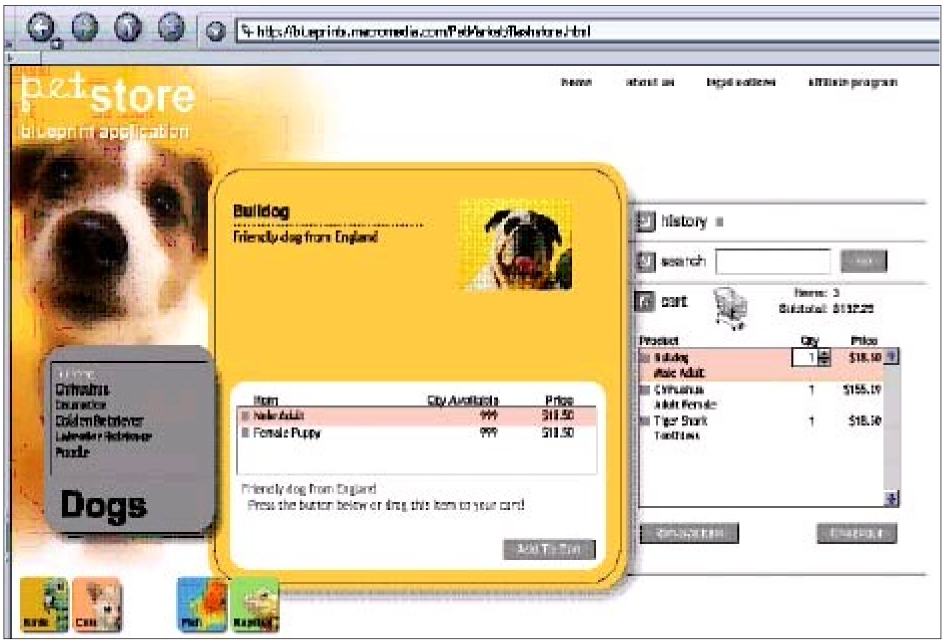
```
<%@ taglib uri="FlexTagLib" prefix="mm" %>
<html>
<%
 // In a real-life app, you get the XML data by invoking a method o
n a JavaBean
    String data="<employee name='John Smith' email='jsmith@mail.com
'/><employee name='Jane Doe' email='jdoe@mail.com'/>";
%>
This is html<br>
<mm:mxml>
    <mx:Application width="300" height="200" xmlns:mx="http://www.m
acromedia.com/2003/mxml">

        <mx:Model id="myModel">
            <%= data %>
        </mx:Model>

        <mx:DataGrid widthFlex="1" heightFlex="1" dataProvider="{my
Model.employee}"/>

    </mx:Application>

</mm:mxml>
</html>
```

This renders a Flash grid on a users browser, and anyone that does JSP can use it.

Flash

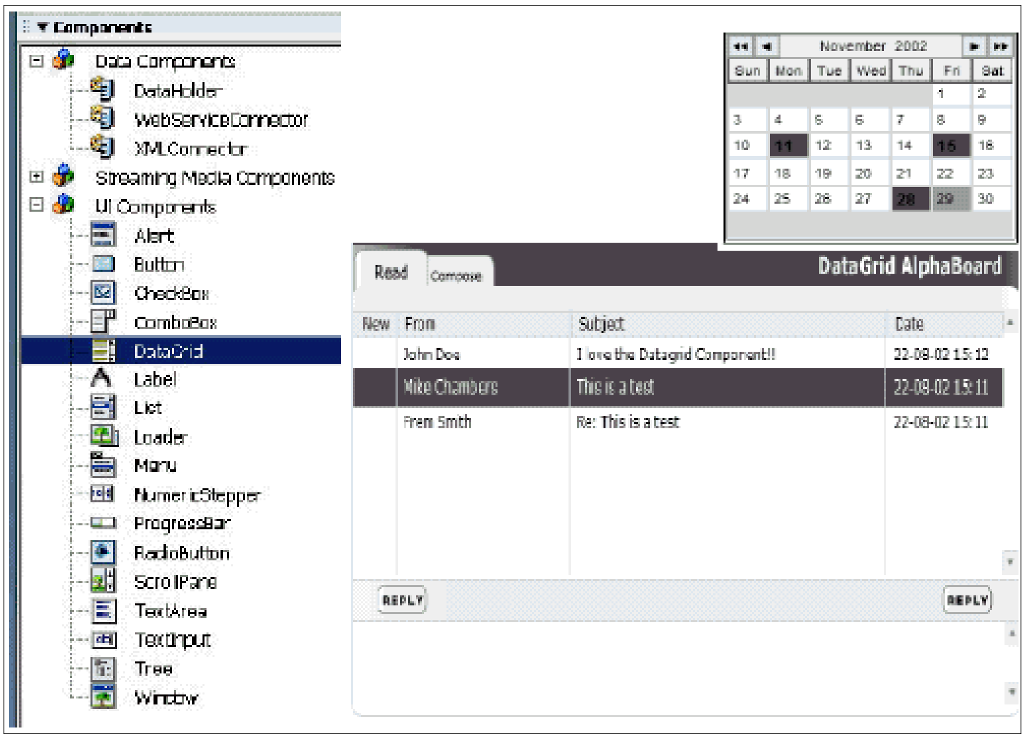
Flash component in the browser can provide a rich, data-bound user interface that draws its data from a server in XML format. Going much beyond the splash screens everyone has become accustomed to, Flash can provide data handling features that are quickly nearing those of heavy client applications (such as javascript client applications).

By executing the application on the client, you are likely to achieve performance improvements, a better user experience, and higher developer productivity when using existing server-side application infrastructure with remote access through web services.



It follows the open SWF (pronounced “swiff”) file format specification used by action script. Flash graphics are vector based and look decent on displays with different resolutions. A Flash application reacts almost instantaneously after the files have been downloaded.

The SWF specification is publicly available. You can find information about SWF at http://www.openswf.org. SWF files can run on Linux, MacOS, Windows, PocketPC and some Palm and phone devices. By coding to flash, we do not need to worry about cross-browser issues.



November 2002

Sun	Mon	Tue	Wed	Thu	Fri	Sat
					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30

New	From	Subject	Date
	John Doe	I love the Datagrid Component!!	22-08-02 15:12
	Mike Chambers	This is a test	22-08-02 15:11
	Frem Smth	Re: This is a test	22-08-02 15:11

Figure 17-1: Rich UI Components

The figure above illustrates some sample components that can be created with Flash/SWF. The editor allows to create bindings between a connection and data components using drag-and-drop. We would just place a component on “stage” and bind it to a WS. Importantly, you do not need to pay for a runtime or production license for your Flash application.

One of the most interesting features of Flash is its ability to deal with XML. Form bean values can be converted to their XML representation and rendered with Flash components. Flash documentation has sometimes implied that Java application servers cannot output XML that is suitable for Flash; this is, of course, entirely untrue.

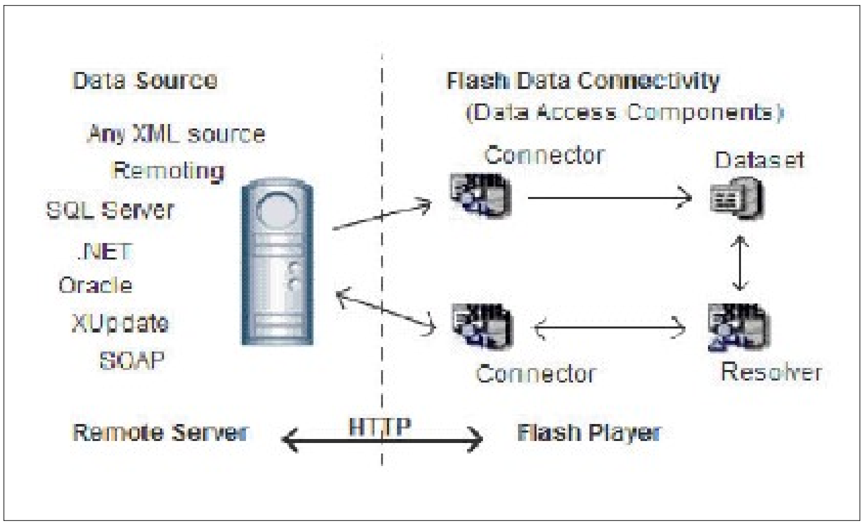


Figure 17-2: Web Service Architecture Using Flash

This schema above illustrates how the Data Access Components in a Flash application communicate with a remote server using XML via HTTP.

Creating Components

You can subclass and create your own visual components that developers can place on stage in Actionscript 2.0:

Import mx.core.UIObject;

```
Class myPackage.MyCommonent extends UIObject {
static var symbolName:String = "MyComponent";
static var symbolOwner:Object = Object(myPackage.MyComponent);
# include "../core/ComponentVersion.as"

    function MyComponent() {}
function init(Void):Void { super.init(); }
function size(Void):Voide { suprt.size(); }
}
```

Blob upload

Images and such are just another form of content. As such it is best managed for large sites inside of a database. One would have a Servlet that would allow us to emit this content in the appropriate section of a JSP, and a way to upload a blob to the db. Commons-upload should help you upload.

To display, something like this:

```
public void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException
    {


        File f = new File(System.getProperty("user.home")+"\\
zoewrap.jpg");
        JPEGImageDecoder decoder = JPEGCodec.createJPEGDecoder(new
FileInputStream(f));
        BufferedImage image =decoder.decodeAsBufferedImage() ;


        // Send back image
        ServletOutputStream sos = response.getOutputStream();
        JPEGImageEncoder encoder =
JPEGCodec.createJPEGEncoder(sos);
        encoder.encode(image);
    }


and
<img src="/servlet/ImageServlet?id=n&rnd=some_random_number"/>
```

Software as Utility

Distributed processing has evolved in waves, oscillating back and forth between client and server. The 60s and 70s had host-based processing on 3270s and VT100 terminals. The 80s brought processing to the client, with Client-Server, such as PowerBuilder running on a Windows PC and Sybase or Oracle Database running on the Server. The 90s brought back host-based processing, with HTML emitted by application servers and minimal processing on the browser. Software is now being partitioned back to the client server model. Application servers can provide WS, and scripts execute in a browser.

The big jump in productivity is that sometimes we do not have even to provide a application server to provide the Web Service for our browser scripts. We can just use 3rd party Web Services provided, such as those provided by Google, Amazon and Ebay. Since we do not need to write that part, the cost of development goes down.

ToDO:

JDNC

Review

MVC applications allows us to separate the presentation layer and the data layer. We can use this Client/Server approach to develop web services or to consume the services and XML from within our browser. By writing scripts, especially actionscript, we can have rich UI. Flash brings in a lot of useful visual components that we can leverage for development, with no run time costs.

Above all, make good looking applications.

LAB AF Opt: Flash 2003

1. Download and install eval version of Flash Pro 2004
2. http://www.macromedia.com/devnet/mx/flash/data_integration/presentation
3. Place a data grid component on Stage.
4. Connect data grid to the sample DB.
5. Retrieve data

Chapter 18: Managed Performance

Operation Monitoring

3 tiered applications in some circles means presentation, data and management. Monitoring an application in production is very important. We can use various logging implements to track production performance of the application.

You monitor the application performance so that you can anticipate upcoming problems and identify slow modules that you can focus on.

One package is JavaMon to monitor production applications. The following is an example of the execute(...) method in the BaseAct class, with monitoring:

```
public final ActionForward execute(...

{

Monitor monDat = MonitorFactory.start("ActionMon.hits" +
getDate());

Monitor monIp = MonitorFactory.start("ActMon.hits" + getIp(req));

ActEvent ae = new ActEvent();

ae.setReq(req);

ae.setBean((SimpleBean) formBean);

ae.setMapping(mapping);

processActScope(ae);

          ...

String parm = ae.getDispatch();

String forward = dispatch(ae);

          Monitor monForward = MonitorFactory.start("ActMon.hits" +
                                    forward + "." + parm);

          if(forward == null)

                forward = "Success";

          monDat.stop();

          monIp.stop();

          monForward.stop();
```

```
            return mapping.findForward(forward);
  }
```

This will generate report data that may look similar to the following:

```
  [TODO: JAMon report screen shot here]
```

Loading Data

Stress testing reduces costs because it allows you to finding problems earlier is much cheaper than finding problems after the application has gone to production.

When testing, the database should have more than just a few rows of data, since that would … not test much, everything would be in the db cache. How many records should you have in your db tables? Well, if you plan to have 30 million records in your databases after 5 years of operation, that is how many records you should have.

DBMonster can be used to generate sample records and insert them into your database as row. The database's memory cache needs to be exhausted in these tests.

A DBMonster configuration file dbmonster-config.xml may look similar to the following:

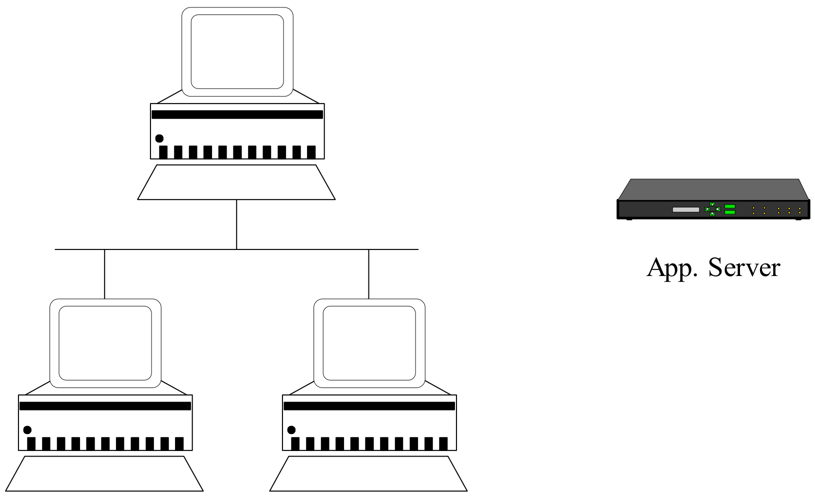
```
<!DOCTYPE dbmonster-schema PUBLIC
„-//kernelpanic.pl//DBMonster Database Schema DTD 1.0//EN‟
„http://dbmonster.kernelpanic.pl/dtd/dbmonster-schema-1.0.dtd‟>
<!-- Schema generated by SchemaGrabber -->
<dbmonster-schema>

<table name=”content” rows=”1000000”>
<key databaseDefault=”true”>
    <generator
       type=”pl.kernelpanic.dbmonster.generator.MaxKeyGenerator”>
          <property name=”columnName” value=”id”/>
    </generator>
</key>
<column name=”title” type=”varchar” nulls=”0”>
<generator
type=”pl.kernelpanic.dbmonster.generator.StringGenerator”>
<property name=”minLength” value=”5”/>
<property name=”maxLength” value=”70”/>
<property name=”allowSpaces” value=”true”/>
```

```
</generator>
</column>
<column name=”content_htm” type=”varchar” nulls=”0”>
<generator
type=”pl.kernelpanic.dbmonster.generator.StringGenerator”>
<property name=”minLength” value=”80”/>
<property name=”maxLength” value=”900”/>
<property name=”allowSpaces” value=”true”/>
</generator>
</column>
```

The program is compiled and executed with a command similar to the following:

```
java -cp .;dbmonster.jar;pg74jdbc3.jar;commons-cli.jar;
log4j.jar;commons-beanutils.jar;commons-logging.jar;
commons-collections.jar;commons-digester.jar;
pl.kernelpanic.dbmonster.DBMonster -c dbmonster-config.xml -s
dbmonster-schema.xml
```

Stress Testing



When testing the application server it important to consider the network IO. For example NewISys 2100 has dual gigabit NICs. Most PC and Laptops ship with a single gigabit card, but almost all server have dual. The first thing that gets saturated on the stress test is sometimes the NIC. Using a regular 100 megabit card on an application server does not make sense. It will be the first thing to saturate. Also, to do a stress test, you need several workstations to generate traffic against the application server.

A useful tool to conduct module load testing is OpenSta. Refer to the OpenSta lab in this book to find out more about OpenSta.

Database Server

A database server should have a maximum amount of memory and a maximum IO including SCSI R/W Cache possible. The more RAM the better, the more R/W cached I/O channels the better. You can review NewISys 2100 as a SCSI R/W cache machine and support for a lot of memory in a 1U size, configured to about $4,000US.

To configure db for memory, for example a Linux server with 8 gigabytes of RAM (), one should allocate all but 100 megabytes to the database, assuming the application server is located on another machine. The remaining 100 megabytes are used by the operating system.

The file /data/postgresql.conf on a Linux machine with 8 gigabytes should include a parameter configured similar to the following:

```
shared-buffers = 987500
```

In PostgreSQL, each buffer is 8K in size. By default, PostgreSQL is configured to use only 512K of RAM.

In Linux, the kernel.shmmax in sysctl.conf should also be increased as follows:

```
kernel.shmax = 7900000
```

This will tell the operating system to allow a process to use more RAM.

Application Server

Database servers often have the maximum memory possible with current technology. On the other hand, application servers usually do not need more than two gigabytes of RAM. Application servers should be configured, like database servers, to make use of all available memory except for 100 megabytes.

As an example, Resin would be started on a two gigabyte UNIX machine, using the Java flag as follows:

```
httpd -J-Xmx1900m
```

You obtain a help list of available Java flag by typing java -X in a console.

A system can have a lot of application servers, each with a DAO cache. However, there is usually only one database server, because replication across multiple database machines slows everything down, unless manual vertical or horizontal partitioning is used. If resources are constrained, always allocate more resources to the database server (and pay attention to I/O channels and R/W cache). In production with multiple application servers, one can configure most DAO's, including iBatis to flush all the caches for critical tasks.

Co-Location

A good idea is to store your servers at an unmanaged co-location facility. How to you calculate the required bandwidth? T1 allows for about 5.250 KB/s. Therefore, if you commonly use pages with images are 40 kB, that means that you can serve out about 130 pages per second.

JDBC DataSource

The DataSource pool has to work in sync. with garbage collection. Let say that you did not configure parallel garbage collection for your JVM properly. On a large system, a garbage collection could take 20 seconds.

During that time, a DataSource would time out in the middle of a retrieve, causing intermittent exceptions. A timeout setting for a datasource pool should be over a minute, just in case, and you should configure the JVM to garbagecollect in parallel.

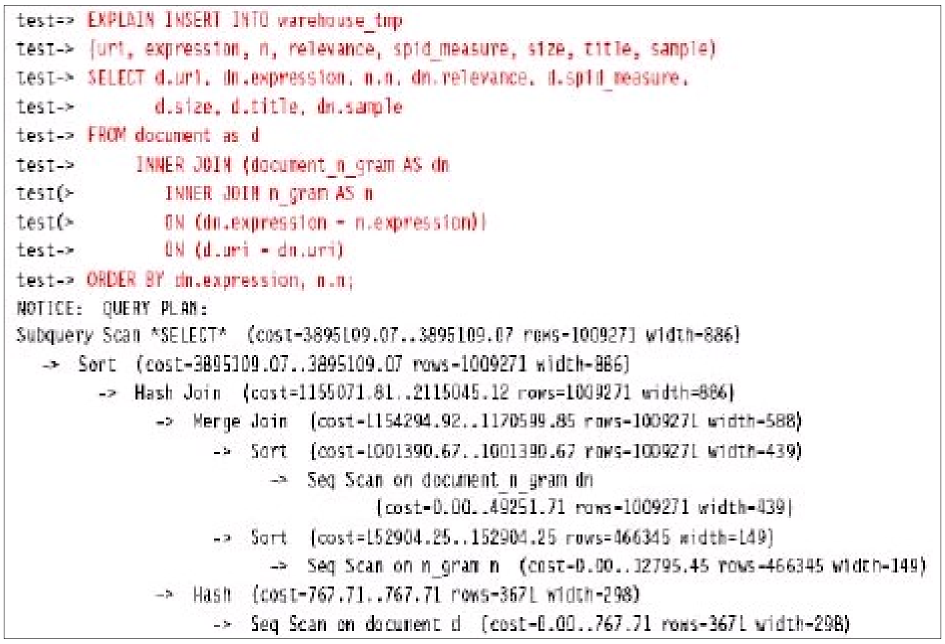
SQL Query Execution Process

In most J2EE applications, the greatest performance gain can be obtained by data design. Something that a lot of Java developers are weak on is SQL beyond the basics.. Maybe this is because even advanced Java developers are sometimes not very experienced in SQL, despite its basis in simple set unions and intersections.

Most SQL engines take the following steps when executing a query to find data:

1. Parse the query
2. Generate paths
3. Review the statistical distribution
4. Select a path
5. Execute (for each)

You can use the EXPLAIN SQL command to see how the data nodes are processed. The following is an example of an EXPLAIN command used with PostgreSQL:



```
test=> EXPLAIN INSERT INTO warehouse_tmp
test-> (uri, expression, n, relevance, spid_measure, size, title, sample)
test-> SELECT d.uri, dn.expression, n.n, dn.relevance, d.spid_measure,
test->        d.size, d.title, dn.sample
test-> FROM document as d
test->      INNER JOIN (document_n_gram AS dn
test(>         INNER JOIN n_gram AS n
test(>         ON (dn.expression = n.expression))
test->         ON (d.uri = dn.uri)
test-> ORDER BY dn.expression, n.n;
NOTICE:  QUERY PLAN:
Subquery Scan *SELECT*  (cost=3895109.07..3895109.07 rows=1009271 width=886)
  ->  Sort  (cost=3895109.07..3895109.07 rows=1009271 width=886)
        ->  Hash Join  (cost=1155071.81..2115045.12 rows=1009271 width=886)
              ->  Merge Join  (cost=1154294.92..1170599.85 rows=1009271 width=588)
                    ->  Sort  (cost=1001390.67..1001390.67 rows=1009271 width=439)
                          ->  Seq Scan on document_n_gram dn
                                    (cost=0.00..49251.71 rows=1009271 width=439)
                    ->  Sort  (cost=152904.25..152904.25 rows=466345 width=149)
                          ->  Seq Scan on n_gram n  (cost=0.00..32795.45 rows=466345 width=149)
              ->  Hash  (cost=767.71..767.71 rows=3671 width=298)
                    ->  Seq Scan on document d  (cost=0.00..767.71 rows=3671 width=298)
```

You can address efficiency issues by analyzing the paths selected by slow queries.

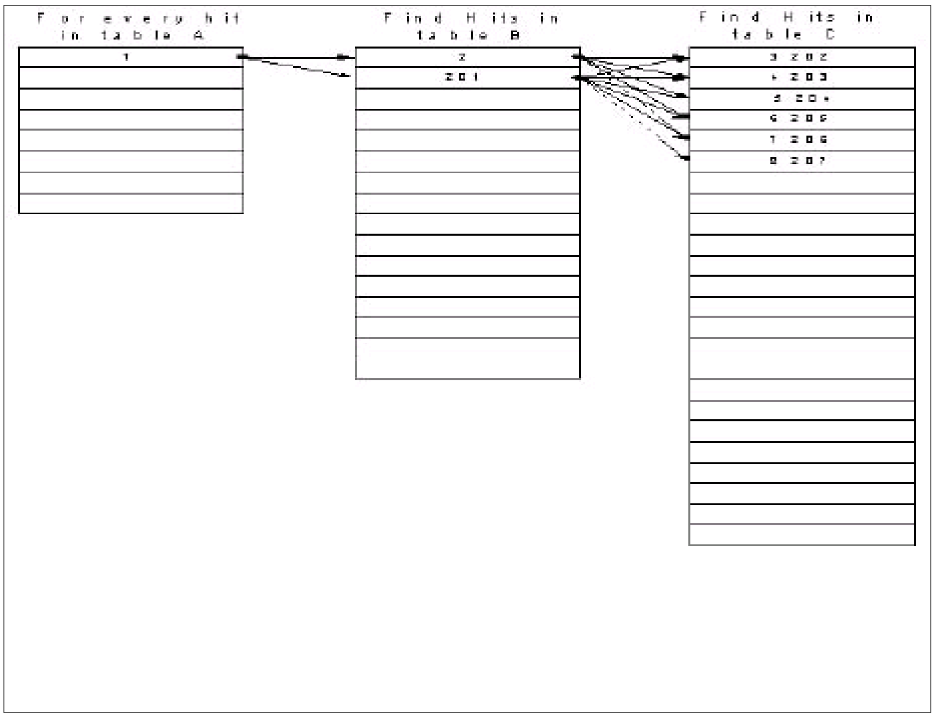
How Database Engines Execute Queries

Internally, a database engine composes data in a table with three main elements: index pages, data leaf tuples and statistical pages.

Data is stored in leaf tuples, typically in binary trees. Index pages point to data pages. When a SELECT statement is issued, index links can be followed to arrive at the desired data, or all data can be scanned. The database examines the statistical pages to decide an index path would reach the desired data, or whether a complete data scan would be most efficient.

Database engines are usually very fast when executing SELECT statements. Index pointers are constantly updated and optimized. If a row is inserted or updated, tree branches may need to be split up and reorganized.

How Database Engines Treat Joins



The illustration below shows a 3-way table join.

For every desired row found in Table A, rows are searched in Table B. For every desired row found in Table B, rows are searched in Table C.

Consider the situation where there are 10 qualifying hits in table A, 100 in table B and 1.000 in Table C. When done in a single query, this means that more than 100.000 records have to be searched via binary trees or data scans.

As the number of joined tables increases, the number of inner table iterations increases in a nonlinear fashion. This is why table joins usually are the biggest computational cost factor in a data driven application.

It may not be necessary to redesign the database schema when optimizing queries. Often, you can help the SQL engine pick a better path by reducing the number of joins in a query, for example by splitting a query. The following is a query across six table joins:

```
select a.col1, f.col7
from
a, b, c, d, e, f
where
    a.col2 = b.col2 and
    b.col3 = c.col3 and
    c.col4 = d.col4 and
    d.col5 = e.col5 and
    e.col6 = f.col6
```

This could be rewritten, as follows, to reduce the number of searches required:

```
select a.col1, c.col4

    into #temp1
    where
        a.col2 = b.col2 and
        b.col3 = c.col3
select #temp.col1, f.col7
from
#temp1, d, e, f
where
    #temp1.col4 = d.col4 and
    d.col5 = e.col5 and
    e.col6 = f.col6
```

The best way to achieve a high performance is to create a good E/R design in the first place. DBAs should be the most experienced staff on the project. However, for them to be able to help create a solid database design, you need to provide them with very detailed mock-ups. Of course, having followed the development approach advocated in this book, you have these mock ups readily available.

Professional database design cannot be learn in one weekend; experience is paramount. Still, the authors recommend SQL Puzzles and Answers by Joe Celko in order to learn about useful techniques, with good examples.

Stored Procedures

Experienced database developers implement stored procedures to help lessen the complexity of the first few steps in query execution. Stored procedures can use intermediate tables and forced path and they can hide complex operations from Java developers. An SQL expert can apply his or her skills here to optimize the query path.

Stored procedures can call other stored procedures. Sometimes, stored procedures are hundreds of pages long.

More SQL Tips

Beyond optimizing joins and using stored procedures, there are a number of other items that you may consider when optimizing data access performance.

Avoid ORDER BY on loaded databases. If a table contains millions of records, one hundred users issuing ORDER BY make the database sort millions of records one hundred times. Instead, create a clustered or covered index, or use GROUP BY for aggregation.

Make queries more restrictive where possible. A more restrictive query is almost always more efficient.

Consider creating temporary working tables in order to build optimal query paths.

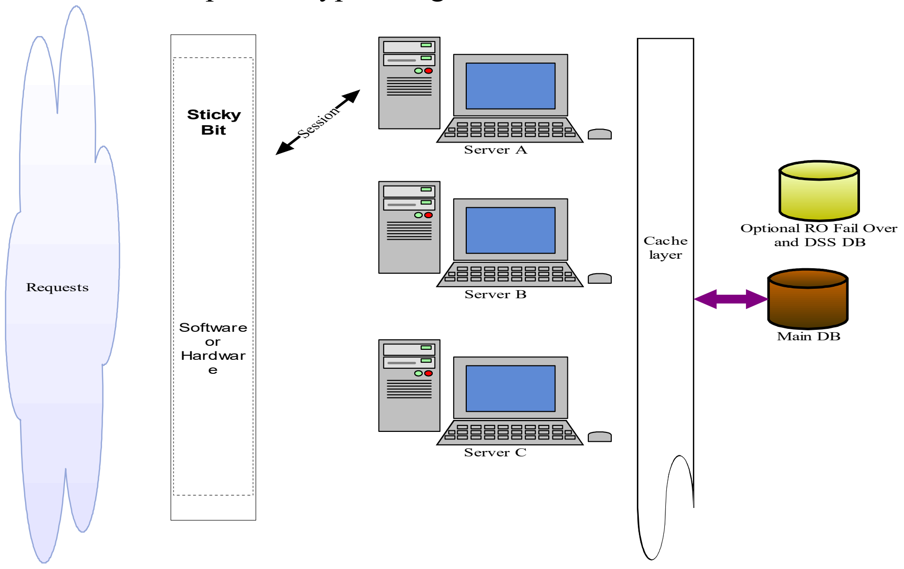
Using DISTINCT usually indicates a less than optimal database design.

Indexes can speed up queries, but they also slow down inserts and updates on highly transactional sites. You can have poor performance if a transaction causes an index page to split.

Always use db generated ID sequence as the PK.

Fail over

Here is an example of a typical high load environment.



Once you find a load of a single box , you can add boxes to get your desired scalability. You can see that there is only one DB, which is why SQL design is so important.
Typically you have a CISCO switch up front to set a sticky bit for a user's session. The sessions get assigned to the server in a round and robin scenario. Once a server gets a user session, all future requests go to that box. Any changes to the DAO get a distributed cache flush using oscache settings.
If a box fails, a user log into a new box and the form beans were saved to db as of a last event.

Review

Monitoring and stress testing the application is an important construction phase step.

LAB AG (Optional): Stress Test

OpenSTA

1. Install OpenSTA and re-start your machine when prompted.
2. Start the **OpenSTA Nameserver** from the start menu.
3. Start the **OpenSTA Commander** from the start menu.
4. Once inside the commander, right click on the "**Scripts**" folder and choose "**New Script**", then "**Http**"
5. Give the script a name, then double click on it to open the **Script Modeler** tool.
6. Choose **Record** from the **Capture** menu. OpenSTA will open up a web browser window and record the pages you browse to use for testing later. Browse to the http://localhost/do/tasks page, then view and/or edit several tasks. Once complete, click back on the **Script Modeler** tool and choose **Stop** from the **Capture** menu. Save this script.
7. Now that a test script has been created, right click on the **Tests** folder and choose "**New Test**", then "**Tests**".
8. After giving the test a name, double click on the test name to edit the test.
9. Click on the script that you created above, and drag it into the **Task 1** field of the test. This tells the test which script to run, a test can run multiple scripts. You can also set options for the task down below, such as how many times each user should run the task, etc..
10. Under the VUs column, enter the number of "virtual users" you want to be simulated in the test. Once you have the parameters setup the way you would like them, click on the green arrow in the tool bar to run the test.
11. Once the test has completed, the results will be listed under the results tab in a folder for the date/time the test was run. Open the folder, and select the checkboxs for the metrics you would like to view.

LAB AH Optional: Stored Proc.

Create and execute a stored proc using a DB Admin tool.

1. Ex:

Create TYPE Usr2 as (

```
id                int,
real_email_id     text,
name_nick_handle  text,
name_last         text,
name_first_init   text,
   role_code          text,
   region_code        text,
   city               text,
   source_code        text,
   topic_interest_code text,
   passwrd text
);
```

CREATE or REPLACE FUNCTION sp_usr2()

```
   RETURNS SETOF Usr2  AS
         'SELECT id,  real_email_id,  name_nick_handle, name_last,

          name_first_init,  role_code, region_code,
          city, source_code, topic_interest_code,
          passwrd
               FROM usr'
   LANGUAGE SQL;
```

SELECT * from sp_usr2();

Chapter 19: Making Your Applications Future-Safe

Release Engineering

The application is usually tested in a staging environment before being placed in production. The operations team takes the application WAR file created by the developer from the development environment to the staging environment.

A quality assurance (Q/A) team uses their testing methods on the application in the staging environment. Because bugs found in production can be very costly in terms of customer confidence, lost business, etc., it is highly unlikely that skipping Q/A will save you money.

When Q/A identifies a bug, the developer fixes the bug and creates a new WAR file for deployment in the staging environment. Since a bug fix or new feature may break something that was already tested, Q/A testing is very important. Q/A will have access to the requirements specification and was possibly writing the user documentation while development was going on.

Once Q/A approves, the operations team takes the WAR file from staging to the production environment. Q/A has the responsibility to catch bugs before production. One good way to predict when the application is stable enough is to watch the ratio of new bugs reported by beta testers vs duplicate reports of known bugs. Once there is a large % of duplicate bugs…. It means that we already know most of the faults, and the application is almost ready.

To keep clear lines of responsibilities, developers should not have access to the code in the staging or production environment. Developers are not responsible for issues appearing in production, since Q/A catches those issues. Q/A often also acts as technical support for users. Operations is responsible for operation of the SQL database and security of the Linux boxes.

The process can be summarized as follows:

1. Operations takes WAR from development to staging
2. Q/A tests the entire application
3. Developers fix bugs detected by Q/A (return to step 1)
4. Q/A approves the release

5. Operations takes WAR from staging to production

While you may have your production environment hosted off-site at a co-location facility in order to save $ on bandwidth, you should never outsource the roles of DBA operations and Security.

Change Management

If your application is successful, new modules will be added frequently. And you will be asked to create many more applications. How do you manage the versions of the JARs that you use in your different applications in /WEB-INF/lib? Image that you want to use Struts 1.3 jars, but you have a few applications in production that are Struts 1.1? Should you go back and regression test? Most of the time, the cost of configuration management is higher than the cost of regression test. That is keeping track of what application has what jars is high. It is cheaper to upgrade all the applications to the new "platform".

Each time you deploy, you are likely to use the latest version of the dozens of JARs in the application. If you have 5 applications with 12 JARs each, innumerable different combinations of JARs are possible, and you quickly have an organizational nightmare.

It is likely that you will be better off updating the existing applications to the latest JARs. Maven (see next section) can help with automating some of this chaos.

Ant and Maven best practices

Ant is an open source tool (http://ant.apache.org) to execute build and deployment tasks. Most IDEs come with Ant integrated. A best practice for Ant is actually configuring your IDE so that you do not have to use Ant or write and manually execute Ant build scripts. If you use Tomcat or Resin, you can have the development server point directly to your web-app directory, so you do not have to package your application to run. Eclipse will automatically recompile changes you make to the code and should copy the compiled classes to the web-inf/classes folder. Deployment may just mean copying the contents of the web-app folder to the production server, possibly using Tomcat (5) Manager.

Maven is a recent replacement for Ant. Maven is also an Apache project, available at http://maven.apache.org. Beyond build tasks, Maven allows you to organize, and document your project. OpenSource IDE support for Maven is forthcoming (http://mevenide.sourcforge.net), and using Maven should help you ease the burden of change management.

CVS

For team development, Concurrent Versions System *(CVS)* is a popular source control software. Eclipse works well with CVS. The CVS plugin to Eclipse is included with the Jasic suite, but you will need access to an existing CVS repository to use it.

Most Linux distributions come with CVS installed. After installation, this is how you unitize CVS. First make a place where cvs file will live such as:

```
mkdir /cvshome
```

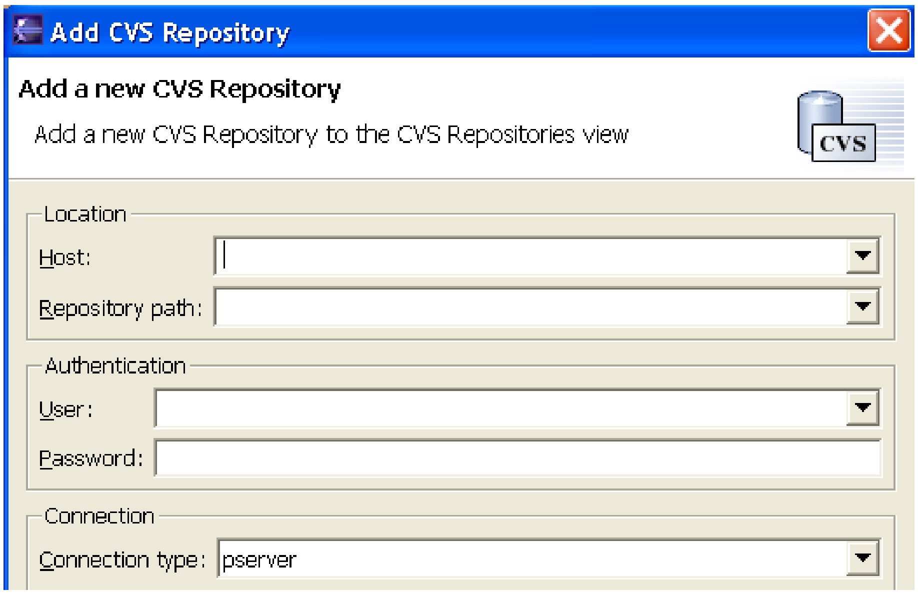
Then create the user group on Linux (using GUI tools) called cvsgroup. Create an account on your linux box for all the developers and then add them to the cvsgroup. Using the GUI (or command line) make sure that the group can read and write in the / cvshome folder. Then initialize CVS like this:

```
cvs -d /cvshome init
```

Then you can just create a blank file (using Vim or pico) in your home folder or a temp folder, and chdir to that folder that has one empty text file. You can create a project now via (v/r are optional):

```
cvs -d /cvshome import nameOfTheProject v r
```

You are now ready to start using it. First you have to add CVS to your IDE like this:


Add CVS Repository

Add a new CVS Repository

Add a new CVS Repository to the CVS Repositories view

Location

Host:

Repository path:

Authentication

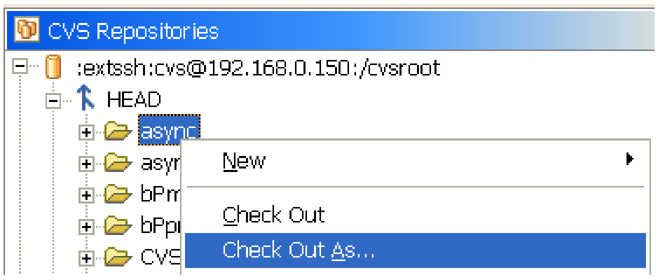
User:

Password:

Connection

Connection type: pserver

Then you need to check out your project and start adding file to it.



When you want to check back in, you just right click on the code and click team-synchronize with repository. Commit will send your files to the CVS and update will get new changes from other team members to your project. It’s a good idea to CVS sync at least a few times a day.

About once every 10 days, a project lead might tag the files in cvs with a version number, for example year_week, so that people can go back in time.
When creating a war file, one should note what year_week tag the release it is.

Fail over Recovery

A fail over procedure in case of system failure should be in place and it should be tested occasionally, so if anything ever does go wrong, .. everyone behaves normal.

While the DBA restores the production database from recovery tapes, you could switch the DNS to point to your staging server, with read-only database access. Once the production database is recovered, typically on a new db server, you can switch back to full read-write. Do you know how long it takes to restore your database from tape to a brand new box?

Fail over procedures should be tested at least twice a year. For example, in 3 weeks on a Saturday a manager gracefully shuts down the production db server. How long to recover to 100% operation, WITHOUT touching the DB server that was shutdown? Yes, a war stand by would be nice. I found hot stand buys to be in-effective.

Which application server to Use?

When choosing an application server platform, you may wish to consider an objective study recently undertaken by Netcraft (http://www.netcraft.com).

[TODO:Netcraft graphic here?]

Figure 20-1: Servlet engines in production

You note that the most popular platforms for production applications in April 2003 were Tomcat (25%), Resin (24.5%) and IBM (24.5%), followed by Oracle/Orion.

You should make sure that your application can work on multiple containers.

Of course, different application servers have different bugs and provide different degrees of leniency with regard to the code you write.

Web apps will grow even more in popularity, since they are so cheap to deploy an so cheap to manage. Imagine the client server days of going around upgrading the VB/ Delphi app on each users desk

Chains Filter

One popular thing to do is install a gZip filter for your application. This reduces page size that is sent via the internet, allowing you so sever more pages per second on the same line.

A Chain is a method called in one class that move up a hierarchy to find an objects that execute the method. For example a user request, wrapped in a context.
Each object that could handle the context would have an interface to handle the context and it could call the next object if it needs to pass it down the chain.

This can be done using a filter for example. A filter implements the filter with a doFilter(… signature.

A future version of Struts, code named Jericho (in CVS already) will replace the request processor with a chain. It will have a ContextBase object based on HashMap so that you can add messages will be passed down the chain. A chain will extend ChainBase, and be registered with a CatalogBase.

Returning false allows the chain to continue. The action signature for Struts 2.0 might look like this:

```
public boolean execute(Context context) throws Exception {
```

Design

Design is best explained as 80/20. You want to only design for the rule, not all the exceptions. This makes it an elegant system that can be maintained. Things that happen rarely are not part of design, but they are implemented as an exception. A new engineer might try to design a complex system, or one that handles all possible "exceptions". KISS. What you want to do is spend effort to see what you can remove.

What is Overrated

Certain technologies and methodologies for web application development have proved to be more controversial than others.

- Extreme Programming: Working without detailed specifications approved by the client is bound to fail. It seems to be a non scientific and a controversial approach

- UML : UML uses a process-oriented approach. Web application processes should be short and data-driven. UML may be good for documentation after construction. E/R diagrams are great. Test cases, however, are very useful.

- O/R: At the end, the clients want data displayed in tabular format, so E/R is just fine and easier to optimize with relational databases.

- EJB: Are relatively complex and are less scalable than other approaches, and should not ever be used for persistence. Visit "EJB's 101 Damnations" which provides 101 reasons why not to use EJBs (see: http://www.softwarereality.com/programming/ejb/index.jsp).

- JSF (Java Server Faces): Heavy UI processing on the server side, with potential scalability issues. Use Client side UI. JSF appears to do a lot of complex work on the server. It limits what graphic artists can contribute. Some even say that JSF is a "design by committee", or a solution for a probem we do not have.

- BluePrints: MVC is so much more powerful.

It happens again and again that people without commercial production experience advocate a technology. You may wish to ensure that you take advice from people who have actually deployed applications with the tools and approaches they recommend.

Common Mistakes

Here are some common mistakes that might save you time:

- Not having useful requirements

- Not declaring the JSP tag in JSP (JSP will ignore the tag)
- Using useBean or bean define (create bean in action instead)
- Mapping a bean in action mapping and then doing bean b = new bean() as opposed to bean b = (bean) formbean.
- Using properties in action. Action is a singleton.
- Using JDBC. You should use a DAO. DB access it the most problematic part of J2EE

And the parting word is:

“Always code to the least amount of astonishment”

** (From a book practical programmer)*

Appendix A

JASIC - an open-source integrated development suite for web applications

basicPortal CMS - a Struts-based portal application with content management

ABOUT JASIC

It is not trivial to build an application that is viable in the long-term. Developers can avoid costly and time-consuming mistakes if they invest thinking and resources into its architecture, rather than just build and fix it tactically.

Jasic is an integrated development suite for Java that leverages several of The Apache Foundation's Jakarta Struts projects into a comprehensive set of features and functionalities for web application development. It provides enormous productivity gains and allows developers to concentrate their resources on features that are unique to their project.

Jasic includes the following third-party software for web application development:

- Eclipse 3 IDE with SolarEclipse web applications plug-in
- Tomcat 5 and Resin 3 Application Server
- Fast JRockit 8.1 Java Virtual Machine (VM) from BEA Systems (or use JDK 1.4 from IBM or Sun)
- PostgreSQL 7.4 database
- pgAdmin II database tool (Windows only)
- iReport visual report designer (Windows only)

However, the main software component of Jasic is basicPortal CMS (bP), a dynamic, Struts-based portal application with content management that combines the functionalities and features that are required for about 80% of all web projects.

basicPortal is not limited to use with the above components. It works out-of-the-box with the components above, but experienced developers may use it with any other J2EE server, IDE or SQL database of their choice.

Developer benefits:

- Rapid installation of several open-source components that are pre-configured to work together.
- Greatly enhanced developer productivity (experienced coders should be able to produce at least two or three modules per day).
- Extremely solid and stable architecture (community's code contributions increase stability).
- Ease of use, yet suitable for large-scale data-driven web applications.
- Breaks complex applications into simple, consistent sets of components.
- Ensures all developers in the team develop code using the same approach.
- Encodes best practices as many users help debug and improve open source components.
- Large developer community provides virtually instant support.

Business benefits:

- Speedy and cost-effective development.
- Substantially reduced time-to-market and quicker product releases.
- Open source: access to source code at no charge.
- Freedom from closed, proprietary systems, thus no licenses to keep track of, renew and upgrade.
- No dependence on specialist labour and vendors.
- Numerous features meet a variety of business needs.
- Highly platform-independent.
- Greatly modular.

ABOUT BASICPORTAL CMS

basicPortal integrates the Jakarta Struts framework with an application server, IDE and connection pools to SQL.

While it is easy to use, basicPortal helps to build large-scale, database-driven web applications that perform extremely well. basicPortal includes support for e-commerce/credit cards, news, lead tracking, content syndication, fora, calendars, web logs ("blogs"), wikis, email support, high-speed standard J2EE security, row-based security, images, blobs and uploads.

basicPortal uses best practices in Java software design, such as J2EE container managed security, Model-View-Controller architecture (MVC 2), JavaBeans, JavaServer Pages (JSP), data access objects (DAO), the JSP Standard Tag Library (JSTL), caching and more.

It demonstrates accepted best practices in web Graphical User Interface (GUI) development using standards such as Cascading Style Sheet (CSS), Struts Tiles and Layout.

Key features:

- Runs on top of any J2EE application server, including Websphere, Weblogic, Oracle IAS, Borland, Novell exteNd, Tomcat, Resin and Orion.
- Is packaged with iBatis SQL database layer for object-relational mapping///SQL DB servers (PostgreSQL, MySQL, Oracle, etc.), but can store dynamic content in any database.
- Supports multi-row display tag, multi CRUD and user events.
- Combines best features of Expresso and Scaffolding.
- Working and configured Jasper Reports.
- Content management system contains sample HTML and XML data to build on.
- Flash Action Script forms for rich client user interface.
- DAO interface and database Layer implementation.
- Contains member sign up and administration facilities (user login, roles, etc.).
- Master Detail processing.
- Uses standard J2EE data source and JDBC Realms for user security.
- Supports RIA+SOA.

THE JASIC DISTRIBUTION

The Jasic distribution is an integrated development suite that includes the following programs:

- basicPortal
- Eclipse 3 IDE with SolarEclipse web applications plug-in
- Tomcat 5 and Resin 3 Application Server
- Fast JRockit 8.1 Java Virtual Machine (VM) from BEA Systems (or use JDK 1.4 from IBM or Sun)
- PostgreSQL 7.4 database
- pgAdmin II database tool (Windows only)

- iReport visual report designer (Windows only)

Variations on this platform are possible. The Java Development Kits (JDKs) from Sun and IBM can be used instead of JRockit, as long as the JDK 1.4+ compatible versions are used. basicPortal has been tested with Resin, Tomcat, and Orion, but other J2EE application servers compatible with JDK 1.4 and JSP 1.2/2.0 can also be used. baseBeans recommends the use of Eclipse with the SolarEclipse plug-in as an IDE, however others such as IntelliJ can be used as well.

PostgreSQL is the recommended database, but any SQL DB would do.

A database administration interface such as pgAdmin II for PostgreSQL is recommended. SQuirreL is another good tool that works with various databases such as PostgreSQL, Oracle, and MySQL.

Developers are encouraged to make platform choices that best suit their needs and the culture of their organizations. However, it should be kept in mind that variations on the platform will require extra configuration to ensure all pieces work together. Also, some development procedures may vary from those specified in other basicPortal documentation if customizations are made.

The latest version of the Jasic distribution for Windows 2000/NT/XP can be downloaded in one ZIP Archive from SourceForge (http://sourceforge.net/projects/basicportal). If you are reading this document from a CD, the CD should include the ZIP Archive.

Linux works great and is the ideal platfrom. At this time, users of Linux and other Unix-type systems must CVS source, download, install, and configure all components separately.

For all the platforms, developers will find the latest downloads on the baseBeans.com and Infonoia.com downloads page (http://www.basebeans.com and http://www.infonoia.com)

BASIC INSTALLATION

Unzip the Jasic installation zip file into **C:\jasic\** or **D:\jasic\**.

To minimize the possibility of conflicts, make sure that there is only one Java Development Kit on your server. This can be done in the *Control Panel -> Add/Remove Programs* interface and by searching for files such as javac.exe outside of the Jasic directory. The Java Virtual Machine that comes bundled with Windows should not cause a problem.

Next, put the Java directories in your system variables. If JRockit will be your JDK, go into the *Control* Panel -> System -> Advanced -> Environment Variables section and make the following changes:

```
      JAVA_HOME should be C:\jasic\jrockit81

      PATH should be %JAVA_HOME%\bin;%SystemRoot%\
system32;%SystemRoot%
```

Note: A frequent error in configuration is not ***setting the one system environment path to*** *%JAVA_HOME%\bin;%SystemRoot%\system32;%SystemRoot%.*

If you installed Jasic on a drive other than C:, replace **C:\** with the proper drive designation for your system.

Bring up a command prompt by running the program **command.com** (in *Start -> Run*). Issue the command:

```
      C:\>java -version

      java version "1.4.1"

      Java(TM) 2 Runtime Environment, Standard Edition (build 1.4.1)

      BEA WebLogic JRockit(R) Virtual Machine (Native Threads,
Generational      Concurrent Garbage Collector)
```

If the outcome of the command is consistent with the JDK you have selected for development, you can proceed to the next steps.

For any component installation question, you can refer to cheat sheet on baseBeans.com Cheat Sheet page.

It is recommended that you download and install a Netscape (www.netscape.com) or Mozilla (www.mozilla.org) browser. Some tags that work in Internet Explorer may not work in other browsers, so it is best to test on a variety of platforms.

INSTALLING AND CONFIGURING THE DATABASE

You must have some SQL engine to create the tables on since data is stored in a SQL DB.

To install the bundled PostgreSQL software, first run the PostgreSQL Windows installer in **\jasic\pgSQL.** Then run the pgAdmin II installer in the same directory. Accept most defaults in the process, other than the installation directories, which can be set to **\jasic\PostgreSQL** and **\jasic\pgAdmin2.**

Once the database programs are installed, make sure the PostgreSQL service is started in Control Panel -> Administrative Tools -> Services. Then run pgAdmin II from the Windows Start menu. Click the icon in the upper left corner to connect to the database. Click OK at the next dialog box to connect as the default superuser named **postgres**; you may connect with no password from the local computer.

Under Users in the left panel object tree, right-click on **postgres** and select *Change Properties*. Give this user a new password for increased security.

Next, select *(right-click) localhost -> Create Object -> User*. Create a new user called **bpuser**. Give this user a secure password, and grant the user the ability to create new databases. Take note of the user's expiration date and set it far into the future if desired.

Exit pgAdmin II and reconnect as **bpuser**. Select (right-click) localhost -> Create Object -> Database. Create a database called **basicportal** (no data entry other than the database name is required in the dialog box). It is a good idea to use all lower case names in PostgreSQL.

You will now have to run some prewritten SQL scripts to create the tables for use with basicPortal and some sample table data. These scripts are located in **\jasic\bp\WEB-INF\doco**. Under *Databases* in the left panel object tree, select *(right-click) basicportal -> SQL*. Select *Load*, navigate to the file **\jasic\bp\WEB-INF\doco\ table_create.sql**, then select *Open* and *Execute*. The basicPortal tables should then be created. Repeat this step for the file **\jasic\bp\WEB-INF\doco\ table_sample_data.sql**.

Table data can be viewed and edited by right-clicking on the table name and selecting *View Data*. If you will be writing additional modules for basicPortal, you should familiarize yourself with these tables. The table **usr** contains user information and passwords, and is used to implement logins and security. The table **content** is used to enable content management, and contains web page data.

For information on how to manipulate these tables by hand, please see the documentation for PostgreSQL (www.postgresql.org) and pgAdmin II (pgadmin.postgresql.org).

Note: You can save yourself time by testing a JDBC connection via something like http://squirrel-sql.sourceforge.net

STARTING THE APPLICATION SERVER

After database installation, the system is ready to be activated. All other Jasic components are preconfigured for easy use.

If you installed Jasic on a drive other than C:, edit the file **\jasic\resin-3\conf\resin.conf**. Change the drive letter in the following line:

```
<web-app id='/' app-dir='c:\jasic\bp'>
```

Also, change the db connection settings.

```
<init-param url="jdbc:postgresql://localhost:5432/basicportal?
autoReconnect=true"/>

<init-param user="bpuser"/>

<init-param password="changeme"/>
```

Same is true of other app. servers. There is a sample for Tomcat and Resin

Next, start Resin by running the file **\jasic\resin-3\bin\httpd.exe**. A terminal window will appear, and Resin should report that it is started and listening.

In your web browser, navigate to *http://127.0.0.1/test*. This will test the database connection, and should display the contents of one user object and one content object. Then navigate to *http://127.0.0.1*. A sample basicPortal web site should appear. basicPortal is now installed and running!

You need to customize the content and user rights.

If you are configuring your own container, make sure JDBC driver is in commons/lib.

Ex: JDBC Realm setting

```
<Realm className="org.apache.catalina.realm.JDBCRealm" debug="99"

               driverName="org.postgresql.Driver"

           connectionURL="jdbc:postgresql://localhost:5432/

                basicportal?autoReconnect=true"

         connectionName="bpuser" connectionPassword="changeme"

userTable="usr" userNameCol="real_email_id" userCredCol="passwrd"
      userRoleTable="usr" roleNameCol="role_code" />
```

Ex: App. Context setting

```
       <Context path="" docBase="c:/jasic/bp" debug="0"

              reloadable="true" crossContext="true">

       <Resource name="bppool" auth="Container" type="javax.sql.DataSource"/>

              <ResourceParams name="bppool">

                     <parameter>

                            <name>password</name>

                            <value>changeme</value>

                     </parameter>

                     <parameter>

                            <name>url</name>

<value>jdbc:postgresql://localhost:5432/basicportal?autoReconnect=true</
value>

                     </parameter>

                     <parameter>

                            <name>driverClassName</name>

                            <value>org.postgresql.Driver</value>

                     </parameter>

                     <parameter>

                            <name>username</name>

                            <value>bpuser</value>

                     </parameter>

                     <parameter>

                            <name>factory</name>
```

```
    <value>org.apache.commons.dbcp.BasicDataSourceFactory</value>

                </parameter>

                <parameter>

                        <name>maxIdle</name>

                        <value>1</value>

                </parameter>

                <parameter>

                        <name>maxActive</name>

                        <value>3</value>

                </parameter>

                <parameter>

                        <name>maxWait</name>

                        <value>5000</value>

                </parameter>

                        <parameter>

                        <name>removeAbandoned</name>

                <value>true</value>

                </parameter>

                <parameter>

                        <name>removeAbandonedTimeout</name>

                        <value>20</value>

                </parameter>

                </ResourceParams>

        </Context>
```

```
            </Host>
```

In the case of Tomcat, you will also need to copy the dbcp and commons-jar to common/lib, so that the connection pool will work in Tomcat.

PREVIEW OF BASICPORTAL USER MODULE

basicPortal is an example of the code needed on 80% of all web projects, implemented using the best practices and design available. It has sample beans, Struts Actions, and DAOs that can be used and extended by developers. It also contains sample XML files with settings for Struts Menus, Validators, and Tiles so that these files do not have to be written from scratch. basicPortal is currently running on http://www.basebeans.com and http://www.infonoia.com .

Users can register to become members of the site, and administrators can edit their information and privileges.

PREVIEW OF THE BASICPORTAL CONTENT MANAGEMENT MODULE

basicPortal has a content management system (CMS), allowing administrators to add text or HTML content through an easy web interface. All pages are stored in the system database.

Each web page has an identifying tag such as **HOME** or **ABOUT**. Any page can be displayed easily by passing basicPortal an HTTP parameter such as **content=ABOUT**.

```
public class BaseBean extends ValidatorForm implements Collection {

public final static Log log =

LogFactory.getLog(org.apache.scaffoldingLib.base.BaseBean.class.getNa
me());

protected Dao _dao = null;

protected List _rList = null; // results

protected Map _current = null; // property

public BaseBean() {

      _rList = new List(); }

public String testIt() throws Exception {

      throw new Exception("no test implemented"); }
```

```
public void find (Object parm) {

      _dao = myDao();

      _dao.populate(Long parm);

      _rList = dao.getList()

}

public String getX () {

      return(String) List.get("x");

}

public void setX(String arg) {

      List.put(x,arg);

}
```

CODE SAMPLE

Above is a code sample illustrative of the kinds of techniques used in basicPortal. It is a DAO helper class which contains a status log and a list of results retrieved from its Data Access Object. The object methods allow unit testing to be implemented, and allow the list to be viewed and modified.

SUPPORT

Infonoia, Europe - www.infonoia.com

Infonoia - a play on information and paranoia - builds professional Java/J2EE applications for the Internet, Intranet and Extranet and provides Java/J2EE and Struts training, mentoring, development, products and support (Infonoia's team speaks English, French and German). Infonoia is a key contributor to basicPortal and provides commercial support for JASIC and basicPortal CMS in Europe.

Infonoia has extensive experience in application server technologies and open source standards. Indeed, it has been the first company in Switzerland and among the first in Europe to use application server technology back in 1998. Its architects and programmers mastered Java web applications long (in IT terms) before it became a standard with J2EE.

Infonoia's clients include public and private organisations, such as Deloitte & Touche, Tetra Pak, Lombard Odier Darier Hentsch or the Canton and Republic of Geneva.

Feel free to contact jasic@infonoia.com with any questions about JASIC, basicPortal or other training, mentoring, development and support services and products.

Infonoia SA

Rue de Berne 7

1201 Geneva

Switzerland

Phone: +41 22 900 00 09

Fax: +41 22 900 00 18

Email: jasic@infonoia.com

www.infonoia.com

baseBeans Engineering, United States - www.basebeans.com

baseBeans Engineering is the author of Jasic and basicPortal CMS and provides commercial support for both applications in the USA. The company provides best practices web development services and has worked for NASA, CSC, Ford Motor Company, Bank of Amercia and others. Accomplishments include working with large teams, budgets of over one million dollars, and terabyte size databases with over 40,000 concurrent users.

Vic Cekvenich is a principal developer at baseBeans Engineering. He specializes in project recovery, has written a book on using the MVC/Struts framework with Data Access Objects (DAO) 14 months ahead of others and was named Trainer of the Year by the Java Developer's Journal.

Appendix B – Tomcat

TODO: Tomcat deployer

BugFinder

LuntBuild

Appendix C- Linux Scripts

Remote access to Linux using VNC

Make sure the VNC is installed on the Linux server. Run
rpm -q vnc-server
to check if you have it.

If you don't already have it installed, you can find it
on the RedHat CD and install it.

On the server, run
vncserver
the first time, it will set a password for the vnc session.
it will also create the default $HOME/.vnc/xstartup file
which you can tweak if you want. You can check the logs
in $HOME/.vnc/ directory to make sure everything starts fine.
It will say "desktop is your_hostname:#" where # is a number.
If you are the only VNC user on the system, it will default
to 1.

On your MS Windows PC, download the VNC viewer from
www.realvnc.com
Again, you only need the VNC viewer. Here is the direct URL
as of today:
http://www.realvnc.com/dist/vnc-3.3.7-x86_win32_viewer.exe
You might want to create a shortcut on your desktop once
you've downloaded it. You can then click the shortcut
to launch the viewer.
it will prompt you for "VNC Server" where you should
put your_linux_servername_or_ip:#

(you have to use the # number as server told you earlier).
It will them prompt for session password, which is the password you just setup on the server side.

voila! you have the X on MS Windows.

random notes:

1) you can click the top left icon on the VNC viewer for a list of options.
2) you can close or disconnect your VNC viewer, and restart and reconnect to the VNC server to see your old session. The old session will be kept on server until you kill the vncserver.
3) on the server, you can stop the vncserver by running vncserver -kill :#   (where # is your X display number)
4) the connection between vnc viewer to vnc server is not encrypted, just like normal ftp/telnet.
   so you should only use it on a local trusted LAN.
   you should consider using ssh tunneling if you want to use VNC over the untrusted Internet.
   check this URL for details:
   http://www.uk.research.att.com/vnc/sshvnc.html

Appendix D – Jdate Time

CopyRight **Spasic**

JDateTime provides precise, functional and configurable date and time manipulation. It has been designed to have all of the functionalities of similar date/ti,e classes, but to be simpler to use, and yet enhanced and fast as possible. JDateTime should satisfy the most of date/time needs.

JDateTime uses Astronomical Julian Date Numbers to store date/time information as Julian Dates. Precision of JDateTime is set to 1 millisecond.

Initialization

JDateTime can be initialized in many ways. Default initialization sets the current date and time. Further, it may be set to a specific date and/or time. Number of milliseconds since 1970 (e.g. System.currentTimeMillis) may also be used. Date and time of JDateTime object can also be set by each field independently, if there is a need for that. And at the end, JDateTime can be initialized or set from instance of any available Java date/time class, as well as from String, as it will be shown later.

```
JDateTime jdt = new JDateTime();          // current date/time
jdt.set(1999, 03, 21, 19, 50, 17.321); // specific date/time
jdt.set(2001, 07, 13);                 // specific date
jdt.setYear(2003);                        // sets just specific year
jdt.set(System.currentTimeMillis());   // milliseconds
```

Java time/date classes

JDateTime, as said above, can be initialized from an object of any available Java date/time class:

- java.sql.Timestamp
- java.util.GregorianCalendar
- java.util.Calendar
- java.sql.Date
- java.util.Date

But JDateTime can also act as a factory for above classes: it can return a new instance of any of above classes. There is also a possibility to store date/time values to an existing object of above classes.

```
jdt = new JDateTime(CalendarInstance);      // from calendar
jdt.loadFrom(TimestampInstance);            // date/time from calendar
Calendar c = (Calendar) jdt.getInstance(Calendar.class);      // new calendar instance
Calendar c = jdt.getCalendarInstance();     // shortcut for previous
jdt.storeTo(c);                       // store to existing calendar
```

Besides above default Java date/time classes, it is possible to add so-called 'converters' for any other custom date/time class.

Strings

One of the power features of JDateTime is a possibility to set time from and to get time as a String. There is a default date/time format, but it is also possible to specify custom format, both globally or just for one method call. JdtFormater implementations may be used for custom date/time string setting and getting. Default formatter uses enhanced set of patterns defined by ISCO 8601 standard.

```
jdt.get();                                  // get date/time in default format
jdt.get("YYYY.MM.DD");                      // get date in specified format
jdt.set("01-01.1975", "DD-MM.YYYY");        // set date from specified format
```

.

If we complicate things, they get less simple.

-- Professor at Cambridge University

...Simplifications have had a much greater long-range scientific impact than individual feats of ingenuity. The opportunity for simplification is very encouraging, because in all examples that come to mind the simple and elegant systems tend to be easier and faster to design and get right, more efficient in execution, and much more reliable than the more contrived contraptions that have to be debugged into some degree of acceptability....Simplicity and elegance are unpopular because they require hard work and discipline to achieve and education to be appreciated.

▰ Edsger W. Dijkstra

Complexity is a sign of technical immaturity. Simplicity of use is the real sign of a well design product whether it is an ATM or a Patriot missile.

▰ Daniel T. Ling

Simplicity is the soul of efficiency.

-- Austin Freeman, "The Eye of Osiris"

Perfection (in design) is achieved not when there is nothing more to add, but rather when there is nothing more to take away.

▰ Antoine de Saint-Exupéry

Controlling complexity is the essence of computer programming.

-- Brian Kernighan

The difference between a good and a poor architect is that the poor architect succumbs to every temptation and the good one resists it.

▰ Ludwig Wittgenstein

▰

Simplicity does not precede complexity, but follows it.

Alan J. Perlis

Best Practices Struts

Coverage of Tomcat 5, Struts 1.2, Eclipse 3 and more:

- Reporting
- Stress Test
- CMS
- Memberships
- Security
- Shopping Cart
- Searching
- Drop Down Options
- Forums
- Blogs