

1. Home	3
1.1 AppFuse QuickStart	4
1.1.1 Maven for Newbies	5
1.2 Commercial Support	6
1.3 Demos and Videos	8
1.4 FAQ	9
1.5 News	14
1.5.1 Articles	21
1.5.2 Blogs	21
1.5.3 Presentations	22
1.6 Project Information	22
1.6.1 Developer Guide	24
1.6.1.1 Continuous Integration	25
1.6.1.2 Project Policies	25
1.6.1.3 Release Process	26
1.6.2 Issue Tracking	28
1.6.3 Mailing Lists	29
1.6.4 Source Repository	29
1.6.5 Sponsors	30
1.7 Reference Guide	32
1.7.1 Ajax	32
1.7.2 AppFuse Maven Plugin	38
1.7.3 CSS Framework	39
1.7.4 Database Profiles	40
1.7.5 IDEs	40
1.7.5.1 Eclipse	40
1.7.5.2 IDEA	42
1.7.5.3 MyEclipse	43
1.7.5.4 NetBeans	43
1.7.6 Licenses	44
1.7.7 Maven 2	45
1.7.7.1 Maven Plugins	47
1.7.8 OS Specific Issues	47
1.7.9 Persistence Frameworks	48
1.7.9.1 Hibernate	48
1.7.9.2 iBATIS	49
1.7.9.3 JPA	50
1.7.10 Security	50
1.7.10.1 Apply Security to Managers and DAOs	51
1.7.10.2 CAPTCHA Integration	51
1.7.10.2.1 CAPTCHA Integration for Spring Modules	54
1.7.10.3 Configuring the Menu System	55
1.7.10.4 Database Encryption with Jasypt-Hibernate	55
1.7.10.5 LDAP Authentication	60
1.7.10.6 Protecting Actions and Controllers	67
1.7.10.7 Secure JSF components	67
1.7.11 SiteMesh	73
1.7.12 Spring	74
1.7.13 Version Control	74
1.7.13.1 Hosting	74
1.7.14 Web Filters	75
1.8 Tutorials	76
1.8.1 Development Environment	77
1.8.1.1 Installing an SMTP Server	78
1.8.1.2 Using JRebel with IntelliJ IDEA	79
1.8.2 Enterprise Integration	80
1.8.2.1 Clustering with Terracotta	80
1.8.2.2 JBPM Integration	86
1.8.3 Migration Guide	94
1.8.3.1 Release Notes 2.1.0 M1	95
1.8.3.2 Release Notes 2.1.0 M2	97
1.8.3.3 Release Notes 2.1.0	100
1.8.4 Persistence	104
1.8.4.1 AppFuse Core Classes	107
1.8.4.2 Using Hibernate	109
1.8.4.2.1 Java 5 Enums Persistence with Hibernate	113
1.8.4.2.2 Managing Hibernate through MBeans	119
1.8.4.3 Using iBATIS	119
1.8.4.4 Using JPA	127
1.8.5 Services	131
1.8.5.1 Web Services	140
1.8.6 Web	142
1.8.6.1 Using JSF	143
1.8.6.2 Using Spring MVC	160
1.8.6.3 Using Struts 2	175



# Home

**AppFuse** is an open source project and application that uses open source tools built on the Java platform to help you develop Web applications quickly and efficiently. It was originally developed to eliminate the ramp-up time found when building new web applications for customers. At its core, AppFuse is a project skeleton, similar to the one that's created by your IDE when you click through a wizard to create a new web project.

AppFuse 2 is a restructuring of AppFuse 1.x to use Maven, Java 5 and annotations. The major reasons we use Maven are:

- Dependency downloading
- Easier maintenance (everything can be compiled at once)
- Easier upgrading for end-users

If you'd like to help out or ask questions about AppFuse 2, please do so on the [user mailing list](#). If you'd like to use AppFuse 2, see the [Demos and Videos](#) and use the [QuickStart Guide](#) to get started.



## April 4, 2011: AppFuse 2.1.0 Released!












Please see the [QuickStart Guide](#) to start using AppFuse or read the [News Announcement](#) to see what's new.

Features in AppFuse 2:

- Maven Integration
- Spring Annotations
- Web Frameworks support: JSF, Struts 2, Spring MVC, Stripes, Tapestry 5 and Wicket
- JDK 5, Annotations, JSP 2.0, Servlet 2.4
- JPA Support
- Generic CRUD backend
- Full Eclipse, IDEA and NetBeans support
- Fast startup and no deploy with Maven Jetty Plugin
- Testable on multiple appservers and databases with Cargo and profiles

Please use the menu on the left to navigate this site.

## Recent Project News

Title	Author	Date Posted
 <a href="#">AppFuse 2.1 Released!</a>	<a href="#">Matt Raible</a>	Apr 04, 2011
 <a href="#">AppFuse 2.1.0 Milestone 2 Released</a>	<a href="#">Matt Raible</a>	Nov 15, 2010
 <a href="#">AppFuse 2.1.0 Milestone 1 Released</a>	<a href="#">Matt Raible</a>	Nov 19, 2009
 <a href="#">A Letter to the AppFuse Community</a>	<a href="#">Matt Raible</a>	Nov 05, 2009
 <a href="#">New Tutorials - External Sorting with Display Tag and Integrating Compass 2.0</a>	<a href="#">Matt Raible</a>	May 22, 2008
 <a href="#">New Documentation!</a>	<a href="#">Matt Raible</a>	May 13, 2008
 <a href="#">AppFuse 2.0.2 Released</a>	<a href="#">Matt Raible</a>	May 11, 2008
 <a href="#">MarkMail and FishEye</a>	<a href="#">Matt Raible</a>	Dec 05, 2007
 <a href="#">Integrating Crowd with AppFuse and Acegi</a>	<a href="#">Matt Raible</a>	Nov 28, 2007
 <a href="#">AppFuse 2.0.1 Released</a>	<a href="#">Matt Raible</a>	Nov 26, 2007
 <a href="#">Thanks goodness for free bandwidth</a>	<a href="#">Matt Raible</a>	Sep 20, 2007
 <a href="#">AppFuse 2.0 Released!</a>	<a href="#">Matt Raible</a>	Sep 18, 2007
 <a href="#">AppFuse 2.0 RC1 Released</a>	<a href="#">Matt Raible</a>	Sep 04, 2007

 AppFuse vs. Grails vs. Rails	Matt Raible	Aug 10, 2007
 New Server at Contegix	Matt Raible	Jul 27, 2007

## AppFuse QuickStart

To start developing Java applications with AppFuse 2, please following the instructions below:

### Table of Contents

1. [Create](#) a new project.
2. [Run](#) it.
3. [Have Fun](#).

### Create a project

1. Setup your [Development Environment](#) - or follow the steps below if you're a veteran.
  - a. [Download](#) and install JDK 5+ (make sure your JAVA\_HOME environment variable points to the JDK, not a JRE).
  - b. [Download](#) and install MySQL 5.x (or use a different database [see here](#)).
  - c. Setup a local [SMTP server](#) or change `mail.properties` (in `src/main/resources`) to use a different host name - it defaults to "localhost".
  - d. [Download](#) and install Maven 2.2.1+.
2. From the command line, cd into your "Source" directory (c:\Source on Windows, ~/dev on Unix) run the Maven command you see below after choosing your web framework and other options.



#### Choose your Stack

AppFuse comes in a number of different *flavors*. To optimize and simplify your experience as a user, we've created a number of different archetypes (a.k.a. starter projects). There are currently three types of AppFuse Archetypes: light, basic and modular. Light archetypes are bare-bones, basic archetypes contain User Management and Security and modular archetypes contain "core" and "web" modules and are ideal for creating projects that have a re-usable backend. You should change the **groupId** to match your preferred package name and the **artifactId** to match your project's name.

Your browser does not support iframes



The warnings you see when creating your project are expected. If you see **BUILD SUCCESSFUL** at the end, your project was created successfully.

If you have issues using the **archetype:generate** command above, such as, **ResourceNotFoundException** or **Error merging velocity template**, then you may have an older or unsupported version of the maven-archetype-plugin. You can use the maven coordinates to specify which version of the plugin should be used: **mvn org.apache.maven.plugins:maven-archetype-plugin:2.0-alpha-4:generate -B ....** This command will download and add the specified plugin to your local maven repository in order to execute the goal correctly.

You should be able to run AppFuse immediately if you have a [MySQL 5.x](#) database installed and it's accessible to root using no password. If you'd prefer to use an embedded database, we've recently added [Database Profiles](#) for H2, HSQLDB, etc.

### Run your application

Running AppFuse is easy now. Once the archetype project is created, Maven will create and populate your database using the [hibernate3](#) and [dbunit](#) plugins, respectively. All you have to do is use Maven to run the Jetty container and view your application.

1. Decide if you want to change AppFuse from "embedded mode" to full-source mode by running **mvn appfuse:full-source** from your project's root directory. When you run `appfuse:full-source` after checkin you may run into problems.



#### For AppFuse versions < 2.1.0

The SVN repository location has changed. To avoid errors like "svn: PROPFIND request failed on" you'll need to add the following to the `appfuse-maven-plugin`'s configuration section for everything to work. Add this to the created pom file in your project root.

```
<trunk>https://svn.java.net/svn/appfuse~svn/</trunk>
```

2. To view your application run **mvn jetty:run** from your project's directory (for a **modular** project, you'll need to run **mvn jetty:run** from your project's **web** directory (after installing the core module)). Maven will start Jetty and you should be able to view your application in your browser at <http://localhost:8080>.
3. Check your new project into source control, unless you have a good reason not to. [Google Code](#) and [GitHub](#) are good options.
4. Run **mvn** from within your project to download JARs, Tomcat and run the integration tests in your project.



The default username/password for an admin user is **admin/admin**. For a regular user, use **user/user**.

If you receive OutOfMemory errors when using **mvn jetty:run**, see [this mailing list thread](#).



### Changing database settings

To change your MySQL database settings, simply change the `<jdbc.*>` properties at the bottom of your pom.xml. See [Database Profiles](#) to use a database other than MySQL.



#### MySQL Security

Running MySQL using root with no password is not the most secure thing to do. Once your database is created, you can change the root password using the command below:

```
mysql --user=root --pass='' mysql -e "update user set
password=password('newpw') \
where user='root'; flush privileges;"
```

AppFuse uses the username "root" and a blank password by default. To change these values, modify the `<jdbc.username>` and `<jdbc.password>` properties in your project's pom.xml (at the bottom).

## Develop your application

You can develop your application using [Eclipse](#), [IDEA](#) or [NetBeans](#). For Eclipse, [install m2eclipse](#) and import your project. For IDEA, simply use File > Open Project and point to your project's directory. Further instructions can be found in the [IDE Reference Guide](#).

The [Tutorials](#) should help you get started developing your application.

## Maven for Newbies

This page is intended to be a place to document my impressions about Maven and AppFuse 2.x as I take my AppFuse 1.9.x experience and move forward into the future of webapp development.

Now for a bit of background about me. I'm the Development Supervisor for [Sum-Ware, Inc.](#), which basically means I make sure the other developer and network admin have something to do and I have to get done the stuff that's left.

I'm taking a two pronged approach to learning AppFuse 2.x. At work I have a Windows desktop running Sun Java 1.6 and Eclipse 3.2. The project I'm starting at work uses the "appfuse-basic-struts" archetype. At home I'm doing a different project on my Ubuntu laptop with Sun Java 1.6 and Eclipse 3.2 (I may switch to IDEA on this project--dunno yet). This project is built on the "appfuse-modular-struts" archetype.

I'm sticking with Struts 2 (at least for now) for simplicity and my own sanity. I'd like to limit the number of variables as much as I can during this learning process.

---

### Targets v. Goals

- The first thing that I noticed is the maven "goals" are similar, but different than ant "targets." Goals are cool because they are universal and apply to all maven projects, but targets are a bit easier to figure out because they exist entirely within your build.xml file (more or less). The [Maven 2](#) page has a good list of commonly used goals and what they are for.
- I must say even if it is not obvious how to use Maven in the beginning I must say it is pretty easy to get started with it. Once the prerequisites were installed (none of which were hard to do), I had AppFuse running within 10 minutes. And most of that time was spent letting maven fetch dependencies.

### IDE integration

- Eclipse integration is pretty nice. But I did have a couple of weird things happen.

- When creating my POJO I had to clean the project a couple of times to get the Eclipse to recognize the annotations
- I ran the command to create the M2\_REPO stuff in my project. That did work, I could see all the dependant JAR's listed in the project, but it didn't fix the annotation problem mentioned above.
- I had to add the M2\_REPO variable to the environment as well. After that things went pretty smooth.
- I have not used IDEA with AppFuse 2.x yet, but I hear it is better than Eclipse. 😊

## Dependency downloads

- This is really cool and a bit weird at the same time. I have gotten a number of Warning and Error messages while maven is downloading dependencies. But the really weird part is stuff seems to still work. Check out this output for example:

```
[INFO] Wrote settings to
/home/nathan/src/fetch-calendar/web/.settings/org.eclipse.jdt.core.prefs
[INFO] Wrote Eclipse project for "fetch-calendar-webapp" to /home/nathan/src/fetch-calendar/web.
[INFO]
Sources for some artifacts are not available.
List of artifacts without a source archive:
  o org.apache.struts:struts2-core:2.0.6
  o commons-fileupload:commons-fileupload:1.1.1
  o opensymphony:clickstream:1.0.2
  o org.apache.struts:struts2-spring-plugin:2.0.6
  o velocity-tools:velocity-tools-view:1.0
  o javax.servlet:jsp-api:2.0
  o org.apache.struts:struts2-codebehind-plugin:2.0.6
  o freemarker:freemarker:2.3.8
  o org.tuckey:urlrewrite:2.5.2

[INFO]
[INFO]
[INFO] -----
[INFO] Reactor Summary:
[INFO] -----
[INFO] AppFuse Modular Application ..... SUCCESS [6.338s]
[INFO] AppFuse Modular Application - Core ..... SUCCESS [57.907s]
[INFO] AppFuse Modular Application - Web (Struts 2) ..... SUCCESS [2:20.721s]
[INFO] -----
[INFO] -----
[INFO] BUILD SUCCESSFUL
[INFO] -----
[INFO] Total time: 3 minutes 27 seconds
[INFO] Finished at: Tue Apr 03 22:05:12 PDT 2007
[INFO] Final Memory: 48M/63M
[INFO] -----
```

How can struts2-core fail to download, yet I can run "mvn jetty:run-war" perfectly? I have no idea... :?

Okay... I'm dumb. I just realized it can't get the source packages, but the binaries were downloaded just fine. Maybe someone else will

read this and avoid public humiliation 😊

## Hello World

I'm sure there is a reason for the App.java and AppTest.java (and Core.java for modular archtypes) classes that get created when you make a new project. I just have no idea what that is. Maybe someone could link to an answer?

## Maven and HTTP proxy settings

If your local network is behind a HTTP proxy (note that this is **different** from a Maven repository proxy) you will see something like 'Error transferring file'. Find out the hostname and port of your proxy and add the following to your command-line: -DproxySet=true -DproxyHost=<your proxy host> -DproxyPort=<your proxy port>

Note: if you've tried and failed a few times, the central repository may have become blacklisted. Add the -U command line option to force an update.

## Commercial Support

Commercial support for AppFuse is offered by [Raible Designs, Inc.](#). Raible Designs was started by Matt Raible, the founder of AppFuse. Matt has intimate knowledge of AppFuse, as well as all the frameworks that it uses, particularly Spring, Hibernate, Maven 2, Ant, Ajax, DWR, JSF, Struts 2, Spring MVC and Tapestry. In addition, Matt has worked with many clients over the years to help setup **development ecosystems** that facilitate solid, maintainable applications. IDEs, Issue Trackers, Source Control, Continuous Integration and Automated Releases can make your software development team a well-oiled machine.

Raible Designs currently offers a number of different support options: from phone/e-mail support to consulting and training. Prices and services are listed below. Please [contact us](#) for more information.

1. [AppFuse Support](#)
2. [Consulting](#)
3. [Training](#)

## AppFuse Support

The support packages below include both phone and e-mail support. [Contact us](#) to purchase a support agreement or if you're interested in longer-term support services and rates. If you need faster response times, that's certainly something we can provide.

	10 hours/month	20 hours/month	40 hours/month
<b>Support Hours</b>	8-5, M-F, MST	8-5, M-F, MST	8-5, M-F, MST
<b>Response Time</b>	8 hours	8 Hours	8 Hours
<b>Incident Limit</b>	Unlimited	Unlimited	Unlimited
<b>Number of Contacts</b>	Unlimited	Unlimited	Unlimited
<b>Price per Month</b>	\$3,500	\$6,000	\$10,000

## Consulting

While Matt is available for long term consulting engagements, you might find it useful to consider our 2-week **QuickStart Engagement**. This includes starting your application with AppFuse, setting up developer's environments (and documenting for new developers) as well as setting up your Development Ecosystem (DE). Development Ecosystems typically consist of IDEs, Source Control, an Issue Tracker and Wiki (for documentation, Continuous Integration and Automated Releases. If you're keen on using open source, the following stack works quite well for ecosystems:

- Eclipse
- Subversion
- Trac
- Hudson
- Maven 2

If you're open to commercial tools, we highly recommend [IntelliJ IDEA](#) and [Atlassian](#) products. This stack includes:

- IntelliJ IDEA
- Subversion
- JIRA/Confluence
- Bamboo
- Maven 2

The price for a 2 week **QuickStart Engagement** is \$15,000 and includes two days (first and last) onsite.

## Training

Raible Designs offers a training course titled **Open Source Java Jewels**. This course is delivered in two formats: one for Managers and one for Developers (with labs). The Manager course is 1.5 days while the Developer course is 3 days. Prices are listed at the top right of each course's agenda.

The instructor will provide a full development environment on an Ubuntu VMWare image. Instructions will also be provided to setup Windows and OS X. Minimum hardware requirements are 2GB RAM, Windows XP/Mac/Linux and 5 GB of disk space. Internet access should be provided for students as well. The course will be delivered onsite.

### Open Source Java Jewels for Managers

\$5000 (1-4 students) / \$10,000 (up to 12 students)

Day 1: Spring, Hibernate, Maven and Spring MVC

- [9:00] Introductions and Overview
- [9:30] Spring Framework Overview
- [11:00] Introduction to Hibernate and JPA
- *Lunch Break*
- [1:00] Introduction to Maven 2
- [2:00] Introduction to Spring MVC
- [3:00] Developing web applications with Ajax and DWR

#### Day 2: Testing, Ajax and AppFuse

- [9:00] Testing Spring Applications
- [10:00] UI Testing with Canoo WebTest, JWebUnit and Selenium
- [11:00] Introduction to AppFuse 2.0

## Open Source Java Jewels for Developers

\$7000 (1-4 students) / \$15,000 (up to 12 students)

#### Day 1: Spring and Hibernate

- [9:00] Introductions and Overview
- [9:30] Presentation: Spring Framework Overview
- [10:30] Lab: Loading the BeanFactory and using IoC
- [11:00] Presentation: The BeanFactory in-depth
- *Lunch Break*
- [1:00] Lab: Writing a custom PropertyEditor
- [2:00] Presentation: Introduction to Hibernate and JPA
- [3:00] Lab: Persistence with Spring and Hibernate
- [4:00] Presentation: Spring Persistence, DAOs and Caching

#### Day 2: Maven, Testing and Spring MVC

- [9:00] Presentation: Introduction to Maven 2
- [10:00] Presentation: Testing Spring Applications
- [11:00] Lab: Testing with JUnit, jMock and TestNG
- *Lunch Break*
- [1:00] Presentation: Introduction to Spring MVC
- [2:00] Lab: Create a Master/Detail Screen
- [4:00] Presentation: Advanced Spring MVC

#### Day 3: Ajax, AppFuse and UI Testing

- [9:00] Lab: Validation and Handling Dates with Spring MVC
- [10:00] Developing web applications with Ajax and DWR
- [11:00] Lab: Master/Detail Ajaxified and Autocomplete
- *Lunch Break*
- [1:00] Presentation: Introduction to AppFuse 2.0
- [2:00] Lab: Creating Projects with AppFuse
- [2:30] Presentation: Maven Plugins and AMP (AppFuse Maven Plugin)
- [3:00] Lab: Using AMP to generate a table's UI
- [4:00] Presentation: UI Testing with Canoo WebTest, JWebUnit and Selenium



#### Additional Modules

In addition to the sessions listed here, we also have presentations and labs covering Spring AOP, Spring Transactions, Acegi Security and Remoting/Web Services with Spring. You can also replace any references to Spring MVC with Struts 2, JSF or Tapestry.

## Contact

Please [contact Raible Designs](#) if you have any questions.

## Demos and Videos

There are online demos of AppFuse as well as a few videos. In addition, there are a number of public sites [powered by AppFuse](#).

## Online Demos

Use username **admin** and password **admin** to login as an administrator - **user/user** will allow you to login as a user. Click on the links below to

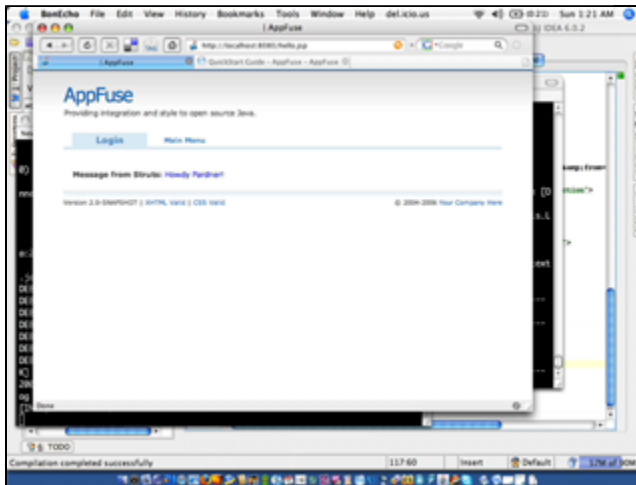


view a live demo.

- JSF + Spring + Hibernate
- Struts 2 + Spring + Hibernate
- Spring MVC + Spring + Hibernate
- Tapestry + Spring + Hibernate

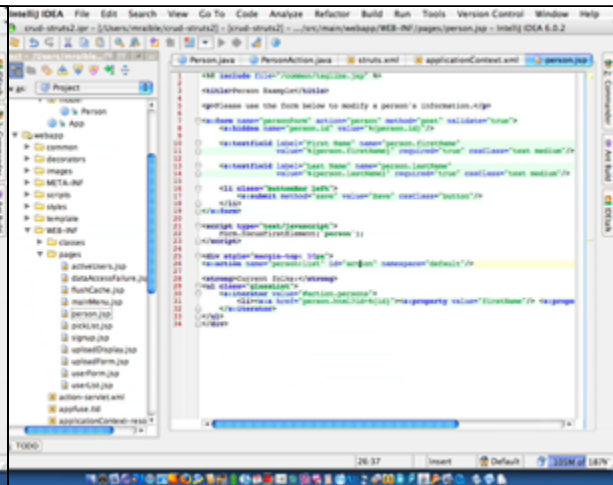
## Videos

### Hello World



Watch QuickTime Video  
13.2 MB, 6 minutes 14 seconds

### Easy CRUD with Struts 2



Watch QuickTime Video  
14.2 MB, 6 minutes 6 seconds

Here you can download high quality videos (they're much clearer) 50.88 MB:  
<https://appfuse.dev.java.net/files/documents/1397/58384/appfuse-2.0-videos.zip>

The projects created in these videos can be downloaded from the [appfuse-demos](#) project.

## FAQ

This page is a growing list of frequently asked questions about AppFuse.

### Table of Contents

- What is AppFuse?
- What are AppFuse's conventions for package naming?
- What is the difference between AppFuse 1.x and 2.x?
- Where are the Java source files and JSPs in 2.x?
- Can I use Ant instead of Maven in 2.x?
- Can I run AppFuse 2.x using Java SE 1.4?
- How do I customize the header and footer messages in 2.x?
- How do I customize Login and Error Pages in 2.x?
- Why don't I have to install Tomcat with 2.x?
- What if I already have Tomcat installed and I'd like to use it?
- What does "basic" and "modular" mean when it comes to AppFuse 2.x archetypes?
- How do I tell IDEA to ignore deployment descriptors when compiling?
- How can I export my database using DbUnit?
- How can I change my AppFuse-generated project to use iBATIS or JPA?
- Is there any documentation that describes what all the filters do in web.xml?
- How do I configure LDAP Authentication?
- How do I debug my application when it's running in Jetty?
- How do I make Jetty start on a specific port?
- When saving Chinese or Japanese characters in a form, I'm getting java.sql.SQLException: Incorrect string value. How do I solve this?
- When using jetty:run, CSS and JavaScript files are locked - how can I fix this?
- Using Struts, how can I dynamically add columns to a Table using a list of beans nested in my table row data?

## What is AppFuse?

AppFuse is an open source project and application that uses open source tools built on the Java platform to help you develop Web applications quickly and efficiently. It was originally developed to eliminate the ramp-up time found when building new web applications for customers. At its core, AppFuse is a project skeleton, similar to the one that's created by your IDE when you click through a wizard to create a new web project.

AppFuse 1.x uses Ant to create your project, as well as build/test/deploy it. AppFuse 2.x uses Maven 2 to create your project as well as build/test/deploy it. IDE support is much better in 2.x because you can generate the IDE project files with Maven plugins. AppFuse 1.x uses XDoclet and JDK 1.4+. AppFuse 2.x uses annotations and JDK 5+.

## What are AppFuse's conventions for package naming?

It's recommended you start with `com.companyname` if you're commercial or `org.organizationname` if you're non-profit. Most folks match their internet domain name. From there, use your application name.

From there, AppFuse expects you to use the following conventions:

```
com.company.app.model -> Entities
com.company.app.dao -> Dao Interfaces
com.company.app.dao.hibernate -> Hibernate Implementations
com.company.app.service -> Service Interfaces
com.company.app.service.impl -> Service implementation
com.company.app.webapp.action -> Struts Actions (this really depends on the framework you're using)
```

## What is the difference between AppFuse 1.x and 2.x?

AppFuse 2.0 works rather differently from AppFuse 1.x - Maven keeps the AppFuse core classes as dependencies. You add your code, and Maven will merge in the code from AppFuse. This should make it much easier for you to upgrade to future releases of AppFuse.

If you run `mvn jetty:run-war` and point your browser to <http://localhost:8080/> you will see that you have a fully fledged AppFuse project present. If you need to change any of the files in the web project, simply run `mvn war:inplace` in the web project to populate the `src/main/webapp` source directory. You will need to delete `WEB-INF/lib` and `WEB-INF/classes/struts.xml` (if you're using Struts) after doing this.

## Where are the Java source files and JSPs in 2.x?

AppFuse 2.x is a fair amount different from 1.x in that you don't get all the source to start your project with. The main reason for this is to make it possible for you to upgrade your application to a newer version of AppFuse. For a more detailed explanation, please see [this mailing list thread](#).

If you'd like to modify AppFuse's core entity classes, see [how to modify core classes](#). If you'd like to convert your project so it doesn't rely on AppFuse dependencies and includes AppFuse's source instead, run `mvn appfuse:full-source`.

## Can I use Ant instead of Maven in 2.x?

If you create a project using one of the basic archetypes in M5 and then run `mvn appfuse:full-source`, you can use a modified version of AppFuse Light's `build.xml` to build/test/deploy your project. Below are the steps needed to add Ant support to a basic archetype:

- Run `mvn appfuse:full-source` from your project's directory
- Download `build.xml` into your project.
- Edit `build.xml` to replace occurrences of "appfuse-light" with your project's name.
- Download `maven-ant-tasks-2.0.9.jar` and copy it to a "lib" directory in your project. You will need to create this directory if it doesn't already exist. You can also add this JAR to your `$ANT_HOME/lib` directory.

We don't plan on supporting Ant for modular archetypes. If someone wants to contribute Ant support for them and support it, we'll be happy to add it.

## Can I run AppFuse 2.x using Java SE 1.4?

AppFuse 2.x requires Java 5 for development, but not for deployment. You should be able to use the [Retrotranslator Plugin](#) to convert your WAR to be 1.4-compliant.

## How do I customize the header and footer messages in 2.x?

If you would like change the following heading:

### AppFuse

Providing integration and style to open source Java

Change `src/main/resources/ApplicationResources.properties` and change values for the `webapp.name` and `webapp.tagline` keys. To modify

the "Your Company Here" text and its URL in the footer, you'll need to modify the `company.name` and `company.url` in this same file.

## How do I customize Login and Error Pages in 2.x

If you would like change the any of the pages that appear in the root directory of your application, for example:

**login.jsp, error.jsp, 404.jsp, logout.jsp**

Put copies of your modified jsp files into `src/main/webapp`.

If you want to copy existing versions of these files and modify them, look for them in your `target/YOUR_APP_NAME/` directory.

## Why don't I have to install Tomcat with 2.x?

If you run **mvn integration-test**, the Maven Cargo plugin will download and install Tomcat and run the integration tests against it. If you want to see your project running in a server, you have two options:

1. Run **mvn jetty:run-war** - this will fire up a Jetty instance and it will be available at <http://localhost:8080>.
2. Run **mvn cargo:start -Dcargo.wait=true** and your application will be deployed to Tomcat and available at <http://localhost:8081/applicationname-version>.

You can also add the [Maven 2 Tomcat Plugin](#) to your `pom.xml` and use **mvn tomcat:run** and **mvn tomcat:run-war**.

```
<plugin>
  <groupId>org.codehaus.mojo</groupId>
  <artifactId>tomcat-maven-plugin</artifactId>
  <configuration>
    <path>/</path>
  </configuration>
</plugin>
```

## What if I already have Tomcat installed and I'd like to use it?

If you want to change [Cargo](#) to point to an existing instance of Tomcat, you should be able to change the bottom of `pom.xml` from:

```
<container>
  <containerId>${cargo.container}</containerId>
  <!--home>${cargo.container.home}</home-->
  <zipUrlInstaller>
    <url>${cargo.container.url}</url>
    <installDir>${installDir}</installDir>
  </zipUrlInstaller>
</container>
```

To:

```
<container>
  <containerId>${cargo.container}</containerId>
  <home>${cargo.container.home}</home>
</container>
```

You can also deploy to a running instance of Tomcat with Cargo.

## What does "basic" and "modular" mean when it comes to AppFuse 2.x archetypes?

Basic archetypes are meant for developing web applications with no re-usable backend. This is the recommended archetype to use with AppFuse 2.x. The [Tutorials](#) assume you're using this archetype.

Unlike Ant, Maven requires you to have a different "module" for each artifact you want to produce (i.e. JAR or WAR). This is what the modular archetypes are for. If you're an experienced Maven 2 user, you might consider this archetype. It's also useful if you need a backend that can be

used by multiple clients.

## How do I tell IDEA to ignore deployment descriptors when compiling?

Right-click on the file in the messages window and choose "Exclude From Compile".

## How can I export my database using DbUnit?

Running `mvn dbunit:export` will export your database to `target/dbunit/export.xml`. You can override the destination location by passing in a `-Ddest` parameter:

```
mvn dbunit:export -Ddest=sample-data.xml
```

## How can I change my AppFuse-generated project to use iBATIS or JPA?

As of AppFuse 2.0 M5+, it's as easy as changing your changing the `<dao.framework>` property in your root `pom.xml` to **ibatis** or **jpa** (**jpa-hibernate** in 2.0 M5). For further instructions, please see the [JPA](#) and [iBATIS](#) tutorials.

## Is there any documentation that describes what all the filters do in web.xml?

See the [Web Filters Reference Guide](#) for a list of what each filter in `web.xml` is used for.

## How do I configure LDAP Authentication?

See the [LDAP Authentication](#) page for a sample configuration.

## How do I debug my application when it's running in Jetty?

Dan Allen has a nice writeup of how to do [remoting debugging with Jetty](#).

## How do I make Jetty start on a specific port?

There are two easy methods depending on whether you want the change to be permanent (i.e. each time you run 'mvn jetty:run-war' or a manual port change.

Permanent: In your project `pom.xml` look for the plugin with the artifactid 'maven-jetty-plugin' and add the `<connectors>` elements to it:

```
<plugin>
  <groupId>org.mortbay.jetty</groupId>
  <artifactId>maven-jetty-plugin</artifactId>
  <version>6.1.5</version>
  <configuration>
    ...
    <connectors>
      <connector implementation="org.mortbay.jetty.nio.SelectChannelConnector">
        <port>8081</port>
      </connector>
    </connectors>
  </configuration>
</plugin>
```

Manual: run 'mvn jetty:run-war -Djetty.port=8081'

## When saving Chinese or Japanese characters in a form, I'm getting java.sql.SQLException: Incorrect string value. How do I solve this?

To solve this problem, you'll need to drop and recreate your database with UTF-8 enabled. The following example shows how to do this with a database named **myproject**:

```
mysqladmin -u root -p drop myproject
mysqladmin -u root -p create myproject
mysql -u root -p
<enter password>
mysql> ALTER DATABASE myproject CHARACTER SET utf8 DEFAULT CHARACTER SET utf8 COLLATE utf8_general_ci
mysql> exit
```

You can also configure MySQL to use UTF-8 by default by adding the following to your c:\Windows\my.ini or /etc/my.cnf (on Unix) file:

```
[mysqld]
default-character-set=utf8
```

## When using jetty:run, CSS and JavaScript files are locked - how can I fix this?

Jetty buffers static content for webapps such as html files, css files, images etc and uses memory mapped files to do this if the NIO connectors are being used. The problem is that on Windows, memory mapping a file causes the file to be locked, so that the file cannot be updated or replaced. This means that effectively you have to stop Jetty in order to update a file.

The problem is explained more in [Jetty's documentation](#). To fix this, add a line with `<webDefaultXml>` to your maven-jetty-plugin configuration:

```
<groupId>org.mortbay.jetty</groupId>
<artifactId>maven-jetty-plugin</artifactId>
<version>6.1.5</version>
<configuration>
  ...
  <webDefaultXml>src/main/resources/webdefault.xml</webDefaultXml>
</configuration>
```

The default webdefault.xml file is found in the lib/jetty.jar at org/mortbay/jetty/webapp/webdefault.xml. Extract it to a convenient disk location and edit it to change useFileMappedBuffer to false:

```
<init-param>
  <param-name>useFileMappedBuffer</param-name>
  <param-value>false</param-value>
</init-param>
```

## Using Struts, how can I dynamically add columns to a Table using a list of beans nested in my table row data?

This is the scenario. You are presenting a list of your data on a standard mymodelList.jsp form, using the [Display tag library](#). Your data bean has an attribute that is a list of related data beans. In this scenario, you wish to display that related data in extra columns in your table. Here is one way to achieve this.

The following code example assumes that your Action class has generated a list of parentBean objects and exposes that with a getParentBeans() method that returns a List<ParentBean>. The ParentBean object includes an attribute that is a list of ChildBean objects. Our simple beans have the following structure:

ParentBean
<ul style="list-style-type: none"><li>• String: id</li><li>• String: title</li><li>• List&lt;ChildBean&gt; childBeans</li></ul>

ChildBean
<ul style="list-style-type: none"> <li>• String: id</li> <li>• String: name</li> <li>• Date: lastUpdated</li> </ul>

... and the markup goes something like this...

parentBeanList.jsp
<pre> &lt;!-- Normal page header and footer stuff excluded for brevity --&gt; &lt;display:table name="parentBeans" class="table" requestURI="" id="parentBeans" export="true" pagesize="25"&gt;   &lt;!-- this is a normal column displaying the "ParentBean.title" attribute --&gt;   &lt;display:column property="title" sortable="true" href="editParentBean.html"     media="html" paramId="id" paramProperty="id" titleKey="parentBean.title"/&gt;   &lt;display:column property="title" media="csv excel xml pdf" titleKey="parentBean.title"/&gt;    &lt;!-- Iterate through the childBeans list if present and display a name and lastUpdated column for   each --&gt;   &lt;!-- We gain access to our parentBean's childBeans list using the #attr.parentBeans.childBeans   reference --&gt;   &lt;s:if test="#attr.parentBeans.childBeans != null &amp;&amp; #attr.parentBeans.childBeans.size() &gt; 0"&gt;     &lt;!-- IMPORTANT NOTE - in this example, the List is actually implemented by a TreeMap (to     facilitate sorting) so the     reference is to the list's "values()" method, otherwise the iterator will return a Map     object (containing     a key and your bean) rather than just the bean. If your childBeans list is a normal List     (eg ArrayList)     you'd simply put 'value="#attr.parentBeans.childBeans"' in the next line --&gt;     &lt;s:iterator id="childBean" value="#attr.parentBeans.childBeans.values()" status="col"&gt;       &lt;!-- Now, complete the display:column object. --&gt;       &lt;display:column sortable="false" titleKey="childbean.name"&gt;         &lt;!-- Note, the reference to \${id/ } in the next row is obtained by using the id         attribute value from         the &lt;display:table tag (see above). In this example, we edit a childBean by         going to the         editor for its ParentBean --&gt;         &lt;a href="editParentBean.html?id=\${parentBeans.id/ }"&gt;           \${childBean.name/ }         &lt;/a&gt;       &lt;/display:column&gt;       &lt;display:column sortable="false" titleKey="childbean.lastUpdated"&gt;         &lt;fmt:formatDate value="\${lastUpdated/ }" pattern="\${timePattern/ }"/&gt;       &lt;/display:column&gt;     &lt;/s:iterator&gt;   &lt;/s:if&gt;    &lt;!-- Paging and export properties excluded for brevity --&gt; &lt;/display:table&gt; </pre>

## News

See [Articles](#), [Blogs](#), [Roadmap](#) and [Presentations](#) for more AppFuse propaganda.



Monday, April 4, 2011

AppFuse 2.1 Released!

Last changed Apr 04, 2011 10:10 by [Matt Raible](#)

Labels: [release](#)

The AppFuse Team is pleased to announce the release of AppFuse 2.1. This release includes upgrades to all dependencies to bring them up-to-date with their latest releases. Most notable are JPA 2, JSF 2, Tapestry 5 and Spring 3. In addition, we've migrated from XFire to CXF and

enabled REST for web services. There's even a new **appfuse-ws** archetype that leverages Enunciate to generate web service endpoints, documentation and downloadable clients. This release fixes many issues with archetypes, improving startup time and allowing `jetty:run` to be used for quick turnaround while developing. For more details on specific changes see the [release notes](#).



#### What is AppFuse?

AppFuse is an open source project and application that uses open source frameworks to help you develop Web applications with Java quickly and efficiently. It was originally developed to eliminate the ramp-up time when building new web applications. At its core, AppFuse is a project skeleton, similar to the one that's created by your IDE when you click through a wizard to create a new web project. If you use [JRebel with IntelliJ](#), you can achieve zero-turnaround in your project and develop features without restarting the server.

### Get Started Now »

#### Release Details

[Archetypes](#) now include all the source for the web modules so using `jetty:run` and your IDE will work much smoother now. The backend is still embedded in JARs, enabling you to choose with persistence framework (Hibernate, iBATIS or JPA) you'd like to use. If you want to modify the source for that, [add the core classes to your project](#) or run `"appfuse:full-source"`.

AppFuse comes in a number of different flavors. It offers "light", "basic" and "modular" and archetypes. Light archetypes use an embedded H2 database and contain a simple CRUD example. Light archetypes allow code generation and full-source features, but do not currently support Stripes or Wicket. Basic archetypes have web services using CXF, authentication from Spring Security and features including signup, login, file upload and CSS theming. Modular archetypes are similar to basic archetypes, except they have multiple modules which allows you to separate your services from your web project.

AppFuse provides archetypes for JSF, Spring MVC, Struts 2 and Tapestry 5. The light archetypes are available for these frameworks, as well as for Spring MVC + FreeMarker, Stripes and Wicket. You can see demos of these archetypes at <http://demo.appfuse.org>.

For information on creating a new project, please see the [QuickStart Guide](#).

If you have questions about AppFuse, please read the [FAQ](#) or join the [user mailing list](#). If you find any issues, please report them on the mailing list or [create an issue in JIRA](#).

Thanks to everyone for their help contributing patches, writing documentation and participating on the mailing lists.

We greatly appreciate the help from [our sponsors](#), particularly [Atlassian](#), [Contegix](#), [JetBrains](#), and [Java.net](#). Atlassian and Contegix are especially awesome: [Atlassian has donated licenses to all its products](#) and [Contegix has donated an entire server](#) to the AppFuse project.

Posted at Apr 04, 2011 by  [Matt Raible](#) |  0 comments | [Edit](#)



Monday, November 15, 2010

AppFuse 2.1.0 Milestone 2 Released

Last changed Nov 15, 2010 16:39 by [Matt Raible](#)

Labels: [release](#)

The AppFuse Team is pleased to announce the 2nd milestone release of AppFuse 2.1. This release includes upgrades to all dependencies to bring them up-to-date with their latest releases. Most notable are Spring 3 and Struts 2.1. This release fixes many issues with archetypes and contains many improvements to support Maven 3. For more details on specific changes see the [2.1.0 M2 release notes](#).



#### What is AppFuse?

AppFuse is an open source project and application that uses open source frameworks to help you develop Web applications quickly and efficiently. It was originally developed to eliminate the ramp-up time when building new web applications. At its core, AppFuse is a project skeleton, similar to the one that's created by your IDE when you click through a wizard to create a new web project. If you use [JRebel with AppFuse](#), you can achieve zero-turnaround in your project and develop features without restarting the server.

### Get Started Now »

#### Release Details

[Archetypes](#) now include all the source for the web modules so using `jetty:run` and your IDE will work much smoother now. The backend is still embedded in JARs, enabling you to choose with persistence framework (Hibernate, iBATIS or JPA) you'd like to use. If you want to modify the source for that, [add the core classes to your project](#) or run `"appfuse:full-source"`.

AppFuse comes in a number of different flavors. It offers "light", "basic" and "modular" and archetypes. Light archetypes use an embedded H2 database and contain a simple CRUD example. In the final 2.1.0 release, the light archetypes will allow code generation like the basic and modular archetypes. Basic archetypes have web services using CXF, authentication from Spring Security and features including signup, login, file upload and CSS theming. Modular archetypes are similar to basic archetypes, except they have multiple modules which allows you to separate your services from your web project.



AppFuse provides archetypes for JSF, Spring MVC, Struts 2 and Tapestry 5. The light archetypes are available for these frameworks, as well as for Spring MVC + FreeMarker, Stripes and Wicket.


Please note that this release does not contain updates to the documentation. Code generation will work, but it's likely that some content in the [tutorials](#) won't match. For example, you can use annotations (vs. XML) for Spring MVC and Tapestry is a whole new framework. I'll be working on documentation over the next several weeks in preparation for the 2.1 final release.

For information on creating a new project, please see the [QuickStart Guide](#).

If you have questions about AppFuse, please read the [FAQ](#) or join the [user mailing list](#). If you find bugs, please [create an issue in JIRA](#).

Thanks to everyone for their help contributing patches, writing documentation and participating on the mailing lists.

Posted at Nov 15, 2010 by  [Matt Raible](#) |  0 comments | [Edit](#)

 Thursday, November 19, 2009  
[AppFuse 2.1.0 Milestone 1 Released](#)  
Last changed Nov 19, 2009 22:42 by [Matt Raible](#)  
Labels: [release](#)

The AppFuse Team is pleased to announce the first milestone release of AppFuse 2.1. This release includes upgrades to all dependencies to bring them up-to-date with their latest releases. Most notable are [Hibernate](#), [Spring](#) and Tapestry 5.



#### What is AppFuse?

AppFuse is an open source project and application that uses open source tools built on the Java platform to help you develop Web applications quickly and efficiently. It was originally developed to eliminate the ramp-up time found when building new web applications for customers. At its core, AppFuse is a project skeleton, similar to the one that's created by your IDE when you click through a wizard to create a new web project.

### Get Started Now »

#### Release Details

[Archetypes](#) now include all the source for the web modules so using `jetty:run` and your IDE will work much smoother now. The backend is still embedded in JARs, enabling you to choose with persistence framework (Hibernate, iBATIS or JPA) you'd like to use. If you want to modify the source for that, [add the core classes to your project](#) or run `"appfuse:full-source"`.

In addition, AppFuse Light has been [converted to Maven](#) and has archetypes available too. AppFuse provides archetypes for JSF, Spring MVC, Struts 2 and Tapestry 5. The *light* archetypes are available for these frameworks, as well as for Spring MVC + FreeMarker, Stripes and Wicket.

Other notable improvements:

- Added [Compass support](#) thanks to a patch from [Shay Banon](#).
- Upgraded from XFire to CXF for Web Services.
- Moved Maven repository to [Sonatype's OSS Repository Hosting](#) for snapshots and releasing to Maven Central. There are no longer any AppFuse-specific artifacts, all are available in central. Thanks to [Sonatype](#) for this great service and its [excellent repository manager](#).
- Upgraded to Canoo WebTest 3.0. *Now if we could just get its [Maven Plugin](#) moved to Codehaus.*
- Added [Ajaxified Body](#) to AppFuse Light archetypes.
- Infrastructure upgrades, including [JIRA 4](#), [Confluence 3](#), [FishEye 2](#), [Bamboo 2](#) and [Crowd 1.6](#). Many thanks to [Atlassian](#) and [Contegix](#) for their excellent products and services.
- For more details on specific changes see the [release notes](#).

Please note that this release does not contain updates to the documentation. Code generation will work, but it's likely that some content in the [tutorials](#) won't match. For example, you can use annotations (vs. XML) for dependency injection and Tapestry is a whole new framework. I'll be working on documentation over the next several weeks in preparation for Milestone 2.

AppFuse is available as several Maven archetypes. For information on creating a new project, please see the [QuickStart Guide](#).



To learn more about AppFuse, please read Ryan Withers' [Igniting your applications with AppFuse](#).

The 2.x series of AppFuse has a minimum requirement of the following specification versions:


- Java Servlet 2.4 and JSP 2.0 (2.1 for JSF)
- Java 5+

If you have questions about AppFuse, please read the [FAQ](#) or join the [user mailing list](#). If you find bugs, please [create an issue in JIRA](#).

Thanks to everyone for their help contributing code, writing documentation, posting to the mailing lists, and logging issues.



Posted at Nov 19, 2009 by  Matt Raible |  0 comments | [Edit](#)

 Thursday, November 5, 2009  
[A Letter to the AppFuse Community](#)  
Labels: [status](#)

I wrote a blog post titled "A Letter to the AppFuse Community" tonight to let y'all know what's going on with the AppFuse project and my thoughts on its future. Don't worry, it's mostly good stuff. You can find it at the following URL:

[http://raibledesigns.com/rd/entry/a\\_letter\\_to\\_the\\_appfuse](http://raibledesigns.com/rd/entry/a_letter_to_the_appfuse)

Thanks for your support,

Matt

Posted at Nov 05, 2009 by  Matt Raible |  0 comments | [Edit](#)


 Thursday, May 22, 2008  
[New Tutorials - External Sorting with Display Tag and Integrating Compass 2.0](#)

Chris Barham has been a busy guy lately. Not only has he written a tutorial on integrating external sorting and pagination with the Display Tag, but he's also written a tutorial on how to integrate Compass 2.0 into AppFuse. Check out the links below for more information:

- [AppFuse + DisplayTag: External Sorting and Paging Example](#)
- [Integrating Compass 2.0 into AppFuse](#)


Awesome work Chris.

Posted at May 22, 2008 by  Matt Raible |  0 comments | [Edit](#)

 Tuesday, May 13, 2008  
[New Documentation!](#)

If you're looking for an excellent article explaining what AppFuse is, please see Ryan Withers' [Igniting your applications with AppFuse](#). For in-depth coverage of AppFuse and all its supporting frameworks, there's no better place to go than David Whitehurst's [AppFuse Primer](#). Thanks to both Ryan and David for their contributions to this project.

Posted at May 13, 2008 by  Matt Raible |  0 comments | [Edit](#)

 Sunday, May 11, 2008  
[AppFuse 2.0.2 Released](#)  
Last changed May 11, 2008 23:41 by [Matt Raible](#)  
Labels: [release](#), [appfuse](#)

The AppFuse Team is pleased to announce the release of AppFuse 2.0.2. This release includes upgrades to Spring Security 2.0, jMock 2.4, the ability to customize code generation templates and many bug fixes.

For information on upgrading from 2.0.1, see the [Release Notes](#) or [changelog](#). AppFuse 2.0.2 is available as a Maven archetype. For information on creating a new project using AppFuse, please see the [QuickStart Guide](#) or the [Demos and Videos](#).

The 2.0 series of AppFuse has a minimum requirement of the following specification versions:


- Java Servlet 2.4 and JSP 2.0
- Java 5+

If you've used AppFuse 1.x, but not 2.x, you'll want to read the [FAQ](#). Join the [user mailing list](#) if you have any questions. The [Maven Reference Guide](#) has a map of Ant -> Maven commands. [Maven for Newbies](#) might also be useful if you've never used Maven before.

Thanks to everyone for their help contributing code, writing documentation, posting to the mailing lists, and logging issues.

We greatly appreciate the help from [our sponsors](#), particularly [Atlassian](#), [Contegix](#), [JetBrains](#), and [Java.net](#). Atlassian and Contegix are especially awesome: [Atlassian has donated licenses to all its products](#) and [Contegix has donated an entire server](#) to the AppFuse project.

Posted at May 11, 2008 by  Matt Raible |  0 comments | [Edit](#)

 Wednesday, December 5, 2007  
MarkMail and FishEye

We've recently added some new features to the AppFuse project. The first is we've installed FishEye and Crucible (a code review tool) on AppFuse's server at Contegix. We've used FishEye [in the past](#), but now we have it on our own servers. You can view it at:

<http://source.appfuse.org>

The 2nd news item is that the good folks at [Mark Logic](#) (primarily Jason Hunter) has setup a kick-ass mailing list archive for us at:

<http://appfuse.markmail.org>


This is an easily searchable set of archives and goes all the way back to when our mailing list started back in March 2004. In the future, they hope to provide RSS/Atom feeds and allow posting (like Nabble does). I've added a link to these archives to the [mailing list](#) page.

Posted at Dec 05, 2007 by  [Matt Raible](#) |  0 comments | [Edit](#)

 Wednesday, November 28, 2007  
[Integrating Crowd with AppFuse and Acegi](#)  
Last changed Nov 28, 2007 12:22 by [Matt Raible](#)

The good folks at Atlassian have written a nice tutorial on [integrating Crowd with AppFuse and Acegi](#) to create a nice SSO solution. Crowd is a web-based single sign-on (SSO) tool that simplifies application provisioning and identity management. The AppFuse project uses it to maintain a single user store for JIRA, Confluence (this site), and Bamboo. It's worked *awesome* for us.

Posted at Nov 28, 2007 by  [Matt Raible](#) |  0 comments | [Edit](#)

 Monday, November 26, 2007  
[AppFuse 2.0.1 Released](#)  
Last changed Nov 26, 2007 10:19 by [Matt Raible](#)

The AppFuse Team is pleased to announce the release of AppFuse 2.0.1. This release squashes a number of bugs and includes an upgrade to Spring 2.5. To learn more about Spring 2.5's features, see InfoQ's [What's New in Spring 2.5: Part 1](#) article.

For information on upgrading from 2.0, see the [2.0.1 Release Notes](#) or [changelog](#). AppFuse 2.0.1 is available as a Maven archetype. For information on creating a new project using AppFuse, please see the [QuickStart Guide](#) or the [Demos and Videos](#).

The 2.0 series of AppFuse has a minimum requirement of the following specification versions:


- Java Servlet 2.4 and JSP 2.0 (2.1 for JSF)
- Java 5+

If you've used AppFuse 1.x, but not 2.x, you'll want to read the [FAQ](#). Join the [user mailing list](#) if you have any questions. The [Maven Reference Guide](#) has a map of Ant -> Maven commands. [Maven for Newbies](#) might also be useful if you've never used Maven before. There is [some support for Ant](#) in this release.

Thanks to everyone for their help contributing code, writing documentation, posting to the mailing lists, and logging issues.

We greatly appreciate the help from [our sponsors](#), particularly [Atlassian](#), [Contegix](#), [JetBrains](#), and [Java.net](#). Atlassian and Contegix are especially awesome: [Atlassian has donated licenses to all its products](#) and [Contegix has donated an entire server](#) to the AppFuse project.

Posted at Nov 26, 2007 by  [Matt Raible](#) |  0 comments | [Edit](#)

 Thursday, September 20, 2007  
[Thanks goodness for free bandwidth](#)  
Last changed Sep 20, 2007 11:24 by [Matt Raible](#)

It's a good thing this project gets [free bandwidth from Contegix](#)! Looking at September's stats for [static.appfuse.org](#) (the site that hosts our Maven repo), we're averaging 2.15 GB per day. *Thanks Contegix!*

Day	Number of visits	Pages	Hits	Bandwidth
01 Sep 2007	130	357	1125	1.34 GB
02 Sep 2007	150	382	1422	1.00 GB
03 Sep 2007	250	683	2383	1.77 GB
04 Sep 2007	359	1215	3833	2.58 GB
05 Sep 2007	303	1413	3728	2.10 GB
06 Sep 2007	285	1369	3853	1.75 GB
07 Sep 2007	275	58258	60075	2.83 GB
08 Sep 2007	151	401	1198	1.15 GB
09 Sep 2007	156	322	1020	1.18 GB
10 Sep 2007	315	857	2903	2.63 GB
11 Sep 2007	307	846	2751	2.49 GB
12 Sep 2007	308	952	2649	2.71 GB
13 Sep 2007	310	909	2863	2.44 GB
14 Sep 2007	318	1163	3107	2.25 GB
15 Sep 2007	182	311	1036	1.15 GB
16 Sep 2007	178	446	1551	1.17 GB
17 Sep 2007	344	899	2900	2.55 GB
18 Sep 2007	395	19705	21656	2.99 GB
19 Sep 2007	497	3332	5400	4.45 GB
<b>20 Sep 2007</b>	277	1804	2787	2.38 GB

Posted at Sep 20, 2007 by  Matt Raible |  0 comments | [Edit](#)



Tuesday, September 18, 2007

AppFuse 2.0 Released!

Last changed Sep 18, 2007 16:08 by [Matt Raible](#)

The AppFuse Team is pleased to announce the release of AppFuse 2.0!

The road to AppFuse 2.0 has been a long journey through Mavenland, annotations and generics. We're pleased to announce that we're finally finished after 13 months of development. Thanks to all the developers, contributors and users for helping test, polish and prove that AppFuse 2 is an excellent solution for developing Java-based applications. Your time, patience and usage of AppFuse has made it the strong foundation it is today.

This release contains a number of bug fixes for AMP, an upgrade to Tapestry 4.1.3, the addition of Tacos, support for Oracle and changes to prevent XSS attacks.

AppFuse 2.0 is available as a Maven archetype. For information on creating a new project using AppFuse, please see the [QuickStart Guide](#) or the [Demos and Videos](#).

If you've used AppFuse 1.x, but not 2.x, you'll want to read the [FAQ](#). Join the [user mailing list](#) if you have any questions. The [Maven Reference Guide](#) has a map of Ant -> Maven commands. [Maven for Newbies](#) might also be useful if you've never used Maven before. There is [some support for Ant](#) in this release.

For more information, please see the [2.0 Release Notes](#). The 2.0 series of AppFuse has a minimum requirement of the following specification versions:

- Java Servlet 2.4 and JSP 2.0 (2.1 for JSF)
- Java 5+

New features in AppFuse 2.0 include:

- Maven 2 Integration
- Upgraded WebWork to Struts 2
- JDK 5, Annotations, JSP 2.0, Servlet 2.4
- JPA Support
- Generic CRUD backend
- Full Eclipse, IDEA and NetBeans support
- Fast startup and no deploy with Maven Jetty Plugin
- Testable on multiple appservers and databases with Cargo and profiles

We appreciate the time and effort everyone has put toward contributing code and documentation, posting to the mailing lists, and logging issues.

We also greatly appreciate the help from our sponsors, particularly [Atlassian](#), [Contegix](#), [JetBrains](#), and [Java.net](#). Atlassian and Contegix are especially awesome: [Atlassian has donated licenses to all its products](#) and [Contegix has donated an entire server](#) to the AppFuse project.

Posted at Sep 18, 2007 by  Matt Raible |  0 comments | [Edit](#)

 Tuesday, September 4, 2007

AppFuse 2.0 RC1 Released

Last changed Sep 05, 2007 10:38 by [Matt Raible](#)

Labels: [appfuse](#), [release](#)

The AppFuse Team is pleased to announce the release of AppFuse 2.0 RC1!

This release marks a huge step in the march to releasing AppFuse 2.0. This release puts the finishing touches on the [AppFuse Maven Plugin](#) (AMP), which offers CRUD generation, as well as the ability to change AppFuse from "embedded mode" to "full source" (like 1.x). In addition, we've addressed over 100 issues in preparation for the final 2.0 release. We hope to fix any bugs related to this release and release 2.0 Final in the next week or two.

The [videos](#) still represent how M5 works, but things have been simplified (now you don't need to run `appfuse:install` after `appfuse:gen`).

AppFuse 2.0 is available as a Maven archetype. For information on creating a new project using this release, please see the [QuickStart Guide](#) or the [Hello World video](#).

If you've used AppFuse 1.x, but not 2.x, you'll want to read the [FAQ](#). Join the [user mailing list](#) if you have any questions. The [Maven Reference Guide](#) has a map of Ant -> Maven commands. [Maven for Newbies](#) might also be useful if you've never used Maven before. There is [some support for Ant](#) in this release.

For more information, please see the [2.0 RC1 Release Notes](#). The 2.0 series of AppFuse has a minimum requirement of the following specification versions:

- Java Servlet 2.4 and JSP 2.0 (2.1 for JSF)
- Java 5+

We appreciate the time and effort everyone has put toward contributing code and documentation, posting to the mailing lists, and logging issues.

We also greatly appreciate the help from our sponsors, particularly [Atlassian](#), [Contegix](#), [JetBrains](#), and [Java.net](#). Atlassian and Contegix are especially awesome: [Atlassian has donated licenses to all its products](#) and [Contegix has donated an entire server](#) to the AppFuse project. Thanks guys - you rock!


**Update:** I've uploaded a [247-page PDF version of the RC1 documentation](#) to java.net. This PDF contains the relevant pages from the wiki that help you develop with AppFuse 2.0.

Posted at Sep 04, 2007 by  Matt Raible |  0 comments | [Edit](#)

 Friday, August 10, 2007  
AppFuse vs. Grails vs. Rails

AppFuse often gets compared to [Ruby on Rails](#) and [Grails](#) when folks are talking about full-stack productivity-enhancing frameworks. If you'd like to learn my opinion on this, please read [AppFuse vs. Grails vs. Rails](#) on my Raible Designs blog.

Posted at Aug 10, 2007 by  Matt Raible |  0 comments | [Edit](#)

 Friday, July 27, 2007  
New Server at Contegix

Last changed Jul 27, 2007 08:28 by [Matt Raible](#)

[Contegix](#) has been gracious enough to donate a server to the AppFuse project. Not only do we get a whole server to ourselves, but they're managing it and making sure it stays up all the time.

I've moved [JIRA](#) onto their servers, as well as [Confluence](#). Confluence is at [wiki.appfuse.org](#) and as DNS entries begin to change, [appfuse.org](#) will switch to this server. [apache.appfuse.org](#) is the new "static" server and the DNS change has started for that as well. [demo1.appfuse.org](#) is the new location of [demo.appfuse.org](#) and DNS changes are pending.

If you have a moment, please play a bit with [wiki.appfuse.org](#), [apache.appfuse.org](#) and [demo1.appfuse.org](#) to see if you see anything strange.

In addition to our normal services, we've also got received new licenses from Atlassian for [Crowd](#) (SSO) and [Bamboo](#) (Continuous Integration server). Since Atlassian's tools are built on a lot of the same software that AppFuse uses, I feel like we're somewhat eating our own dogfood.

We merged the accounts for JIRA and Confluence into Crowd. If you had accounts in both, JIRA won (as long as you had the same username, etc.).

The builds for AppFuse 2.x are still done by [Hudson](#), but I hope to change this in the near future. If someone has time to fiddle with Bamboo in the next few days - let me know and I'll give you appropriate permissions.

Thanks Contegix - you guys rock!

Posted at Jul 27, 2007 by  [Matt Raible](#) |  0 comments | [Edit](#)

## Articles

As of January 2007, no articles have been written about AppFuse 2.0. The good news is this site has a plethora of information that should serve you well. If you have trouble finding anything, please [let us know on the mailing list](#).

### AppFuse 1.x Articles

- [AppFuse: Start Your J2EE Web Apps](#) - AppFuse is a web application to accelerate developing J2EE web applications. It's based on open source tools such as Hibernate, Spring, and Struts. It also provides many out-of-the-box features such as security and user management. Creator Matt Raible provides an overview of its history and features.
- [Seven simple reasons to use AppFuse \(Chinese\)](#) - Getting started with open source tools for the Java platform such as Spring, Hibernate, or MySQL can be difficult. Throw in Ant or Maven, a little Ajax with DWR, and a Web framework – say, JSF – and you're up to your eyeballs just trying to configure your application. AppFuse removes the pain of integrating open source projects. It also makes testing a first-class citizen, allows you to generate your entire UI from database tables, and supports Web services with XFire. Furthermore, AppFuse's community is healthy and happy – and one of the few places where users of different Web frameworks actually get along.

In addition, there's a [Chinese Article](#) with no English translation.

## Blogs

The page has a list of AppFuse-related blogs from developers and users.

### Raible Designs

(Matt Raible is a UI Architect specializing in open source web frameworks. [Contact me for rates.](#))



[Livin' it up in Vegas at TSSJS 2011](#)

[Adding Search to AppFuse with Compass](#)

[WebSockets with Johnny Wey at Denver JUG](#)

[JSR 303 and JVM Web Framework Support](#)

[Upgrading to JSF 2](#)

[Fixing XSS in JSP 2](#)

[Upcoming Conferences: TSSJS in Las Vegas and 33rd Degree in Krak](#)

[Implementing Ajax Authentication using jQuery, Spring Security and HTTPS](#)

[Integration Testing with HTTP, HTTPS and Maven](#)

[Implementing Extensionless URLs with Tapestry, Spring MVC, Struts 2 and JSF](#)

[Sanjiv Jivan's Blog](#)



(Yet Another J2EE Blog)

[Smart GWT 2.4 Released](#)

[Smart GWT Touch & Mobile support](#)

[Smart GWT 2.2 Released](#)

[DeltaWalker : My new favorite OSX diff/merge tool](#)

[St. Patricks Day Smart GWT teaser](#)

[Smart GWT 2.1 Released with lots of goodies](#)

[Smart GWT 2.0 Released with Enterprise Blue skin](#)

[SmartGWT 1.3 Released](#)

[When CSV exports should not always be comma separated](#)

[Solving the DTO / Anemic Domain Model dilemma with GWT](#)

[David L. Whitehurst](#)



(Quality Software Development Consultant)

[Rule Language](#)

[What do you really think of Cloud Computing?](#)

[Benefits of a Continuous Build](#)

[Version Control Management](#)

[An AppFuse-Like JSON-to-Flex example](#)  
[Running WebSphere Security local within Rational Application Developer \(RAD\)](#)  
[Dear Mr. Jim Goodnight \(2nd Edition\)](#)  
[Java Guys Try Ruby on Rails](#)  
[Looking for a Remote Architect or Team Lead?](#)  
[CAS'ifying the JBoss jmx-console](#)



Rambling about... 

(Everything and nothing in particular.)

[Triptance.com is a candidate for BBVA Open Talent awards](#)  
[How to startup in Spain: Tetuan Valley Startup School](#)  
[Getting head dumps on OutOfMemoryException](#)  
[Naive scripts for mirroring P2 repositories](#)  
[First impressions on Eclipse 3.5 RC4, IAM, Scala IDE and Lift](#)  
[Saving different Active Record classes in a single transaction](#)  
[Why are my bundles not working?](#)  
[Optimizing Oracle and Hibernate performance, how it turned out](#)  
[Optimizing Oracle and Hibernate performance](#)  
[IAM has two new committers](#)

## Presentations

This page has a list of articles and presentations about AppFuse

- [What's new in AppFuse 2.0](#)
- [Seven Simple Reasons to use AppFuse](#)
- [Migrating from Struts 1 to Struts 2](#)

### What's New in AppFuse 2.0

Created by [Matt Raible](#)

The 2.0 version of AppFuse is designed to make developing Java web applications with Spring much easier. By using and leveraging Maven 2, users will be able to quickly add/remove dependencies, as well as develop their applications quickly and efficiently from their favorite IDE. This release will includes a move to annotations, JSP 2.0 and Spring 2.0.

This presentations covers what's new in AppFuse 2.0, as well as describe experiences migrating from Ant to Maven, Spring 1.x to 2.0, and leveraging annotations for Java development.

 [View Presentation](#)

### Seven Simple Reasons to Use AppFuse

Created by [Matt Raible](#)

AppFuse is an open source project/application that uses best-of-breed Java open source tools to help you develop web applications quickly and efficiently. Not only does it provide documentation on how to develop light-weight POJO-based applications, it includes features that many applications need out-of-the-box: authentication and authorization, remember me, password hint, skinnability, file upload, Ajax libraries, signup and SSL switching. This is one of the main features in AppFuse that separates it from the other "CRUD Generation" frameworks like Ruby on Rails, Trails and Grails. AppFuse is already an application when you start using it, which means code examples are already in your project. Furthermore, because features already exist, the amount of boiler-plate code that most projects need will be eliminated.

In this session, you will learn Seven Simple Reasons to Use AppFuse. If you don't use it to start your own projects, hopefully you will see that it provides much of the boiler-plate code that can be used in Java-based web applications. Since it's Apache Licensed, you're more than welcome to copy/paste any code from it into your own applications.

 [View Presentation](#)

### Migrating from Struts 1 to Struts 2

Created by [Matt Raible](#)

Struts has outgrown its reputation as a simple web framework and has become more of a brand. Because of this, two next generation frameworks are being developed within the project: Shale and Action 2.0. Action 2.0 is based on WebWork, and though its backing beans are similar to JSF, its architecture is much simpler, and easier to use.

Migrating to Struts Action 2.0 is more about unlearning Struts than it is about learning the "WebWork Way". Once you understand how simple WebWork is, you'll find that Struts Action 2.0 is a powerful framework that supports action-based navigation, page-based navigation, AOP/Interceptors, components and rich Ajax support.

Come to this session to see code comparisons and learn about migration strategies to use the kick-ass web framework of 2006.

 [View Presentation](#)

# Project Information

AppFuse provides the open-source community with an excellent example of an application that can be used to kickstart the initial creation of a professional web application. Everything is configured and ready-to-run as soon as it's downloaded. The single most important goal of AppFuse is to provide a foundational application where you don't have to worry about configuring versions, jar dependencies, or integrating bleeding-edge Java technologies. That's where AppFuse provides some comfort. AppFuse solves many integration issues before you fall short trying to do it all by yourself. All of the hot new Java technologies are being considered for inclusion into AppFuse and only the best of breed will finally make the public releases.

Visit here often and please keep an eye on AppFuse. This is where Java development begins. And while AppFuse is free, it takes long hours and hard work, so get the source, create a new project, and log any bugs you find in the source we gave you at our [JIRA issue tracking](#) web site.

## History

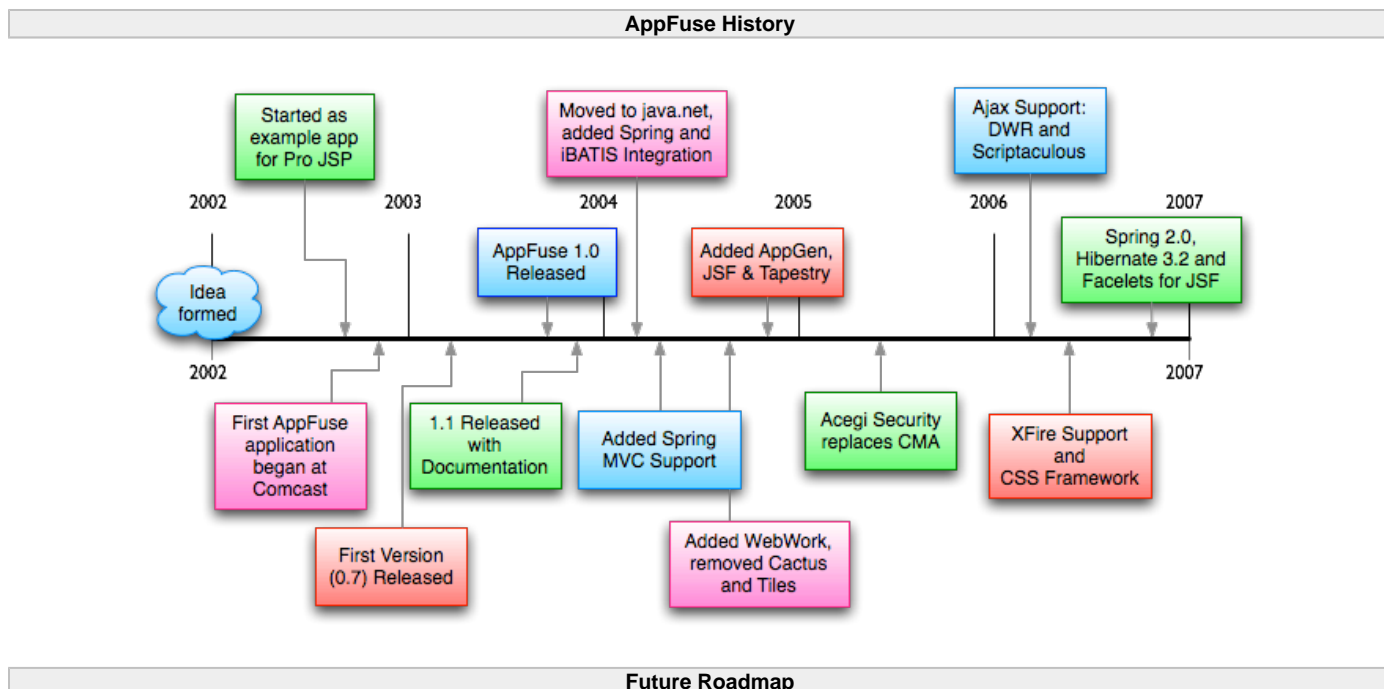
In late 2002 Matt Raible, a software developer, had a vision. This vision was AppFuse. Matt began with sample application concepts or architectures that demonstrate the integration of bleeding-edge Java/J2EE technologies that were available to the open-source community. The application has evolved now from a pattern-driven Struts-Xdoclet application he was using for a book chapter he was authoring on Struts to a professional web application that he implemented at Comcast . Like most developers or software managers, we like things that work. AppFuse is the jumpstart application that works. AppFuse takes a good number of complicated technologies and integrates them in a single application in a modular fashion. This makes it easy to use. And while you're still learning Hibernate or Spring, e.g. you're not waiting on the application to work or come alive.

Most model or archetype projects are simplistic in nature. Example projects are normally simple because you want to copy them for your next project. This is where AppFuse breaks all the rules. Matt developed this complex model application to do it all. But, Matt's logic was different. For example, if you don't need a secure login for your users, you just remove that part. Or, if you prefer to use iBatis instead of Hibernate, you can because an iBatis implementation is there and it works just like the Hibernate one.

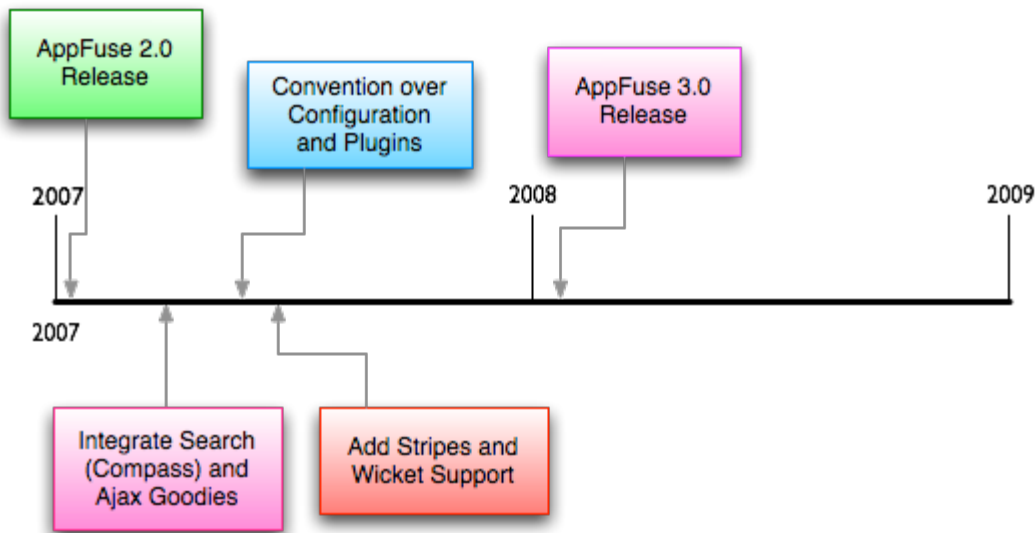
AppFuse has become the model application that Matt Raible has perfected over the years with help from the open-source public. That's you. Please [download it](#) and give it a try for your next web project.

## Timelines

Below are history and roadmap timelines for the AppFuse project. They were created with [OmniGraffle](#). You can read more about AppFuse's history in [AppFuse: Start Your J2EE Web Apps](#).







## Developer Guide

### Committer Information

Below is a list of things you'll need to do after you've become an AppFuse committer.

1. Create an account on <https://appfuse.dev.java.net>. Subscribe to the [users](#), [developers](#) and [issues](#) mailing lists.
2. Create an account in JIRA, you should be able to use it to login to [Confluence](#) and [Bamboo](#) and [FishEye](#). We use [Crowd](#) to manage SSO across the different apps.
3. Once you've been granted the Developer role on java.net, you should be able to checkout the source using the following command. See the [Source Repository](#) page if you need help using Subversion.

```
svn checkout https://appfuse.dev.java.net/svn/appfuse/trunk appfuse
```

4. Install MySQL and an SMTP Server as described in the [QuickStart Guide](#).
5. Execute the following command and grab a beer. Maven will download artifacts, compile, test and run all the Canoo Web tests in Tomcat.

```
cd appfuse
mvn install
```

6. To test archetypes, run `ant -f archetypes/build.xml` from the root directory.
7. To test AMP, run `ant -f plugins/appfuse-maven-plugin/build.xml` from the root directory.

If everything passes, you're ready to start contribute some code! The following steps should help you understand the project better.

1. In 2.0.x, AppFuse uses a war-overlay system where `web/common` is a WAR that's overlayed into other WARs (`web/*`). The [maven-warpath-plugin](#) allows WAR projects to depend on the classpath of their dependent WARs. It's likely we'll [move away from this in 2.1](#). Because of this setup, when testing web modules, you have to use `mvn jetty:run-war` instead of `mvn jetty:run`.
2. Read [Project Policies](#) and [Continuous Integration](#).
3. Add your picture and bio to the [Project Team](#) page.
4. Please ask any questions you may have on the developers mailing list.

### Deploying Snapshots

To deploy AppFuse snapshots to <http://static.appfuse.org/repository>, create a `~/.m2/settings.xml` file with the following settings:



```
<?xml version="1.0" encoding="UTF-8"?>
<settings>
  <servers>
    <server>
      <id>appfuse-snapshots</id>
      <username>username</username>
      <password>password</password>
    </server>
  </servers>
</settings>
```

Then you can run "mvn deploy" from the AppFuse project. Please contact Matt Raible for the username and password.

## Deploying Plugin Websites

To deploy AppFuse plugins' documentation to <http://static.appfuse.org/plugins>, create a `~/m2/settings.xml` file with the following settings:

```
<?xml version="1.0" encoding="UTF-8"?>
<settings>
  <servers>
    <server>
      <id>appfuse-plugins</id>
      <username>username</username>
      <password>password</password>
    </server>
  </servers>
</settings>
```

Then you can run "mvn deploy" from the "plugins" module. Please contact Matt Raible for the username and password.

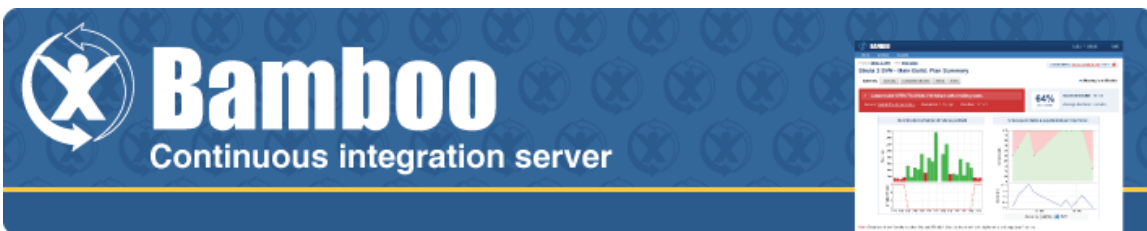
Currently deployed plugin websites include the [AppFuse Plugin](#) and the [WarPath Plugin](#).

## Continuous Integration

Bamboo is used for continuous integration on the AppFuse project. You can view the current status of the build at the following URL:

<http://builds.appfuse.org>

If you'd like to be notified of build status, you can subscribe to its [RSS feed](#).



## Project Policies

The following policies and procedures are intended to ensure that AppFuse will continue to achieve its project objectives and support the community in the context of an expanding development team.

1. This project uses [JIRA](#). Please log a task in JIRA for any changes you make to SVN, with the exception of very minor changes that users are unlikely to ever be interested in searching for and/or the change affects code that has never been in an officially released version of the project (e.g. ongoing changes to a new feature in SVN HEAD that hasn't been released previously).
2. Any users running from SVN HEAD are warmly encouraged to [subscribe to the commits mailing list \(RSS\)](#) so that they can keep an eye on commit comments. Developers are encouraged to join this mailing and read the commit comments. If anyone has a concern with any commit, please raise it on [the dev mailing list \(RSS\)](#) so that the broader community can participate. Alternatively, contact the author of the change directly if you think that would be more appropriate or diplomatic.
3. Please make your commit comments informative, yet not too detailed. If you include the JIRA issue number (i.e. APF-200), the changes will show up under the "FishEye" link in JIRA. Detailed comments are ideally placed in the JIRA task. In the case of a contribution by a

non-developer, please use the SVN commits to reflect who provided the contribution and add that person's name to /pom.xml in the contributors section. If the contributors section does not list the name of someone who has contributed accepted code, please add them.


4. If you add a major new feature, please announce it on the developers mailing list. That way people using the project have an idea of what is coming up in the next release, and any implementation-specific comments can be received prior to the first release when users will start expecting some degree of consistency and stability. It also encourages people to try out your new feature.
5. Until we add some sort of source formatting (i.e. Jalopy), please follow Sun's [Code Conventions for Java](#). In other words, curly braces on the same line and 4 spaces instead of tabs.
6. The /sandbox can be used to obtain feedback from fellow developers and the community about your code, general approach or new ideas. If you have SVN rights, please use /sandbox instead of emailing ZIP files to other developers for feedback. The community should understand that code in the sandbox is unsupported, subject to refactoring, may not have any unit tests, and may be removed at any time. The /sandbox will never be included in official releases. It's a "scratching pad" only.
7. Never check in code if the unit tests fail. This means, at minimum, successfully running "mvn" from the root directory. Always name your unit test classes so they end in "Test" - this ensures that Maven picks them up. If there is code in SVN which you didn't write and it is breaking the unit tests, please correct it yourself - don't leave SVN "broken" whilst waiting for the responsible developer to address it (the delay causes confusing and long-running threads on the list and forum). You can always rollback to the previous working version if in doubt of how the class works (just remember to comment the commit appropriately and let the author know).
8. Please update the Reference Guide, Tutorials and JavaDocs for any new major features. The JavaDocs should always be correct. The reference guide may be kept updated with less rigor, although please briefly discuss any major new features.
9. Developers should try to monitor the [user](#) and [issues](#) mailing lists. RSS feeds are available for both of these lists via [Nabble](#). The user list is very active, and it takes a lot of work if not shared around. Please don't hesitate to reply to users. Please read Kathy Sierra's [How to Build a User Community](#) for advice on how we can make the AppFuse Community a great place to be.
10. You should feel free to put to vote any proposed new developers. New developers should firstly be encouraged to attach patches to JIRA tasks to illustrate their understanding of the project, or, if they're long-time users, they might be given access without this JIRA stage if they're undertaking a major new feature.
11. Developers should be subscribed to the developer mailing list. Obviously it would take significant time to read every thread, but reading the high priority messages (as indicated by the subject line) is needed to ensure we all have a way of communicating.
12. Please do not hesitate to assign yourself any JIRA task that is unassigned, or assigned to me and not in the "In Progress" status. Also feel free to approach fellow developers to volunteer to work on tasks they might be assigned but haven't started.
13. No code in SVN is "sacred". If you have a good idea or refactoring for an area of code that someone else wrote, raise it on developer list or contact the author directly. Please don't commit changes to such code unless it is a unit test failure correction, or you've firstly raised it on the developer list or directly with the author.
14. People's priorities are ever-changing, and we're all short on time. For this reason it's perfectly understandable that over time developers will move on to other things. This is not a negative reflection in any way - just part of any long-term project. If a developer no longer has the time or inclination to participate in the project, please send an email to [dev@appfuse.dev.java.net](mailto:dev@appfuse.dev.java.net) or myself. I will remove the SVN rights and reassign any JIRA tasks. Importantly, this helps find a new maintainer of the former developer's code (or, in very extreme cases, their code might be relocated to the sandbox or removed).
15. Keep the warm community spirit. The AppFuse community is a nice place to be - especially compared with some of the other open source communities out there where people are abused, ignored, insulted or excluded. No policy or procedure (including those above) should ever compromise operating in a considerate and diplomatic manner that respects the dignity of each individual member of the community. If in doubt, please contact me directly first. If I am ever guilty of this, please let me know and I will correct myself.



Thanks for your help in connection with the above. If you have any suggestions for improving these policies and procedures, please use the developer list to raise them.

Matt Raible  
Project Admin

## Release Process

✔ To keep this list current, it is **strongly recommended** that release managers refer to and follow this list each time a distribution is created. If any of the steps need to be amended, then please update the list.

1	Review <a href="#">JIRA</a> for any issues without a fix version set, and for any issues that should be resolved for the pending release.
2	<div>Delete the appfuse artifacts in your local Maven2 repository and obtain a fresh checkout:</div> <div><pre>svn co https://svn.java.net/svn/appfuse~svn/trunk appfuse-#-#</pre></div>
	<div> Make sure and disable <a href="#">Bamboo's deploy plan</a> to ensure artifacts are not deployed during a release.</div>
3	Find and replace <code>##-##-SNAPSHOT</code> in build.xml and *.java. Find and replace <code>&lt;appfuse.version&gt;##-##-SNAPSHOT&lt;/appfuse.version&gt;</code> with <code>&lt;appfuse.version&gt;##-##&lt;/appfuse.version&gt;</code> . Use "Preparing for ##-## Release" as the commit message.

	<div>  <p>Make sure your MAVEN_OPTS environment variable has at least 128MB of memory and no debugging options. An ideal setting is <code>-Xms128M -Xmx256M</code>.</p> </div>
4	Run the Maven Release Plugin to prepare the release:
	<pre>mvn release:prepare -Dtag=APPFUSE_#.#.# -DautoVersionSubmodules=true -Dusername=SVNUSERNAME -Dpassword=XXX</pre>
5	Run the commands below to perform the release:
	<pre>mvn release:perform -Dusername=SVNUSERNAME -Dpassword=XXX -Dpgp.passphrase="XXX" -Darguments="-Dpgp.passphrase=XXX"</pre> <p>See <a href="#">MGPG-9</a> for more information.</p>
	<div>  <p>This may fail because AppFuse plugins and dependencies can't be found. Run the following command to fix this problem.</p> <pre>mvn install -DskipTests -f target/checkout/pom.xml</pre> </div>
6	Run the Maven Release Plugin again - using the same command you did in Step 5 (installing plugins again is unnecessary).
7	Review and Close the <a href="#">Staged Release</a> Use "Staged release of AppFuse #.#.#." as the Close message. Promote the release to <b>AppFuse Releases</b> after closing.
8	Update <a href="#">JIRA</a> with tag/release date; Update <a href="#">QuickStart Guide</a> with new version; Create new release notes page, and link to prior release page and JIRA filters. Add News story with link to JIRA Release Notes and Upgrade Guide.
9	Export the Confluence documentation as a PDF and upload to java.net.
10	Create a bundle of all the dependencies used by AppFuse and upload to java.net.
11	Update demos on <a href="#">demo.appfuse.org</a> .
12	Find and replace <code>###</code> with <code>###-SNAPSHOT</code> in <code>build.xml</code> and <code>*.java</code> . Find and replace <code>&lt;appfuse.version&gt;###&lt;/appfuse.version&gt;</code> with <code>&lt;appfuse.version&gt;###-SNAPSHOT&lt;/appfuse.version&gt;</code> . Use "Preparing for next development iteration" as the commit message.
13	Deploy the new snapshot from the AppFuse trunk (with the latest POMs):
	<pre>mvn install deploy -DskipTests</pre>
14	Announce the release on the user list and enjoy your success.
15	Re-enable the <b>deploy</b> plan in Bamboo.

## Resources

- [Apache Struts Release Guidelines](#)
- [Struts Maintenance Guide for Maven](#)
- [OpenJPA Release Guidelines](#)

## Retagging

If a tagged build needs to be retagged, be sure to delete the old tag first.

```
svn delete https://svn.java.net/svn/appfuse-svn/tags/APPFUSE_#_#_# -m "APF-### Removing first try at 2.##.##."
```

## Sample Test Build Announcement

✓ Test builds are only announced to the dev list.

*The test build of AppFuse 2.0 is available.*

*No determination as to the quality ('alpha,' 'beta,' or 'GA') of this release has been made, and at this time it is simply a "test build". We welcome any comments you may have, and will take all feedback into account if a quality vote is called for this build.*

*Release notes:*

- [LINK]

*Maven 2 staging repository:*

- <http://static.appfuse.org/repository>

*We appreciate the time and effort everyone has put toward contributing code and documentation, posting to the mailing lists, and logging issues.*

## Sample Release/Quality Vote

*The AppFuse ### test build has been available since ## ### ##.*

*Release notes:*

- [LINK]

*Maven 2 staging repository:*

- <http://static.appfuse.org/repository>

*If you have had a chance to review the test build, please respond with a vote on its quality:*

☐ Leave at test build  
☐ Alpha  
☐ Beta  
☐ General Availability (GA)

*Everyone who has tested the build is invited to vote. Votes by AppFuse committers are considered binding. A vote passes if there are at least three binding +1s and more +1s than -1s.*

*Please remember that a **binding** +1 for GA implies that you intend to support the release by applying patches and responding to posts to the user and dev lists.*

## Sample Release Announcement

*The AppFuse team is pleased to announce the release of AppFuse ### \$GRADE.*

*AppFuse ### is available as a Maven archetype. For information on creating a new project using this release, please see the QuickStart Guide:*

<http://appfuse.org/display/APF/AppFuse+QuickStart>

*The ###.x series of AppFuse has a minimum requirement of the following specification versions:*

- Java Servlet 2.4 and JavaServer Pages (JSP) 2.0
- Java 2 Standard Platform Edition (J2SE) 5.0

*The release notes are available online at:*

- [LINK]

## Issue Tracking

This project uses [JIRA](#) for issue tracking and project management.

Issues, bugs and feature requests should be submitted to the following issue tracking system for this project.

<http://issues.appfuse.org/browse/APF>

## Mailing Lists

There are a number of mailing lists that have been established for the use and development of AppFuse. Even if you don't use AppFuse, we're confident you'll enjoy joining our community. We use some of the hottest technologies in Java today. If you use Spring, Hibernate, JSF, Maven 2 or Ajax, we'd love to hear your insights. Not only do we think that there's no such thing as a dumb question, we also believe [there's no such thing](#) as a dumb answer. 😊

### User List

This list is the heart of the AppFuse community. All support questions, design pondering and shared successes are conducted here.

- [Subscribe](#)
- [Post](#)
- [Unsubscribe](#)
- [Archive](#)
  - [MarkMail](#)
  - [Nabble](#)
  - [Gmane](#)

To ask a single question and subscribe to its answers, please use our [forums](#), graciously hosted by [Nabble](#).

### Developer Lists

These lists are used by the AppFuse development team to discuss the direction of the project as well as keep track of issues and changes.

Feel free to join any of these you are interested in, but please be sure to post any support questions to the User list. All of the AppFuse developers actively read and respond to the User list, but many experienced AppFuse users who may have encountered similar problems do not read the development list.

#### Dev

- [Subscribe](#)
- [Unsubscribe](#)
- [Post](#)

#### Issues

- [Subscribe](#)
- [Unsubscribe](#)

#### Changelog

- [FishEye](#)
- [RSS Feed](#)
- [Subscribe](#)
- [Unsubscribe](#)

## Source Repository

This project uses [Subversion](#) to manage its source code. Instructions on Subversion use can be found at <http://svnbook.red-bean.com/>.

Access the source code repository for the AppFuse project in one of the following ways:

- Browse source code online to view this project's directory structure and files: [java.net](#) and [FishEye](#)
- Check out source code with a Subversion 1.x client. For example:

```
svn checkout https://svn.java.net/svn/appfuse~svn/trunk appfuse --username guest
```

The password for the guest account is "" (nothing).

The best clients for Subversion are [TortoiseSVN](#) (Windows only), [SmartSVN](#) (Java) and [Subclipse](#) (Eclipse). For command line clients, we recommend [Cygwin](#) on Windows. For OS X, you should be able to install [Fink](#) and run "fink install svn-client".

### Access from behind a firewall

For those users who are stuck behind a corporate firewall which is blocking http access to the Subversion repository, you can try to access it via the developer connection:

```
$ svn checkout https://svn.java.net/svn/appfuse~svn/trunk appfuse
```

### Access through a proxy

The Subversion client can go through a proxy, if you configure it to do so. First, edit your "servers" configuration file to indicate which proxy to use. The files location depends on your operating system. On Linux or Unix it is located in the directory "~/.subversion". On Windows it is in "%APPDATA%\Subversion". (Try "echo %APPDATA%", note this is a hidden directory.)

There are comments in the file explaining what to do. If you don't have that file, get the latest Subversion client and run any command; this will cause the configuration directory and template files to be created.

Example : Edit the 'servers' file and add something like :

```
[global]
http-proxy-host = your.proxy.name
http-proxy-port = 3128
```

### Access via a Git mirror

If you prefer to use Git as your Version Control System, an **unofficial** mirror of the AppFuse repository is available. It gets updated several times a week, and is imported using the [git-svn](#) tool.

In common with other distributed version control systems (DVCS), when you "get" (clone) the git repository, you'll have the entire AppFuse development history available to you.

To clone the Git repository:

```
git clone git://github.com/myabc/appfuse.git
```

## Sponsors

Many organizations have donating free products and hosting for AppFuse development. These companies are truly outstanding in their support of open-source projects, and their products/services rock too! We'd especially like to recognize [Contegix](#) (hosts JIRA, Confluence, Bamboo and Crowd) for their awesome support services. Of course, Atlassian rocks too for providing us with free licenses for their suite of excellent products.

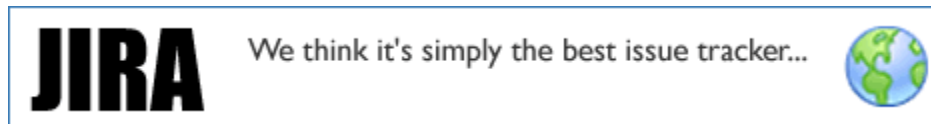
[Raible Designs](#) - sponsors development of AppFuse and offers support services.

*Raible Designs*

[Adaptavist](#) - free [Builder](#) license for Confluence.



Atlassian - free licenses for Bamboo, Confluence, Crowd and JIRA.



Cenqua - provides FishEye monitoring of SVN Repository.



Contegix - provides free hosting of JIRA Issue Tracker.



Gliffy - free license to the Gliffy Plugin for Confluence.



IntelliJ - free IDEA licenses for developers.



java.net - provides source control and mailing lists.



ej-technologies - free JProfiler licenses for developers.



YourKit - free YourKit licenses for developers.



<oXygen/> - free <oXygen/> licenses for developers.



SoftLogica - free WAPT (Web Application Load, Stress and Performance Testing) licenses for developers.



## Donate to the AppFuse Beer Fund

If you'd like to make a donation to the AppFuse Beer Fund, please click on the PayPal image below. Funds will be used to celebrate releases and motivate developers to stay up late working on AppFuse. 😊

## Reference Guide

There are many different components and systems involved when building a modern web application. From the build tool to the business logic to the cool effects you can do with Ajax. This reference guide is a work-in-progress that's created based on user feedback when using AppFuse. Please click on one of the topics below to continue.

- [Ajax](#)
- [AppFuse Maven Plugin](#)
- [CSS Framework](#)
- [Database Profiles](#)
- [IDEs](#)
- [Licenses](#)
- [Maven 2](#)
- [OS Specific Issues](#)
- [Persistence Frameworks](#)
- [Security](#)
- [SiteMesh](#)
- [Spring](#)
- [Version Control](#)
- [Web Filters](#)
- [Web Frameworks](#)
- [Workflow](#)

## Ajax

AppFuse ships with a number of Ajax libraries:

- [DWR](#)
- [Prototype](#)
- [Scrip.taculo.us](#)

DWR is **Easy Ajax for Java**. It allows you to easily expose Java objects (including Spring beans) as JavaScript objects. It's one of the best available libraries for Ajax development. To learn more about DWR and how it works, see [Developing AJAX Applications the Easy Way](#) by Joe Walker.

Prototype and Script.aculo.us are both libraries that simplify Ajax development. By using them, you won't have to worry as much about browser compatibility issues.

For good documentation about Prototype, see:

- Ryan Campbell's [Quick Guide to Prototype](#)
- Sergio Pereira's [Developer Notes for prototype.js](#)

The best Script.aculo.us documentation is available on the [Script.aculo.us Wiki](#).

In addition to these libraries, Struts 2 and Tapestry both include [Dojo](#). JSF and Spring MVC do not ship with any built-in Ajax libraries.







Here is a dojo example using Struts2. There is a normal action with a method save and a set method for the object EvenSpan which holds an id and a name.

```
<script type="text/javascript">
  djConfig = { isDebug: false };
</script>
<script type="text/javascript" src="../../scripts/dojo/dojo.js"></script>

<script type="text/javascript">
  dojo.require("dojo.event.*");
  dojo.require("dojo.io.*");

  function callback(type, data, evt){
    if (type == 'error'){
      alert('Error when retrieving data from the server!');
    }
  }

  function doOnCellEdit(id,value){
    dojo.io.bind({ url: 'eventSpan.html',
      handler: callback,
      content: { 'method:save':true, 'eventSpan.name':value,'eventSpan.id':id }
    });

    return true;
  }
</script>
```

this part is an extension on the following fine tutorial:

<http://cwiki.apache.org/S2WIKI/struts-2-spring-2-jpa-ajax.html>

take care: the tutorial does not do anything apart from demonstrating the working of ajax. the java code is prototype only.

This tutorial misses out on 3 things:

1a) integration it within appfuse setup, taking care of the dojo files  
1b) integration it within appfuse setup, taking care of the importing order of the js files

2) getting it working with the decorator, or better without 😊

1a) the dojo files can't be run from the struts.jar file.

To solve this open the struts2-core-2.0.6.jar  
copy the dojo folder into  
src/main/webapp/struts/dojo

Open your web.xml and make sure the staticFilter is including the right folder:

```
<filter>
  <filter-name>staticFilter</filter-name>
  <filter-class>org.appfuse.webapp.filter.StaticFilter</filter-class>
  <init-param>
    <param-name>includes</param-name>
    <param-value>/struts/dojo/*</param-value>
  </init-param>
</filter>
```

1b) make sure the dojo files are imported first.

To do this open your /src/main/webapp/decorators/default.jsp file and add this line above! all other js imports:

```
<s:head theme="ajax" debug="true"/>
```

(otherwise it will conflict with at least the scriptaculous.js)

2) make sure your ajax return string will not be decorated.  
There are many option but i like to do this:

open your decorators.xml file:

```
<decorators defaultdir="/decorators">
  <excludes>
    <pattern>/*ajax=true*</pattern>
    <pattern>/*decorate=false*</pattern>
    <pattern>/struts/dojo/*</pattern> <!-- OK to remove if you're
not using Struts -->
    <pattern>/resources/*</pattern>
  </excludes>
  <decorator name="default" page="default.jsp">
    <pattern>/*</pattern>
  </decorator>
</decorators>
```

I added the decorate=false part. URLs with this parameter will not be decorated.

So an example would be nice right?

Below an ajaxTest.jsp page which will be the caller.

Then an ajaxReturn.jsp page which will be the returned data.

I expect you can make an AjaxAction with the methods String ajax(), getPersons() etc..by your self.

This is connected by struts like this:

```
<action name="ajax" class="ajaxTest">
  <result>/WEB-INF/pages/employee/ajaxTest.jsp</result>
  <result
name="ajax">/WEB-INF/pages/employee/ajaxReturn.jsp</result>
</action>
```

The ajaxTest.jsp:

```

<%@ taglib prefix="s" uri="/struts-tags"%>
  <head>
    <script type="text/javascript">
      dojo.event.topic.subscribe("/edit", function(data, type,
request) {
        alert("type: "+type);
        alert("data: "+data);
        if(type="load"){
          document.getElementById("result").innerHTML=data;
        }
      });
    </script>
  </head>

  <!-- URL link to struts action-->
  <s:url id="ajaxText" action="ajax" method="ajax" >
  <s:param name="decorate" value="false" />
  </s:url>

  <!-- Div where content will be displayed -->
  <s:div theme="ajax" id="weather" href="{ajaxText}">
    loading content... from the ajax action the ajax method is called.
    than the ajaxReturn.jsp is rendered here.
  </s:div>

  <p>Persons</p>
  <s:if test="persons.size > 0">
    <table>
      <s:iterator value="persons">
        <tr id="row_<s:property value="id"/>">
          <td>
            <s:property value="firstName" />
          </td>
          <td>
            <s:property value="lastName" />
          </td>
          <td>
            <!-- call the remove method on the ajax action no return-->
            <s:url id="removeUrl" action="ajax" method="remove">
              <s:param name="id" value="id" />
              <s:param name="decorate" value="false" />
            </s:url>
            <s:a href="{removeUrl}" theme="ajax" >Remove</s:a>

            <!-- call the edit method an the ajax action. the
result (ajaxResult.jsp)
will be handed to the edit javascript mothed
attached to dojo (above) -->
            <s:url id="editUrl" action="ajax" method="ajax">
              <s:param name="id" value="id" />
              <s:param name="decorate" value="false" />
            </s:url>
            <s:a href="{editUrl}" id="a_{id}" theme="ajax"
notifyTopics="/edit">Edit</s:a>
          </td>
          <td>
            <a href=ajax!remove.html?id=2>remove me no ajax</a>
          </td>
        </tr>
      </s:iterator>
    </table>
  </s:if>

  <hr>
  <div id=result></div>

```

The ajaxResult.jsp:

```
<%@ taglib prefix="s" uri="/struts-tags"%>
Make some nice form here. Now show all persons.
<s:iterator value="persons">
<table><tr><td><s:property value="firstName" /></td>
<td><s:property value="lastName" /></td>
</tr>
</table>
</s:iterator>
```

OK here is my action to:

```
package nl.incipio.match.webapp.action;

import java.util.List;

import org.appfuse.model.User;

import com.opensymphony.xwork2.Preparable;

import nl.incipio.match.util.MyBaseAction;

public class AjaxTestAction extends MyBaseAction implements Preparable {
    private static final long serialVersionUID = 378605805550104346L;

    private List<User>      persons;

    private Long            id;

    @Override
    public String execute() throws Exception {
        log.debug("just getting the stuff");
        persons = (List<User>) getRequest().getAttribute("persons");
        if (persons == null) {
            log.debug("just ones please");
            persons = userManager getUsers(null);
            getRequest().setAttribute("persons", persons);
        } else {
            log.debug("persons" + persons.size());
        }
        return SUCCESS;
    }

    public List<User> getPersons() {
        return persons;
    }

    public void setPersons(List<User> persons) {
        this.persons = persons;
    }

    public String remove() throws Exception {
        log.debug("do some removing here when i feel like it id:" + id);
        if (persons != null) {
            log.debug("before persons" + persons.size());
            persons.remove((id.intValue() - 1));
            log.debug("after persons" + persons.size());
        }
        return SUCCESS;
    }

    public String save() throws Exception {
        log.debug("do some saving here when i feel like it");
    }
}
```

```
        return execute();
    }

    public String ajax() {
        log.debug("ajax is doing something id:"+id);
        return "ajax";
    }

    public String edit() throws Exception {
        log.debug("do some editing here when i feel like it");
        return execute();
    }

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public void prepare() throws Exception {
        log.debug("i'm getting prepared!!!");
    }
}
```

```
}  
}
```

spring config in applicationContext.xml:

```
<bean id="ajaxTest"  
class="nl.incipio.match.webapp.action.AjaxTestAction">  
  <property name="userManager" ref="userManager"/>  
</bean>
```

## AppFuse Maven Plugin

### About the AppFuse Maven Plugin (AMP)

This plugin currently does two things:

- Code generation for CRUD
- Customizing AMP Templates
- Converts your project to use AppFuse's source

For more documentation on this plugin, see [its Maven-generated site](#).

### Generating CRUD with AMP



#### Stop and think before generating code!

At first, I didn't want to add a code-generation feature like this because you end up with a 1-to-1 relationship between tables/pojos, DAOs and Managers. On most of my projects, I have far fewer DAOs and Managers than POJOs.

Using this plugin, you can run the following command to generate CRUD screens/classes for a POJO:

```
mvn appfuse:gen -Dentity=Name
```



#### Package Names

The default package name is `${groupId}.model.Name`. If you want to generate your code in a sub-package, specify a full package name like `-Dentity=com.myproject.subpackage.model.Name` or `-Dentity=~.subpackage.model.Name`. The tilde is a shortcut for `${groupId}`.

If you don't specify the entity name, you're prompted for it. After generating the code, the plugin will install it for you as well, unless you specify `-DdisableInstallation=true`. If you disable installation, you can install it using:

```
mvn appfuse:install -Dentity=Name
```

To remove the installed artifacts, use:

```
mvn appfuse:remove -Dentity=Name
```

If your entity is not defined in hibernate.cfg.xml, it will be added. In a modular project, these commands must be run in the "core" and "web" modules. The plugin is smart enough to figure out when it should/should not generate stuff based on the packaging type (jar vs. war).

There's also a goal that allows you to generate model objects from database tables:

```
mvn appfuse:gen-model
```

Once you've generated and installed the POJO, you can generate crud for it using the `appfuse:gen` command.

We hope to [combine `gen` and `gen-model` into a single command](#).

## Customizing AMP Templates

The FreeMarker templates that AMP uses to generate code are packaged in the plugin itself. Since version 2.0.2, you can copy the code generation templates into your project using the following command:

```
appfuse:copy-templates
```

In previous versions, you can customize templates using the following steps:

1. Checkout the plugin from SVN (username: **guest**, password: <blank>):

```
svn co https://appfuse.dev.java.net/svn/appfuse/trunk/plugins/appfuse-maven-plugin
appfuse-maven-plugin
```

2. Customize the templates in `src/main/resources/appfuse`.
3. Run **mvn install** (use `-Dmaven.test.skip=true` if tests fail and you don't want to fix them).
4. Make sure your project matches the version number you just installed.

## Installing AppFuse's source into your project

The good news is creating an "old style" project is now pretty easy. If you create a new project using 2.0-m5+, you can now use:

```
mvn appfuse:full-source
```

This goal will convert your project to use all of AppFuse's source and remove all dependencies on AppFuse. It will also refactor all package names and directories to match your project's groupId.



### For AppFuse versions < 2.1.0

The SVN repository location has changed so you'll need to add the following to the `appfuse-maven-plugin`'s configuration section for everything to work.

```
<trunk>https://svn.java.net/svn/appfuse~svn/</trunk>
```

What the full-source plugin does:

1. Exports all sources from Subversion into your project. It reads the `dao.framework` and `web.framework` properties to determine what you need.
2. Removes warpath plugin from `pom.xml`.
3. Calculates dependencies by reading `pom.xml` files from the various AppFuse modules. It replaces your dependencies with these new ones. The order of the dependencies added is alphabetical based on groupId.
4. Reads properties from the root AppFuse `pom.xml` and adds the ones that don't exist to your project.
5. Renames packages to match your project's groupId.

## CSS Framework

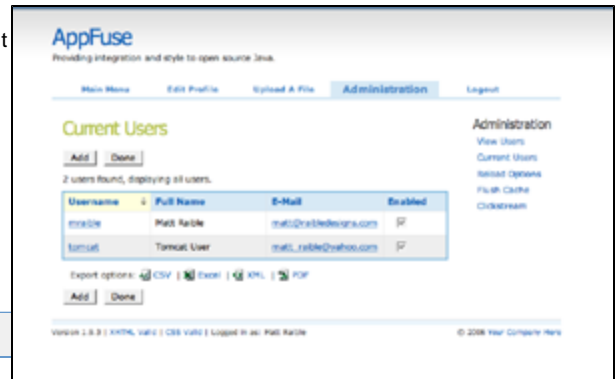
See Mike Stenhouse's [A CSS Framework](#) to learn more about the CSS Framework used in AppFuse. A [CSS Framework Design Contest](#) helped to come up with all the [theme choices for AppFuse](#). Currently, AppFuse ships with [puzzlewithstyle](#), [andreas01](#) and [simplicity](#). More themes from the design contest will be added as demand warrants.

First off, we recommend you [get Firefox](#) with [FireBug](#) installed if you don't have it already - it is a fantastic tool for debugging CSS. Just click on an element in the DOM view, and it will not only show you where that element is on the screen, but also how it is styled and, more importantly, where it gets its styling instructions from.

## Change a profile

If you want to use a different profile open your `web.xml` and change the following context parameter:

```
<context-param>
  <param-name>csstheme</param-name>
  <param-value>simplicity</param-value>
</context-param>
```



## Common Questions

- [How do I adjust the Simplicity theme so it fills the width of my browser?](#)

## How do I adjust the Simplicity theme so it fills the width of my browser?

The divs are explicitly sized in the **simplicity** theme. If you want to display wide lists, you probably need to increase their size. You will find all you need inside `layout-navtop-subright.css`, which is the layout used by default. 1 main column 530 pixels wide with a side column on the right 120 pixels wide. Both these sit inside the content pane which is 701 pixels wide, which in turn sits inside the page div which is 780 pixels wide (inside `layout.css`). To display a wide list, increase the size of at least the page, content pane and the main column. You may also consider using the `layout-1col.css`, which only has a single column for content. To switch change the name of the include file inside `theme.css`.

## Database Profiles

A new feature in [AppFuse 2.0 M4](#) is [Database Profiles in AppFuse 2.0](#).

It means AppFuse should work out-of-the-box with several databases, including:

- Derby
- H2
- HSQLDB
- MySQL
- Oracle
- PostgreSQL
- SQL Server

## IDEs

AppFuse 2.x uses Maven as its build system. IDE support comes in two flavors:

- Maven has the ability to generate project files for most major IDEs.
- IDE Plugins exist so you can run Maven from w/in your IDE.

Using Maven to generate your IDE's project files is recommended. The best IDEs for developing AppFuse-based applications are:

- [Eclipse](#)
- [IDEA](#)
- [MyEclipse](#)
- [NetBeans](#)

If you're unsure of which IDE to use, you might want to read [Comparing IDEs and Issue Trackers](#). IDEA seems to work best with Maven 2.

## Eclipse

Eclipse is by far and away the most popular IDE among Java developers.



## Table of Contents

1. Download and Install
2. Integration with WTP
3. Debugging
4. Spring Support
5. Tips and Tricks

## Download and Install

To make your project Eclipse-aware, perform the following steps:

1. Read [Development Environment](#) for additional configuration information.
2. Download [Eclipse 3.2](#) or [Eclipse 3.3Rc2](#) and use Callisto (Software Update) to get plugins for developing web and database-driven applications.
3. Install Eclipse into \$TOOLS\_HOME/eclipse-3.2.
4. When starting Eclipse, set your workspace directory to C:\Source on Windows and ~/dev on \*nix.
5. In your project's directory, run **mvn eclipse:eclipse**. This will generate project files using the [Maven Eclipse Plugin](#). If you're using a modular archetype, you may need to run **mvn install eclipse:eclipse**.



### Useful Information

Since the newer versions might give you some errors like: "Request to merge when 'filtering' is not identical.....", when you run `mvn eclipse:eclipse`. Try using version 2.6, by running this command instead: `mvn org.apache.maven.plugins:maven-eclipse-plugin:2.6:eclipse`

1. Launch Eclipse and go to **File > Import > Existing Projects into Workspace** (under the General category). Select the root directory of your project, followed by the modules to import. Click **Finish** to complete the process.

Set the classpath variable `M2_REPO`. Eclipse needs to know the path to the local maven repository. You should be able to do this by executing the command: **mvn -Declipse.workspace=C:\Source eclipse:add-maven-repo**.

If that doesn't work, can also define a new classpath variable inside Eclipse. From the menu bar, select *Window > Preferences*. Select the *Java > Build Path > Classpath Variables* page and set `M2_REPO` to equal `~/ .m2/repository`.

After configuring Eclipse, you should be able to compile your project and run tests from within your IDE. For tests that rely on pre-existing data, you may have to periodically run **mvn dbunit:operation** to re-populate your database. You shouldn't need to worry about deploying from Eclipse because you can use the Jetty Plugin (**mvn jetty:run-war**) or Cargo (**mvn cargo:start -Dcargo.wait=true**). If you'd prefer to use the [Eclipse Web Tools Project](#) to auto-deploy your project, see the [Integration with WTP](#) section below.

For more information on using Eclipse with Maven, see [Maven's Eclipse Mini-Guide](#).

If you'd like to run Maven from Eclipse, see the [Maven 2.x Eclipse Plugin](#).



The Eclipse project is likely to show multiple errors which are in fact not really errors. See [APF-649](#)



### Useful Information

If you've used the `"mvn war:inplace"` command, you'll likely have errors in Eclipse about conflicting libraries. These can be fixed by deleting the `src/main/webapp/WEB-INF/lib` directory.

```
rm -r src/main/webapp/WEB-INF/lib/
```

## Integration with Eclipse Web Tools Project (WTP)

See: [Eclipse Web Tools Project](#)

Below are some quick instructions contributed by Shash Chaterjee. For more information see [this thread](#).

1. Use the AppFuse archetypes to create the project. I used the Struts-2 Modular version. If you're not using a modular archetype, you can skip to step #6.
2. Under "myproject", create a new dir called `pom`. Copy `myproject/pom.xml` to `myproject/pom/pom.xml`.
3. Modify `myproject/pom.xml` to change the artifactId to "myproject-pom".
4. Modify `core/pom.xml` and `web/pom.xml` to change the parent pom artifactId to "myproject-pom" and delete the `relativePath` element
5. Modify `myproject/pom.xml` to add the "pom" module preceding to core and web in the modules section
6. Modify `myproject/pom/pom.xml` to remove the "core" and "web" modules
7. Run **mvn install eclipse:eclipse**

8. Edit `myproject/web/.settings/org.eclipse.wst.common.component`:
  - Delete `<wb-resource deploy-path="/" source-path="src/main/webapp"/>`
  - Add `<wb-resource deploy-path="/" source-path="target/myproject-webapp-1.0-SNAPSHOT"/>`
  - Delete `<wb-resource deploy-path="/WEB-INF/classes" source-path="src/main/resources"/>`
  - Edit `myproject/web/.classpath`
  - Delete `<classpathentry kind="src" path="src/main/resources" excluding="ApplicationResources_zh*.properties ... **/*.java"/>`
  - Add `<classpathentry kind="con" path="org.eclipse.jst.server.core.container/org.eclipse.jst.server.tomcat.runtimeTarget/Apache Tomcat v5.5"/>`
  - Add `<classpathentry kind="con" path="org.eclipse.jst.jee.internal.web.container"/>`
9. Preferences -> Java -> Installed JREs: Make sure default JRE is actually the JDK and not the JRE.
10. Preferences -> Server -> Installed Runtimes: Pick an Apache -> Tomcat 5.5 server, and point it to your local installation.
11. Import -> Existing Projects into Workspace: Point to your AppFuse project dir and import `myproject-core` and `myproject-web`.
12. Open J2EE perspective, then Run->Run On Server (when prompted, connect it to the Tomcat server you configured previously)
13. You may want to copy the launch configuration and then add all the properties defined in `myproject/pom.xml`.

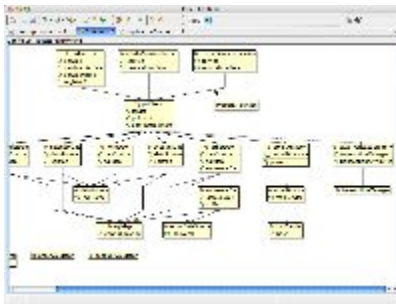
## Debugging

One of the nice things about AppFuse 2.0 is you shouldn't **have** to use Maven for the most part. It has much better IDE support now. You should be able to run most of your tests from within Eclipse. If you can run your tests, you should be able to **debug** them as well. If you write your tests so they don't depend on data already being in the database, you should be able to run your tests all day long. If you depend on pre-existing data, you may have to run `mvn dbunit:operation` (same as `ant db-load` in AppFuse 1.x) every so often, or use DbUnit's Java API to pre-populate your database.

To debug your application while running it in Jetty, see [Debugging with the Maven Jetty Plugin in Eclipse](#).

## Spring Support

[Spring IDE](#) is an excellent plugin to install for viewing, graphing and navigating Spring beans. [Installing it](#) should be fairly straight forward. Below is an example graph created with Spring IDE.



## Tips and Tricks

- **Ctrl+Shift+R** will allow you to find any file within your project. It's a handy way to open files quickly without having to navigate through your source tree.
  - **Ctrl+Shift+T** does the same, but restricted to Java source files (Java view only).
- **Ctrl+Shift+E** opens a dialog with the files you most recently open listed.

In order to clean up the project view in Eclipse, you can hide the files you don't need. First of all, make sure you're in the *Java Perspective* (Window -> Open Perspective). Then click the little (down) arrow in the top right corner of the *Package Explorer* pane. Select **Filters** and check **Libraries from external** and **Libraries from project**. Click OK to continue.

Another useful Eclipse trick is to use abbreviated package names. It's a nice feature on projects where you're inflicted with *super.long.package.name.syndrome*. Go to **Window -> Preferences -> Java -> Appearance**. Check the "Compress all package names" checkbox and type "1." (no quotes) in the text field.

## IDEA

[IntelliJ IDEA](#) is "The Most Intelligent Java IDE". If you're willing to spend a couple hundred to boost your productivity, it's a very enjoyable IDE to work with. Not only does it handle Maven 2 modular projects splendidly, it's great for developing web applications because of its superb JavaScript and CSS support.

## Table of Contents

1. [Download and Install](#)
2. [Spring Support](#)

### 3. Tips and Tricks

#### Download and Install

To make your project IDEA-aware, perform the following steps:

1. Download [IDEA 6.0](#) and install it in \$TOOLS\_HOME/idea-6.0.
2. In your project's directory, run **mvn idea:idea**. This will generate project files using the [Maven IDEA Plugin](#).
3. Open your project in IDEA by running "explorer projectname.ipr" on Windows or "open projectname.ipr" on OS X.



You should be able to compile your project and run tests from within your IDE. For tests that rely on pre-existing data, you may have to periodically run **mvn dbunit:operation** to re-populate your database. You shouldn't need to worry about deploying from Eclipse because you can use the Jetty Plugin (mvn jetty:run-war) or Cargo (mvn cargo:start -Dcargo.wait=true). You can probably use IDEA's app server support to auto-deploy your project. However, no tutorials currently existing for doing this. We'd love the contribution if you happen to create one.

If you'd like to run Maven from IDEA, use IDEA's plugins feature to install Maven 2 support.

- [Maven Reloaded](#): Inspects the project space, updates dependencies, source and test paths, allows execution of goals.
- [Maven-2 Integration](#): Requires a preinstalled Maven-2 and allows execution of standard phases and goals like the Ant plugin.

You can use both plugins together as they don't overlap that much. Thanks to the [Maven user mailing list](#) for this information.

#### Spring Support

IDEA supports code-completion when editing Spring context files. This is a very nice feature and you shouldn't need any additional Spring support. If you want more features, check out [IdeaSpring](#) (\$30).

#### Tips and Tricks

- Ctrl+Shift+N will allow you to find any file within your project. It's a handy way to open files quickly without having to navigate through your source tree.
- Ctrl+Shift+E opens a dialog with the files you most recently open listed.

### MyEclipse

At this time, [MyEclipse](#) doesn't support [Maven 2](#). Therefore, using MyEclipse should be the same as [using Eclipse](#).

### NetBeans

[NetBeans](#) 5.5+ should work great to develop an AppFuse-based application.

#### Table of Contents

1. [Download and Install](#)
2. [Spring Support](#)
3. [Tips and Tricks](#)

#### Download and Install

1. Download [NetBeans 5.5+](#) and install it in \$TOOLS\_HOME/netbeans-5.5.
2. Install [Mevenide](#) for Maven 2 support in NetBeans.
3. Go to **File > Open Project** and navigate to your project. Open it to begin using NetBeans to develop your project.



NetBeans has excellent Maven 2 support. You should be able to install and run your project in NetBean's built-in Tomcat instance.

You can also run Maven commands from within NetBeans (right-click on your project name and choose "Run lifecycle phase" or "Run Custom Goal". You should be able to compile your project and run tests from within your IDE. For tests that rely on pre-existing data, you may have to periodically run **mvn dbunit:operation** to re-populate your database. You don't have to use NetBean's built-in Tomcat instance because you can use the Jetty Plugin (mvn jetty:run-war) or Cargo (mvn cargo:start -Dcargo.wait=true).

#### Spring Support

Installing the [Spring Module for NetBeans](#) is recommended for simplifying Spring development with NetBeans.

#### Tips and Tricks

None yet, please add some!

## Licenses

Below is a list of the open source software used in AppFuse. Some libraries are only included by certain web and persistence frameworks. For example, Acegi JSF is only used when you're using JSF as your web framework.

AppFuse uses an [Apache 2.0](#) license.

Library	License	Commercial Use?
Acegi Security	<a href="#">Apache</a>	Allowed
Acegi JSF Components	<a href="#">Apache</a>	Allowed
ASM	<a href="#">ObjectWeb</a>	Allowed
AspectJ	<a href="#">Eclipse</a>	Allowed
Clickstream	<a href="#">OpenSymphony</a>	Allowed
Commons Lang	<a href="#">Apache</a>	Allowed
Commons Logging	<a href="#">Apache</a>	Allowed
Commons BeanUtils	<a href="#">Apache</a>	Allowed
Commons Collections	<a href="#">Apache</a>	Allowed
Commons DBCP	<a href="#">Apache</a>	Allowed
Commons FileUpload	<a href="#">Apache</a>	Allowed
Commons IO	<a href="#">Apache</a>	Allowed
Core JSF Validator	<a href="#">Apache</a>	Allowed
Display Tag	<a href="#">Artistic</a>	Allowed
DWR	<a href="#">Apache</a>	Allowed
EhCache	<a href="#">Apache</a>	Allowed
Facelets	<a href="#">CDDL</a>	Allowed
Hibernate	<a href="#">LGPL</a>	Allowed
iBATIS	<a href="#">Apache</a>	Allowed
JavaMail	<a href="#">CDDL</a>	Allowed
jMock	<a href="#">jMock Project License</a>	Allowed
JSTL	<a href="#">Apache</a>	Allowed
JUnit	<a href="#">CPL</a>	Allowed
Log4J	<a href="#">Apache</a>	Allowed
MyFaces	<a href="#">Apache</a>	Allowed
Jakarta ORO	<a href="#">Apache</a>	Allowed
OSCache	<a href="#">OpenSymphony</a>	Allowed
Scriptaculous	<a href="#">MIT</a>	Allowed
SiteMesh	<a href="#">OpenSymphony</a>	Allowed
Spring	<a href="#">Apache</a>	Allowed
Spring Modules	<a href="#">Apache</a>	Allowed
Struts	<a href="#">Apache</a>	Allowed
Struts Menu	<a href="#">Apache</a>	Allowed
Tapestry	<a href="#">Apache</a>	Allowed

The DHTML Calendar	LGPL	Allowed + <a href="#">Commercial Available</a>
URL Rewrite Filter	BSD	Allowed
Velocity	Apache	Allowed
WebTest	Canoo License	Allowed
Wiser	Apache	Allowed
XFire	XFire License	Allowed

## Maven 2



Knowledge of [Maven](#) is not required to use AppFuse because the tutorials explain how to use it. If you're interested in learning Maven in-depth, please see [Maven: The Definitive Guide](#).

For a brief introduction to Maven see [Building Web Applications with Maven 2](#) or [The Maven 2 POM demystified](#). Other articles are available from the [Articles on Maven](#) page.

See [Maven Plugins](#) for information on plugins used in AppFuse.

Below is a list of commonly-used helpful Maven commands.

Command	Description
<b>mvn clean</b>	Deletes all files in target directory or directories (for multi-module projects)
<b>mvn jetty:run-war</b>	Packages and deploys your application to Jetty, reachable at <a href="http://localhost:8080">http://localhost:8080</a>
<b>mvn -Dcargo.wait=true</b>	Packages and deploys your application to active Cargo profile (Tomcat 5.5.x by default), reachable at <a href="http://localhost:8080/yourapp-version">http://localhost:8080/yourapp-version</a>
<b>mvn test</b>	Runs all tests in <i>src/test/java</i> . Use "-Dtest=ClassName" (not fully-qualified) to run individual tests.
	Use <b>-Dsurefire.useFile=false</b> if you want to see test failures in your console (like Ant) and <b>-Dmaven.surefire.debug</b> if you want to open a debugger on port 5005 w/ suspend=y.
<b>mvn package</b>	Creates a WAR or JAR depending on your project type
<b>mvn integration-test</b>	Runs UI tests in Tomcat using Cargo
<b>mvn install</b>	Installs generated artifacts in your local repository
<b>mvn site</b>	Creates project site and reports
<b>mvn -U</b>	Checks for updated plugins and downloads if they exist
<b>mvn -o</b>	Work offline
<b>mvn --help</b>	See full list of optional commands

## Ant vs. Maven

If you used AppFuse 1.x in the past and have taken the time to memorize its useful [Ant commands](#), the following table should help you convert to using Maven.

Ant command	Maven command	Description
ant setup-db	mvn hibernate3:hbm2ddl dbunit:operation	Creates and populates your database
ant test-all	mvn integration-test	Runs unit and integration tests
ant clean	mvn clean	Deletes the target directory
ant compile	mvn compile	Compiles all source files
ant war	mvn package	Creates a WAR file
ant deploy	mvn jetty:run-war	Deploys a WAR to embedded instance of Jetty and starts it

ant test-dao	mvn test -Dtest=*DaoTest	Runs all DAO tests
ant test-service	mvn test -Dtest=*ManagerTest	Runs all Manager tests

## Installing a local Maven Repository

If you're using Maven 2 in your organization on multiple projects, it's often a good idea to setup a local Maven repository on your intranet. There's a few different strategies for doing this:

- [Artifactory](#) - a Maven2 proxy repository based on JSR-170. Started with maven-proxy's codebase. See [Setting up a Maven repository with Artifactory](#) for installation instructions.
- [Dead Simple Maven Proxy](#) - DSMP is a proxy. It can be used as a repository server but it's main purpuss it to act as a filter when Maven accesses the internet. If you need a repository server, use Archiva.
- [Maven Archiva](#) - still under development.
- [Maven Proxy](#) - See Sanjiv's blog entry titled [Using maven-proxy to setup an internal maven repository](#) for more information.
- [Proximity](#) - Proximity functions somewhere between http-proxy and proactive-mirror. One of its primary uses is as Java web application to serve as a Maven proxy on your company's intranet.

## Troubleshooting and other Tips

The tips below are copied from Mule's [Tips and Tricks with Maven 2.x](#). Thanks guys. 😊



### OutOfMemoryError

If you are getting `OutOfMemoryError` exceptions when attempting a full build of Mule you may try increasing the max heap **and the PermGen space** sizes. Either export a `MAVEN_OPTS` variable in your shell or add it to the original mvn script. The following settings were confirmed working fine with Mule:

```
MAVEN_OPTS=-Xmx512m -XX:MaxPermSize=256m
```



### Offline option

If you know your downloads are up-to-date, you can use the offline option which may speed up your build considerably depending on network latency:

```
mvn -o
```



### Debug option

Transitive dependencies in m2 are both a blessing and a curse. The debug option is especially helpful when you find yourself in "classloader hell" (conflicting library versions or unwanted libraries are in your classpath). Among other things, it prints out the effective classpath in a tree format which makes it easy to see where each libraries is coming from.

```
mvn -X
```



### Non-recursive option

Executing a Maven goal for a module will by default execute the goal for any modules below it in the hierarchy. If you specifically wish to run the goal for this module only (and not its children) you can use the non-recursive option.

```
mvn -N
```



### Debugging test failures

Surefire, which is the default Maven test runner now, outputs all reports as a set of xml and text files. Any test failure details and stacktraces end up in those files, and not the console. The output can be redirected to the console temporarily by adding the following option

```
mvn -Dsfire.useFile=false
```

This option will skip creating the text report, but the xml one will still be available for transformation by tools.

## Maven Plugins

There are a number of Maven Plugins that AppFuse uses to simplify the development process (listed in alphabetical order).



- **AppFuse Maven Plugin:** This plugin is a replacement for AppGen in AppFuse 1.x. It's used for code generation, as well as other AppFuse-related tasks.
- **Cargo Maven Plugin:** Used to start and stop containers before and after running integration tests (i.e. with Canoo WebTest or Selenium).
- **DbUnit Maven Plugin:** This plugin is a wrapper around **DbUnit** and is used to populate the database with sample data before running tests.
- **Hibernate3 Maven Plugin:** Used to create the database schema when using Hibernate or JPA for the DAO framework.
- **Maven Jetty Plugin:** Used to run an embedded Jetty instance from Maven.
- **Maven WarPath Plugin:** Used to provide features to Maven so WARs can have dependencies.
- **Native2Ascii Maven Plugin:** Converts i18n bundles from native to ascii format.
- **SQL Maven Plugin:** Used to create the database schema when using iBATIS for the DAO framework.
- **Tomcat Maven Plugin:** Similar to the Maven Jetty Plugin - can start an embedded Tomcat instance to deploy your project or manage a remote instance.

For more plugins, see [available Maven plugins](#) and the [Maven 2 Plugins Project](#) (also known as *Mojo*).

## OS Specific Issues

While an AppFuse app can be run on any system that supports the Java requirements there are occasionally issues that effect only certain operating systems. Because these are not "bugs" in AppFuse they are not entered into the [issue tracker](#). Instead they are listed below for reference.

### Ubuntu 7.10

- Many of the directories in `/var/lib/tomcat5.5` are symlinks and in some conditions I haven't quite figured out the owner of these directories or files within are changed. This causes Tomcat to fail to start. To fix this you can run the following command as `root`:  
`chown -R tomcat55:nogroup /var/lib/tomcat5.5`
- Java security is enabled by default in Tomcat. Which makes sense for a production server, but can be quite a pain when developing. To disable Java security in Tomcat edit the following property in `/etc/default/tomcat5.5`:  
`TOMCAT5_SECURITY=no`
- XML parsing is configured incorrectly in a fresh installation of Tomcat on Ubuntu 7.10. So if you are having a hard time parsing those XML config files, remove the following symlink:  
`/usr/share/tomcat5.5/common/endorsed/xml-apis.jar`  
*Thanks to CODESPIN for this one, I forgot about it until finding his blog on the issue.*  
<http://codespin.blogspot.com/2008/01/fixing-common-tomcat-55-problems-on.html>
- The packaged Tomcat 5.5 that comes with Ubuntu 7.10 does not have a catalina.out pipe or other console output. The way to get around this problem is to modify the `src/main/resources/log4j.xml` by replacing the "CONSOLE" appender with a "FILE" appender.

```

<!-- Commented out because we are using a file appender instead -->
<!--
<appender name="CONSOLE" class="org.apache.log4j.ConsoleAppender">
  <layout class="org.apache.log4j.PatternLayout">
    <param name="ConversionPattern"
      value="[myapp] %p [%t] %c{1}:%M(%L) | %m%n"/>
  </layout>
</appender>
-->
<appender name="FILE" class="org.apache.log4j.RollingFileAppender">
  <param name="File" value="/var/log/tomcat5.5/cbeyond.log"/>
  <param name="Append" value="true"/>
  <param name="MaxFileSize" value="1000KB"/>
  <param name="maxBackupIndex" value="5"/>
  <layout class="org.apache.log4j.PatternLayout">
    <param name="ConversionPattern"
      value="[myapp] %p [%t] %c{1}:%M(%L) | %m%n"/>
  </layout>
</appender>
....
<root>
  <level value="WARN"/>
  <appender-ref ref="FILE"/>
</root>

```

## Persistence Frameworks

Persistence frameworks are used in AppFuse to simplify database access code. In addition to simplifying the code you have to write, persistence frameworks offer advantages over raw JDBC. Advantages include: caching, query optimization and developer productivity. Three persistence frameworks are available within AppFuse:

- [Hibernate](#)
- [iBATIS](#)
- [JPA](#)

Currently, the JPA implementation uses Hibernate.

## Hibernate

[Hibernate](#) is an open-source Object/Relational Mapping (ORM) solution. ORM is the technique of mapping an Object model to a Relational model (usually represented by a SQL database). Hibernate was created in late 2001 by Gavin King and a handful of other developers. Since then, Hibernate has become a very popular persistence framework in the Java Community. It's become so popular in fact, that the next versions of EJB and JDO are using Hibernate as a source of good ideas. The reasons for its popularity are mainly due to its good documentation, ease of use, excellent features and [smart project management](#). Hibernate's license is [LGPL](#), which means you can use it and distribute as long as you don't modify its source code. More information on its license is available on the [Hibernate website](#).

Hibernate frees you from hand-coding JDBC. Rather than using SQL and JDBC, you can use domain objects (which are usually POJOs) and use annotations to map that object to a relational database. Annotations indicate which fields (in an object) map to which columns (in a table). Hibernate has a powerful query language called Hibernate Query Language (HQL). This language allows you to write SQL, but also use object-oriented semantics. One of the best parts about its query language is you can literally guess at its syntax and get it right.

Hibernate's [Session interface](#) is similar to a database connection in that it must be opened and closed at appropriate times to avoid errors and memory leaks. In my opinion, the biggest advantage of using Spring with Hibernate is that you don't have to manage opening and closing these Sessions - everything just works.

## Learn More

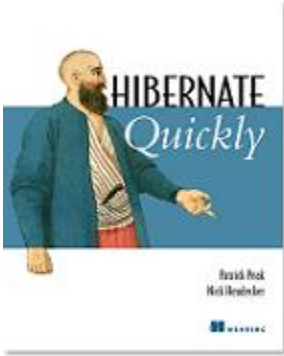
Hibernate has quickly become the de-facto standard for doing persistence in Java. This has resulted in a number of books about the project. While you shouldn't need to buy any books, there are a couple of good ones available:

- [An Introduction to Hibernate 3 Annotations](#) - an introductory article from OnJava.
- [Hibernate Quickly](#) - an excellent book that gets you up and running quickly with Hibernate.
- [Java Persistence with Hibernate](#) - in-depth guide to how Hibernate, JPA and EJB 3 work.





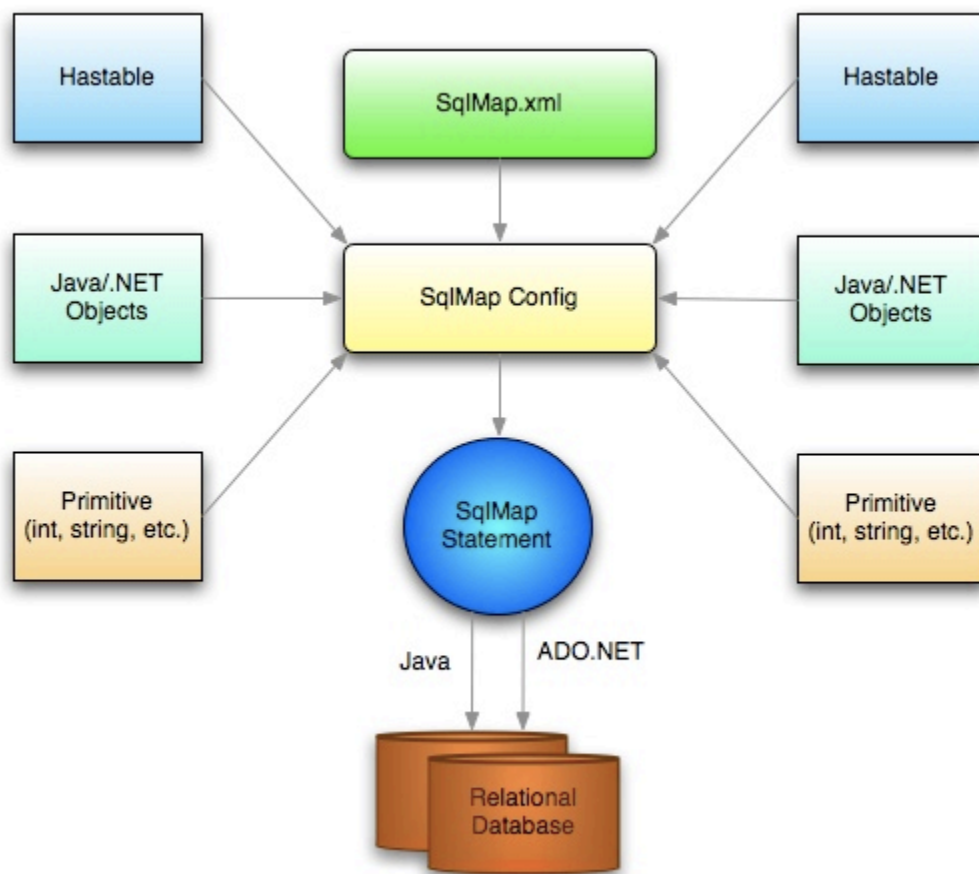
Hibernate's [Reference Documentation](#) is also a great source of information.



## iBATIS

The [iBATIS Data Mapper framework](#) is an open-source persistence framework that allows you to use your model objects with a relational database. In contrast to Hibernate, you write SQL, much like you would with JDBC. You do this in a very simple XML file, allowing abstraction of SQL from Java classes. iBATIS is not an O/R Mapper (ORM). Rather, it is a *Data Mapper*. In Martin's Fowler's [Patterns of Enterprise Application Architecture](#), he describes two patterns: [Data Mapper](#) and [Metadata Mapping](#). The difference is that ORMs (Metadata Mappers) map classes to tables; iBATIS (Data Mapper) maps inputs and outputs to an interface (for example, SQL interface to an RDBMS). An ORM solution works well when you have control of your database. A Data Mapper like iBATIS works well when the database is heavily normalized and you need to pull from several tables to populate an object.

The figure below shows iBATIS's architecture.



iBATIS is the name of a open-source project started by Clinton Begin in 2001. Clinton had a few products, but none of them gained much recognition until the .NET Pet Shop was released, claiming that .NET was superior to Java in developer productivity and performance. Microsoft

published a paper claiming that .NET's version of Sun's PetStore was [10 times faster and 4 times more productive](#). Knowing this wasn't the case, Clinton responded with JPetStore 1.0 in July 2002. Not only did this application have fewer lines of code and a better design than its .NET counterpart, but Clinton implemented it over a few weeks in his spare time!

Clinton's goals while writing JPetStore were to argue the points of good design, code quality, and productivity. The original .NET Pet Shop had a horrible design with much of the business logic contained in stored procedures, whereas JPetStore had a clean and efficient persistence framework.

This framework quickly drew the attention of the open-source community. Today, iBATIS refers to the "iBATIS Database Layer," which consists of a DAO framework and a SQL Map framework. Spring supports the iBATIS SQL Maps by providing helper classes to easily configure and use them. Furthermore, the Spring project includes JPetStore as one of its sample applications, rewriting many of its pieces to use Spring features.

iBATIS is a "sleepier project" in the open-source community. Not many folks know about it, but those who do, really like it. iBATIS's license is [Apache](#), which means you can use it freely as long as your end-user documentation states that your product contains software developed by the Apache Software Foundation. You can modify the code, but then you can no longer distribute it under the Apache name without permission.

iBATIS is an excellent persistence framework to use with existing or legacy databases. You can easily migrate a JDBC-based application to iBATIS (most of the work involves extracting the SQL out of Java classes and into Java files). Not only is iBATIS fast and efficient, but it doesn't hide SQL, which is one of the most powerful and oldest languages today. Using iBATIS's SQL Maps, developers write SQL in XML files and populates objects based on the results of those queries. Much like the Spring/Hibernate combination, iBATIS DAOs require very few lines of code in each method.

In my experience, I've found the following qualities to be true of iBATIS:

- Easy to learn
- Queries are extremely efficient
- Easy transition because of pre-existing SQL
- Just as fast (if not faster) than Hibernate
- Writing iBATIS DAOs is similar to writing Hibernate DAOs

## Learn More

The following is a blog post from the homepage of the [iBATIS](#) project.

### iBATIS in Action Released!

(January 26, 2007) A book for iBATIS? No way! Yes way. The book is here. This is the first edition, targeting primarily the Java platform. But it's definitely useful for .NET and even Ruby users, even if only to understand the concepts and ideas behind iBATIS. Everything else is just syntax. Some people may wonder if we're just trying to make money by selling documentation (sounds familiar to some no doubt). But trust us, writing technical books is NO way to make money! We did this for you, so you'd have a definitive guide to read from instead of wading through

various internet resources. I think we're more worried about the Amazon.com comments than the sales...  I (Clinton) really want to say what a fantastic experience it was to work with Brandon and Larry on this project. You guys did a fantastic job.

» [Buy the book in print or PDF](#)

## JPA

Not much here yet, maybe reading the [Java Persistence API FAQ](#) will help. 

If you'd like to help write this page, please let us know on the [mailing list](#).

## Security

AppFuse was originally developed as part of a sample application for a book I wrote for Apress, [Pro JSP](#). This sample application demonstrated many security features and features for simplifying Struts development. Many of the security features in this application did not exist in J2EE's security paradigm. Authentication using container-managed authentication (CMA) was easy, but Remember Me, password hints, SSL switching, signup, and user management were nonexistent. Furthermore, the ability to protect methods based on roles was not possible in a non-EJB environment.

At first, AppFuse implemented all of these features with its own code and workarounds for CMA. I'd heard about [Acegi Security](#) when I first started learning Spring in early 2004. I compared the number of lines of XML required by Acegi (175) with the number that CMA required in web.xml (20) and quickly dismissed Acegi as too complicated.

A year and a half later – and after writing a chapter about using Acegi Security for another book, [Spring Live](#) – I had changed my mind. Acegi *did* (and still does) require a fair amount of XML, but it really is quite simple once you understand it. When we finally took the plunge and replaced all AppFuse's home-grown features with Acegi Security's features, we ended up deleting a lot of code. Classes upon classes went away, disappearing into the "Acegi handles that now" pile in CVS's Attic.

Acegi Security is simply the best thing that's ever happened to J2EE's security model. It allows you to implement many useful features that aren't part of the Servlet API's security model: authentication, authorization, role-protected methods, Remember Me, password encryption, SSL switching, user switching, and logout. It also allows you to store your user's credentials in an XML file, in a database, in LDAP, or in a single

sign-on system such as Yale's Central Authentication Service (CAS) or SiteMinder.

AppFuse's implementation of many security-related features was nice in the beginning. Now that AppFuse uses Acegi Security, these features – and many more – are easy to implement. Acegi has many points for extension: that is the reason for its large XML configuration file. As we've integrated Acegi over the course of the last year, we've found that we've customized many bean definitions to hook into AppFuse more closely.

The combined ease of development, easily testable code, and loose coupling provided by the Spring IoC container and Acegi Security are the primary reasons that AppFuse is such a pleasure to develop with. These frameworks are nonintrusive and allow clean, testable code. AppFuse integrates many open source projects, and dependency injection allows easy integration of your application's layers.

## Apply Security to Managers and DAOs

You can secure methods on a per-role basis in security.xml. This file is merged into your WEB-INF directory from AppFuse's common-web project. However, you can override it by copying it into your own project's *src/main/webapp/WEB-INF* directory. Here is the current code you'll need to modify to security more beans:

```
<!-- Apply method-level interceptor to userManager bean -->
<aop:config>
  <aop:advisor id="managerSecurity" advice-ref="methodSecurityInterceptor"
    pointcut="execution(* org.appfuse.service.UserManager.*(..))"/>
</aop:config>

<bean id="methodSecurityInterceptor" class=
"org.acegisecurity.intercept.method.aopalliance.MethodSecurityInterceptor">
  <property name="authenticationManager" ref="authenticationManager"/>
  <property name="accessDecisionManager" ref="accessDecisionManager"/>
  <property name="objectDefinitionSource">
    <value>
      org.appfuse.service.UserManager.getUsers=admin
      org.appfuse.service.UserManager.removeUser=admin
    </value>
  </property>
</bean>
```

The easiest way to copy the security.xml file into your project is:

1. Run **mvn package**.
2. Copy *target/yourproject-version/WEB-INF/security.xml* to *src/main/webapp/WEB-INF*.

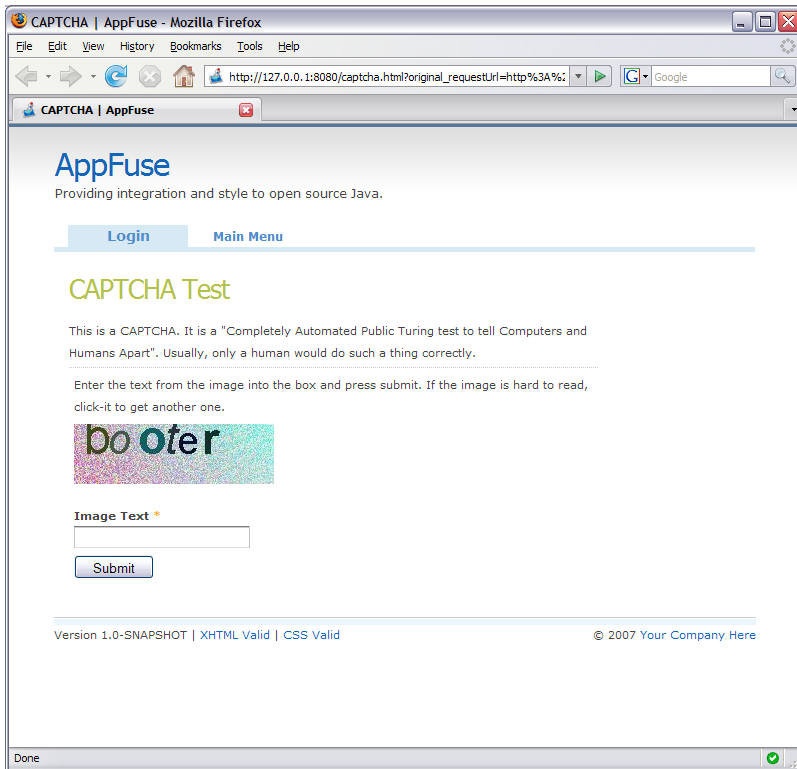
## CAPTCHA Integration

### Introduction

This document provides instructions on adding [CAPTCHA](#) support to an AppFuse project. It is based on AppFuse 2.0-M4 which you can find more about by reading the [Release Notes 2.0 M4](#).

Addition of [CAPTCHA](#) is as simple as unzipping the [appfuse-captcha-2.0-M4.zip](#) file into your "myproject" directory, and editing one pom.xml file.

Here's a screenshot of the final product (click to enlarge):



### **Versions Used**

- AppFuse 2.0-M4
- Maven 2.0.5
- jCaptcha 1.0-RC4

### **Resources**

Download the [appfuse-captcha-2.0-M4.zip](#) sources, which includes these instructions.

General AppFuse support is available at the <http://appfuse.org/> website. Support for jCaptcha can be found at the <http://jcaptcha.sourceforge.net/> website.

### **Assumptions**

This integration guide is based on the [AppFuse QuickStart](#) guide, using a Struts 2 Modular project. It's assumed that you will create an AppFuse project called "myproject" with some package "com.mycompany". It's also assumed that you've followed the AppFuse QuickStart guide and are running MySQL already.

This will work for an AppFuse Struts 2 Modular project archetype. Other archetypes aren't supported, but the included files should be easily adaptable based on your requirements. It's assumed that you've just created a new project using Maven, and haven't edited or modified any of the base files. (If you've already started developing your Struts 2 Modular project, check everything into a source control management system to manage changes with this feature.

It's a good idea to check all of your files into source control before unzipping appfuse-captcha-2.0-M4.zip into your project so you can see what files changed, and how they changed. A list of files modified and added can be found below.

### **Conventions**

Change these mnemonics based on your needs:

DEV\_HOME = your base development directory, where all of your projects live

com.mycompany = your package name

myproject = your project name

### **Verifying your Build**

Create a modular Struts2 based AppFuse project using:

```
cd $DEV_HOME
mvn archetype:create -DarchetypeGroupId=org.appfuse \
-DarchetypeArtifactId=appfuse-modular-struts \
-DremoteRepositories=http://static.appfuse.org/repository \
-DarchetypeVersion=2.0-M4 \
-DgroupId=com.mycompany \
-DartifactId=myproject
```

Test that your project builds using:

```
cd $DEV_HOME/myproject
mvn install
```

Unpack the AppFuse war into your new project, and test that the webapp runs:

```
cd $DEV_HOME/myproject/web
mvn war:inplace
mvn jetty:run-war
```

Open a web browser and use the Signup page to create a new user account. Everything should work at this point, use Ctrl-C to stop Jetty. If everything builds correctly, you can continue, otherwise, consult the <http://www.appfuse.org> website for how to fix it.

Clean things up before adding CAPTCHA support:

```
cd $DEV_HOME/myproject/web
mvn clean
```

Note: it is important to run the "maven clean" goal, otherwise the web app will use the old files in the /target directory without picking up the changes you are about to make.

### **Adding CAPTCHA**

Add a new dependency in the pom.xml file for jCaptcha. Edit the \$DEV\_HOME/myproject/web/pom.xml file, adding (at about line 200):

```
<dependency>
  <groupId>com.octo.captcha</groupId>
  <artifactId>jcaptcha-all</artifactId>
  <version>1.0-RC4</version>
  <scope>compile</scope>
</dependency>
```

Unpack this zip file's contents into your Struts 2 Modular base directory.

```
cd $DEV_HOME/myproject
unzip appfuse-captcha-2.0-M4.zip
```

Run Jetty again:

```
cd $DEV_HOME/myproject/web
mvn clean jetty:run-war
```

Open firefox and browse to the signup page again: <http://127.0.0.1:8080/signup.html> You should be greeted with a CAPTCHA challenge. If so, congratulations, CAPTCHA has been added to your project!

### **Known Issues**

1. The Canoo Webtest for the Signup page fails. Running the web integration-tests using Maven will fail because the Signup page is made inaccessible due to the addition of the CAPTCHA. Is this a feature, or a bug? It's an exercise for the reader to fix this webtest by editing

the \$DEV\_HOME/web/src/test/resources/web-tests.xml file. Maven will result in errors when these commands are run:

```
cd $DEV_HOME/myproject/web
mvn integration-test
#or simply:
mvn
```

## Appendix

Here are the details of the changes that are made, excluding pom.xml modifications:

```
These files are modified:
M web/src/main/resources/ApplicationResources.properties
M web/src/main/resources/struts.xml
M web/src/main/webapp/WEB-INF/applicationContext-struts.xml
M web/src/main/webapp/WEB-INF/security.xml
M web/src/main/webapp/WEB-INF/web.xml

These files are added:
A web/src/main/java/org/appfuse/webapp/JCaptchaServiceProxyImpl.java
A web/src/main/java/org/appfuse/webapp/action/CaptchaAction.java
A web/src/main/java/org/appfuse/webapp/servlet/ImageCaptchaServlet.java
A web/src/main/resources/org/appfuse/webapp/action/CaptchaAction-validation.xml
A web/src/main/webapp/WEB-INF/applicationContext-captcha.xml
A web/src/main/webapp/WEB-INF/pages/captcha.jsp
```

## CAPTCHA Integration for Spring Modules

### Introduction

JCAPTCHA, for Java Completely Automated Public Test to tell Computers and Humans Apart is an open source project where assures the preventing the automated systems to fill data in web applications with junk data. The idea is to have a human command to enter before submitting each web form entry. This document provides instructions on adding CAPTCHA support to spring module of an AppFuse project. It is based on AppFuse 2.0.1 which you can find more about by reading the Release Notes 2.0.1

The CAPTCHA integration for the appfuse struts module can be found [here](#).

Addition of CAPTCHA is as simple as unzipping the appfuse-captcha-2.0-M4.zip file into your "myproject" directory, and editing one pom.xml file.

### Versions Used

- 1.Appfuse 2.0.1
- 2.Maven 2.0.6
- 3.JCAPTCHA 1.0-RC6

### Resources

Download the appfuse-captcha-2.0-M4.zip sources, which includes these instructions.

General AppFuse support is available at the <http://appfuse.org/> website. Support for jCaptcha can be found at the <http://jcaptcha.sourceforge.net/> website.

### Assumptions

This integration guide is based on the AppFuse QuickStart guide, using a Spring Modular project. It's assumed that you will create an AppFuse project called "myproject" with some package "com.mycompany". It's also assumed that you've followed the AppFuse QuickStart guide and are running MySQL already.

This will work for an AppFuse Spring Modular project archetype. This has been integrated in struts project but the included files should be easily adaptable based on your requirements for any archy type if you can collect the basic integration idea. It's assumed that you've just created a new project using Maven, and haven't edited or modified any of the base files. (If you've already started developing your Spring Modular project, check everything into a source control management system to manage changes with this feature.

It's a good idea to check all of your files into source control before unzipping appfuse-captcha-2.0-M4.zip into your project so you can see what files changed, and how they changed. A list of files modified and added can be found below.

### Conventions

### Verifying your Build

## Adding Captcha

# Configuring the Menu System

## Database Encryption with Jasypt-Hibernate

### About this Tutorial

This tutorial will show you how to use [Jasypt](#) hibernate integration to transparently encrypt database columns.

Jasypt is a java library which allows developers to add basic encryption capabilities to their projects with minimum effort, and without the need of having deep knowledge on how cryptography works.

- High-security, standards-based encryption techniques, both for unidirectional and bidirectional encryption. Encrypt passwords, texts, numbers, binaries...
- Transparent integration with Hibernate.
- Suitable for integration into Spring-based applications and also transparently integrable with ACEGI (Spring Security).

Jasypt provides an integration package (*org.jasypt.hibernate.type*) which provides several Hibernate UserType implementations to allow one or several of the properties in a mapped Hibernate entity to be declared as being of an encrypted type. Types allowed to be stored encrypted include strings, binaries (byte arrays), numeric types, booleans and dates.

Persistence of those properties will be done in an encrypted manner, but in a completely transparent way for the application.

This can be useful for encrypting personal data, private messages, etc, so that it is avoided that anyone with read access to the "critical" tables can read all of its contents.

But encryption sets some **limitations** on your Hibernate usage:

- encrypted properties can not be used as part of projection queries (SUM, MAX, ..) as this is done on the database before decryption by hibernate.
- encrypted properties can not also be part of an ORDER BY clause.
- for encrypted properties to appear on a WHERE clause, their encryptor must be configured to use a fixed *SaltGenerator*

Please see [Jasypt](#) for more information.



This tutorial assumes you are using Hibernate as your persistence framework and have already completed the [Persistence](#) and [Using Hibernate](#) tutorials.

### Table of Contents

1. [Adding Jasypt as a dependency to your project](#)
2. [Defining hibernate encrypted Types](#)
3. [Registering Jasypt encryptors as Spring beans](#)
4. [Annotating encrypted properties](#)
5. [Setting encryption password as environment variable](#)

### Adding Jasypt as a dependency to your project

First is adding Jasypt library as a dependency to your project. For that you need to locate `<dependencies />` section on your pom.xml. Once located add the following lines:

```

<dependency>
  <groupId>org.jasypt</groupId>
  <artifactId>jasypt</artifactId>
  <version>1.2</version>
</dependency>

<dependency>
  <groupId>commons-codec</groupId>
  <artifactId>commons-codec</artifactId>
  <version>1.3</version>
</dependency>
<dependency>
  <groupId>commons-lang</groupId>
  <artifactId>commons-lang</artifactId>
  <version>2.3</version>
</dependency>

```

## Defining hibernate encrypted Types

Jasypt uses custom Hibernate *UserTypes* to provide transparent encryption of your data. For this you can create `src/main/resources/jasyptHibernateTypes.hbm.xml` and add the following configuration:

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

<hibernate-mapping>

  <!-- VARCHAR, CLOB, TEXT based types -->

  <typedef name="encryptedString" class="org.jasypt.hibernate.type.EncryptedStringType">
    <param name="encryptorRegisteredName">jasyptHibernateEncryptor</param>
  </typedef>

  <typedef name="encryptedBigDecimalAsString" class=
"org.jasypt.hibernate.type.EncryptedBigDecimalAsStringType">
    <param name="encryptorRegisteredName">jasyptHibernateEncryptor</param>
  </typedef>

  <typedef name="encryptedBigIntegerAsString" class=
"org.jasypt.hibernate.type.EncryptedBigIntegerAsStringType">
    <param name="encryptorRegisteredName">jasyptHibernateEncryptor</param>
  </typedef>

  <typedef name="encryptedBooleanAsString" class=
"org.jasypt.hibernate.type.EncryptedBooleanAsStringType">
    <param name="encryptorRegisteredName">jasyptHibernateEncryptor</param>
  </typedef>

  <typedef name="encryptedByteAsString" class="org.jasypt.hibernate.type.EncryptedByteAsStringType">
    <param name="encryptorRegisteredName">jasyptHibernateEncryptor</param>
  </typedef>

  <typedef name="encryptedCalendarAsString" class=
"org.jasypt.hibernate.type.EncryptedCalendarAsStringType">
    <param name="encryptorRegisteredName">jasyptHibernateEncryptor</param>
  </typedef>

  <typedef name="encryptedDateAsString" class="org.jasypt.hibernate.type.EncryptedDateAsStringType">
    <param name="encryptorRegisteredName">jasyptHibernateEncryptor</param>
  </typedef>

  <typedef name="encryptedDoubleAsString" class=
"org.jasypt.hibernate.type.EncryptedDoubleAsStringType">

```



```
<param name="encryptorRegisteredName">jasyptHibernateEncryptor</param>
</typedef>

<typedef name="encryptedFloatAsString" class=
"org.jasypt.hibernate.type.EncryptedFloatAsStringType">
<param name="encryptorRegisteredName">jasyptHibernateEncryptor</param>
</typedef>

<typedef name="encryptedIntegerAsString" class=
"org.jasypt.hibernate.type.EncryptedIntegerAsStringType">
<param name="encryptorRegisteredName">jasyptHibernateEncryptor</param>
</typedef>

<typedef name="encryptedLongAsString" class="org.jasypt.hibernate.type.EncryptedLongAsStringType">
<param name="encryptorRegisteredName">jasyptHibernateEncryptor</param>
</typedef>

<typedef name="encryptedShortAsString" class=
"org.jasypt.hibernate.type.EncryptedShortAsStringType">
<param name="encryptorRegisteredName">jasyptHibernateEncryptor</param>
</typedef>

<!-- VARBINARY, BLOB based type -->
<typedef name="encryptedBinary" class="org.jasypt.hibernate.type.EncryptedBinaryType">
<param name="encryptorRegisteredName">jasyptByteHibernateEncryptor</param>
</typedef>

<!-- NUMERIC, NUMBER based types -->
<typedef name="encryptedBigDecimal" class="org.jasypt.hibernate.type.EncryptedBigDecimalType">
<param name="encryptorRegisteredName">jasyptBigDecimalHibernateEncryptor</param>
</typedef>

<typedef name="encryptedBigInteger" class="org.jasypt.hibernate.type.EncryptedBigIntegerType">
<param name="encryptorRegisteredName">jasyptBigIntegerHibernateEncryptor</param>
</typedef>
```

```
</hibernate-mapping>
```

Open `src/main/resources/hibernate.cfg.xml` for the basic archetypes (or `core/src/main/resources/hibernate.cfg.xml` for the modular archetypes) and register this file with the following XML:

```
<mapping resource="jasyptHibernateTypes.hbm.xml"/>
```

This sample file includes all encrypted types supported by Jasypt 1.2, you can safely remove those you are not planning to use. It also assumes that you are planning to use a single String encryptor shared between VARCHAR based types, you will register on the next section.

## Registering Jasypt encryptors as Spring beans

Now you'll create Jasypt encryptors that will be used by hibernate encrypted *UserTypes* you just defined on previous section. Open `src/main/webapp/WEB-INF/applicationContext.xml` (or `core/src/main/resources/applicationContext.xml` for a modular archetype) and add the following to it:

```
<bean id="stringEncryptor" class="org.jasypt.encryption.pbe.StandardPBEStrEncryptor" lazy-init="false">
    <property name="algorithm" value="PBEWithMD5AndDES" />
    <property name="password" value="${jasypt_password}" />
</bean>

<bean id="hibernateEncryptor" class="org.jasypt.hibernate.encryptor.HibernatePBEStrEncryptor"
lazy-init="false">
    <!-- This property value must match "encryptorRegisteredName" used when defining hibernate user
types -->
    <property name="registeredName" value="jasyptHibernateEncryptor" />
    <property name="encryptor" ref="stringEncryptor" />
</bean>
```

We leave encryption password as a placeholder as it will be replaced, at Spring context load time, from an environment variable. Notice that we set `lazy-init="false"` to force the beans to get loaded. These beans need to be loaded at startup to register the encryptor.

## Annotating encrypted properties

Next thing to do is annotate persistent properties with *org.hibernate.annotations.Type* annotation and the proper encrypted type. For this we will use on this tutorial a sample object holding 3 properties, of types String, Long and Date, that we want Jasypt to transparently encrypt when persisting them to the database:

```
package org.appfuse.tutorial.model;

import java.io.Serializable;
import java.util.Date;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Table;

import org.hibernate.annotations.Type;

@Entity
@Table(name="sensible_personal_data")
public class SensiblePersonalData implements Serializable{

    private Long id;
    private String name;
    private String email;
    private Long favoriteNumber;
    private Date dateOfBirth;
```

```
@Id @GeneratedValue(strategy=GenerationType.AUTO)
public Long getId() {
    return id;
}

public String getName() {
    return name;
}

@Column(name="email", length=50)
@Type(type="encryptedString")
public String getEmail() {
    return email;
}

@Column(name="favorite_number", length=50)
@Type(type="encryptedLongAsString")
public Long getFavoriteNumber() {
    return favoriteNumber;
}

@Column(name="dob", length=50)
@Type(type="encryptedDateAsString")
public Date getDateOfBirth() {
    return dateOfBirth;
}

public void setName(String name) {
    this.name = name;
}

public void setDateOfBirth(Date dateOfBirthDate) {
    this.dateOfBirth = dateOfBirthDate;
}

public void setEmail(String email) {
    this.email = email;
}

public void setFavoriteNumber(Long favoriteNumber) {
    this.favoriteNumber = favoriteNumber;
}

public void setId(Long id) {
    this.id = id;
}
```

```
}
```

You need to add this class to hibernate.cfg.xml as usual. (See [Persistence](#) tutorial)

### Setting encryption password as environment variable

As keeping encryption password stored on a text file inside the application is not a good practice, we will set it through an environment variable.

By default Spring *PropertyPlaceholderConfigurer* (configured in *applicationContext-resources.xml*) can resolve placeholder properties from environment variables so all we have to do is set it just before starting the application:

For unix folks:

```
export jasypt_password="thisIsMySecretPassword"
mvn jetty:run-war
```

For windows users:

```
set jasypt_password="thisIsMySecretPassword"
mvn jetty:run-war
```

And that's all!!

## LDAP Authentication

---

### Table of content

- [AppFuse 2.1.x + Spring Security 3.x](#)
  - [AppFuse 2.0.x + Spring Security 2.x](#)
  - [AppFuse 1.9.4 + Acegi Security](#)
- 

### AppFuse 2.1.x + Spring Security 3.x

With AppFuse 2.1.x comes a new version of Spring Security, exactly, the 3.0.4.RELEASE, and the working mode has changed since 2.x. This means that the latest solution to work with LDAP doesn't work.

Here you can find the new approach.

#### Adding dependencies to the pom.xml

In the pom.xml, add the following dependencies

```
<dependency>
  <groupId>org.springframework.security</groupId>
  <artifactId>spring-security-ldap</artifactId>
  <version>${spring.security.version}</version>
</dependency>
<dependency>
  <groupId>org.springframework.ldap</groupId>
  <artifactId>spring-ldap-core</artifactId>
  <version>${spring.ldap.version}</version>
</dependency>
<dependency>
  <groupId>org.springframework.ldap</groupId>
  <artifactId>spring-ldap-core-tiger</artifactId>
  <version>${spring.ldap.version}</version>
</dependency>
```

And the corresponding variable at the end:

```
<spring.ldap.version>1.3.1.RELEASE</spring.ldap.version>
```

### ***Configuring the security.xml file***

The default userDao authentication provider

You can comment it to have only one type of authentication, or you can leave it to have a chaining auth, first LDAP and after the userDao validation.

```
<authentication-manager>
  <authentication-provider user-service-ref="userDao">
    <password-encoder ref="passwordEncoder" />
  </authentication-provider>

  <authentication-provider ref="ldapAuthProvider" />
</authentication-manager>
```

Add the LDAP server

```
<ldap-server id="ldapServer"
  url="ldap://localhost:389/dc=example,dc=com";
manager-dn="cn=Manager,dc=example,dc=com"
manager-password="pass" />
```

If you don't specify the manager-dn and manager-password the connection will be anonymous.

The authorization with custom database provider

You need to add the following beans:

```

    <beans:bean id="ldapAuthProvider" class=
"org.springframework.security.ldap.authentication.LdapAuthenticationProvider">
    <beans:constructor-arg ref="ldapBindAuthenticator"/>
    <beans:constructor-arg ref="ldapAuthoritiesPopulator"/>
    <!-- beans:property name="userDetailsContextMapper" ref="ldapUserDetailsContextMapper"/ -->
</beans:bean>

<beans:bean id="ldapBindAuthenticator" class=
"org.springframework.security.ldap.authentication.BindAuthenticator">
    <beans:constructor-arg ref="ldapServer"/>
    <beans:property name="userSearch" ref="ldapSearchBean"/>
</beans:bean>

<beans:bean id="ldapSearchBean" class=
"org.springframework.security.ldap.search.FilterBasedLdapUserSearch">
    <beans:constructor-arg value=""/><!-- user-search-base -->
    <beans:constructor-arg value="(uid={0})"/> <!-- user-search-filter -->
    <beans:constructor-arg ref="ldapServer"/>
</beans:bean>

    <beans:bean id="cppAuthoritiesUserDetailsServiceImpl" class=
"cat.urv.cpp.webapp.util.CppAuthoritiesUserDetailsServiceImpl">
        <beans:constructor-arg ref="userDao"/>
    </beans:bean>

<beans:bean id="ldapAuthoritiesPopulator"
class="org.springframework.security.ldap.authentication.UserDetailsServiceLdapAuthoritiesPopulator">
    <beans:constructor-arg ref="cppAuthoritiesUserDetailsServiceImpl"/>
</beans:bean>

```

With the class CppAuthoritiesUserDetailsServiceImpl you indicates what role have a user. The source code of this class is:

```

package cat.urv.cpp.webapp.util;

import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.dao.DataAccessException;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.core.userdetails.UsernameNotFoundException;

import cat.urv.cpp.dao.UserDao;
import cat.urv.cpp.model.User;

public class CppAuthoritiesUserDetailsServiceImpl implements UserDetailsService {

    private final transient Log log = LogFactory.getLog(CppAuthoritiesPopulator.class);
    private UserDao userDao;

    @Autowired
    public CppAuthoritiesUserDetailsServiceImpl(UserDao userDao) {
        this.userDao = userDao;
    }

    public UserDetails loadUserByUsername(String username)
        throws UsernameNotFoundException, DataAccessException {
        User user = (User) userDao.loadUserByUsername(username);
        return user;
    }
}

```

In case of problems... activate the debug

One recommendation, you can configure the log4j.xml file to see what's happening in the spring security environment:

```

<logger name="org.springframework.security">
  <level value="DEBUG" />
</logger>

<logger name="org.springframework.ldap">
  <level value="DEBUG" />
</logger>

```

## AppFuse 2.0.x + Spring Security 2.x

This part is taken from a [thread](#) on the AppFuse user list.

### *Adding dependencies to the pom.xml*

In the pom.xml, add the following dependencies

```

<dependencies>
...
  <dependency>
    <groupId>org.springframework.ldap</groupId>
    <artifactId>spring-ldap-core</artifactId>
    <version>${spring.ldap.version}</version>
  </dependency>
  <dependency>
    <groupId>org.springframework.ldap</groupId>
    <artifactId>spring-ldap-core-tiger</artifactId>
    <version>${spring.ldap.version}</version>
  </dependency>
...
</dependencies>

```

And the corresponding variable at the end:

```
<spring.ldap.version>1.3.0.RELEASE</spring.ldap.version>
```

### *Configuring the security.xml file*

The default userDao authentication provider

You can comment it to have only one type of authentication, or you can leave it to have a chaining auth, first LDAP and after the userDao validation.

```

<authentication-provider user-service-ref="userDao">
  <password-encoder ref="passwordEncoder" />
</authentication-provider>

```

Add the LDAP server

```

<ldap-server id="ldapServer"
  url="ldap://localhost:389/dc=example,dc=com";
manager-dn="cn=Manager,dc=example,dc=com"
manager-password="pass" />

```

If you don't specify the manager-dn and manager-password the connection will be anonymous.

The authorization

Configure the binding procedure (how ldap will do the authentication) and the populate procedure (how ldap will do the authorization, with this configuration you need to have a cn property in the LDAP to map the correct roles inside the application).

#### *Database authorization with a custom populator*

```
<beans:bean id="userSearch"
  class="org.springframework.security.ldap.search.FilterBasedLdapUserSearch">
  <beans:constructor-arg index="0" value="" />
  <beans:constructor-arg index="1" value="(uid={0})" />
  <beans:constructor-arg index="2" ref="ldapServer" />
</beans:bean>

<beans:bean id="ldapAuthenticationProvider"
  class="org.springframework.security.providers.ldap.LdapAuthenticationProvider" autowire="
default">
  <custom-authentication-provider/>
  <beans:constructor-arg>
    <beans:bean class="org.springframework.security.providers.ldap.authenticator.BindAuthenticator">
      <beans:constructor-arg ref="ldapServer" />
      <beans:property name="userDnPatterns">
        <beans:list><beans:value>uid={0}</beans:value></beans:list>
      </beans:property>
      <beans:property name="userSearch" ref="userSearch" />
    </beans:bean>
  </beans:constructor-arg>

  <beans:constructor-arg>
    <beans:bean class="cat.urv.pdd3.webapp.util.Pdd3AuthoritiesPopulator">
      <beans:constructor-arg ref="userDao" />
    </beans:bean>
  </beans:constructor-arg>
</beans:bean>
```

You can create your custom Populator, in case of you want to have the mapping logic about what role have one user.



```

package cat.urv.custom.webapp.util;

import java.util.HashSet;
import java.util.Set;

import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.ldap.core.DirContextOperations;
import org.springframework.security.GrantedAuthority;
import org.springframework.security.GrantedAuthorityImpl;
import org.springframework.security.ldap.LdapAuthoritiesPopulator;

import cat.urv.custom.dao.UserDao;
import cat.urv.custom.model.Role;
import cat.urv.custom.model.User;
import cat.urv.custom.service.UserManager;

public class CustomAuthoritiesPopulator implements LdapAuthoritiesPopulator {

    private final transient Log log = LogFactory.getLog(CustomAuthoritiesPopulator.class);
    private UserDao userDao;

    @Autowired

    public CustomAuthoritiesPopulator(UserDao userDao) {
        this.userDao = userDao;
    }

    public GrantedAuthority[] getGrantedAuthorities(
        DirContextOperations userData, String username) {

        Set<GrantedAuthority> userPerms = new HashSet<GrantedAuthority>();

        User user = (User) userDao.loadUserByUsername(username);
        Set<Role> roles = user.getRoles();

        //get users permissions from service
        for (Role perm : roles) {
            userPerms.add(new GrantedAuthorityImpl(perm.getName()));
        }
        return (GrantedAuthority[]) userPerms.toArray(new GrantedAuthority[userPerms.size()] );
    }
}

```

#### *LDAP authorization from attribute*

You can also be able to authorize one user using an attribute of the LDAP repository:

```

<beans:bean id="ldapAuthenticationProvider"
    class="org.springframework.security.providers.ldap.LdapAuthenticationProvider" autowire="
default">
<custom-authentication-provider/>
<beans:constructor-arg>
    <beans:bean class="org.springframework.security.providers.ldap.authenticator.BindAuthenticator">
        <beans:constructor-arg ref="ldapServer"/>
        <beans:property name="userDnPatterns">
            <beans:list><beans:value>uid={0}</beans:value></beans:list>
        </beans:property>
        <beans:property name="userSearch" ref="userSearch"/>
    </beans:bean>
</beans:constructor-arg>

<beans:constructor-arg>
    <beans:bean class="org.springframework.security.ldap.populator.DefaultLdapAuthoritiesPopulator">
        <beans:constructor-arg ref="ldapServer"/>
        <beans:constructor-arg value="ou=People"/>
        <beans:property name="groupRoleAttribute" value="cn"/>
    </beans:bean>
</beans:constructor-arg>
</beans:bean>

```

In case of problems... activate the debug

One recommendation, you can configure the log4j.xml file to see what's happening in the spring security environment:

```

<logger name="org.springframework.security">
    <level value="DEBUG"/>
</logger>

<logger name="org.springframework.ldap">
    <level value="DEBUG"/>
</logger>

```

## AppFuse 1.9.4 + Acegi Security

This part is taken from a [thread](#) on the AppFuse user list.

Here's what Matt has done in the past to get LDAP working with AppFuse 1.9.4. The same concepts should be applicable to AppFuse 2.0.x.

1. Change the "authenticationManager" bean to use "ldapProvider" instead of "daoAuthenticationProvider":

```

<bean id="authenticationManager" class="org.acegisecurity.providers.ProviderManager">
    <property name="providers">
        <list>
            <ref local="ldapProvider"/>
            <!--ref local="daoAuthenticationProvider"/-->
            <ref local="anonymousAuthenticationProvider"/>
            <ref local="rememberMeAuthenticationProvider"/>
        </list>
    </property>
</bean>

```

2. Added ldapProvider and supporting beans:

```

<bean id="ldapProvider" class="org.acegisecurity.providers.ldap.LdapAuthenticationProvider">
  <constructor-arg>
    <bean class="org.acegisecurity.providers.ldap.authenticator.BindAuthenticator">
      <constructor-arg ref="initialDirContextFactory"/>
      <property name="userDnPatterns">
        <list>
          <value>uid={0}</value>
        </list>
      </property>
      <property name="userSearch" ref="userSearch"/>
      <property name="userDetailsMapper" ref="ldapUserDetailsMapper"/>
    </bean>
  </constructor-arg>
  <constructor-arg>
    <bean class="org.acegisecurity.providers.ldap.populator.DefaultLdapAuthoritiesPopulator">
      <constructor-arg ref="initialDirContextFactory"/>
      <constructor-arg value=""/>
      <property name="groupRoleAttribute" value="cn"/>
      <property name="groupSearchFilter"
value="(&amp;(objectclass=groupOfUniqueNames)(uniqueMember={0}))"/>
      <property name="searchSubtree" value="true"/>
      <property name="rolePrefix" value=""/>
      <property name="convertToUpperCase" value="false"/>
    </bean>
  </constructor-arg>
</bean>

<bean id="initialDirContextFactory" class="org.acegisecurity.ldap.DefaultInitialDirContextFactory">
  <constructor-arg value="{ldap.url}/{ldap.base}"/>
  <property name="managerDn" value="{ldap.username}"/>
  <property name="managerPassword" value="{ldap.password}"/>
</bean>

<bean id="userSearch" class="org.acegisecurity.ldap.search.FilterBasedLdapUserSearch">
  <constructor-arg index="0" value=""/>
  <constructor-arg index="1" value="(uid={0})"/>
  <constructor-arg index="2" ref="initialDirContextFactory"/>
  <property name="searchSubtree" value="true"/>
</bean>

<bean id="ldapUserDetailsMapper" class="org.acegisecurity.userdetails.ldap.LdapUserDetailsMapper">
  <property name="rolePrefix" value=""/>
</bean>

```

3. Change the passwordEncoder bean to be LdapShaPasswordEncoder:

```

<bean id="passwordEncoder" class=
"org.acegisecurity.providers.ldap.authenticator.LdapShaPasswordEncoder"/>

```

In this example, my ldap.properties (which populates initialDirContextFactory) is set to:

```

ldap.url=ldap://localhost:1389
ldap.base=ou=system
ldap.username=uid=admin,ou=system
ldap.password=secret

```

## Protecting Actions and Controllers

### Secure JSF components

## The Challenge

Very often when building website with lots of different pages you end up with some pages that are accessible to users with certain roles. In a simple case you could have a normal user role and an admin role much like the default AppFuse setup. So you would have pages a normal user role could only access and also pages that an admin role could only access. Now even in this simple case, you would most often have pages that are accessible to both the admin role and the normal role.

Although the page is accessible to both roles you might have information on that same page that only is should be visible to a certain role. Another common scenario is that you have a list of users but the button to edit a user should only be visible to the admin role. The most common technique to my knowledge is to wrap the information or button, in some kind of tag like:

```
<...:isUserInRole role="admin"> BUTTON ONLY VISIBLE TO ADMIN ROLE</...:isUserInRole>
```

In AppFuse you would probably use the Acegi tags for doing this stuff. I think this is a very bad practice; you end up with very messy pages which are hard to maintain. Let's say you had a very small site with 50 pages. If you add a role you would have to go through all your pages and edit the isUserInRole tags. Bad if you ask me! This example was only a simple; imagine you had 20 different roles, like economy role, admin role, IT role etc.

## The Proposed Solution

First of all this idea is inspired by an article written by Rick Hightower and his work inspired me to write a Maven-based solution. Be aware that this solution is based on the MyFaces implementation of the JSF specification and therefore of course is written for Appfuse with the JSF front-end.

As we are using a JSF front-end you probably would know that the pages contain JSF components. A JSF component could be the following.

```
<h:commandButton value="#{text['button.delete']}" action="#{userForm.delete}"
    styleClass="button" onclick="bCancel=false; return confirmDelete('User')"/>
```

Now to avoid wrapping the component in some userrole custom tag or a JSTL if statement would it not be nice if the commandButton would know by itself to be rendered or not? I think so and it would be a very clean approach for the UI designer as he/she would not need to think about enabling or disabling this button. (Note: Tomahawk component are security aware and you can write the roles as attributes to the component).

So what we want to achieve is that all myfaces jsf components will know if to render themselves or not. Sounds kind of tricky because we don't want to extend all myfaces components and write custom code but the solution is pretty simple. We will use an aspect!

All JSF components do implement methods called encodeBegin, encodeEnd and encodeChildren. These methods are responsible for rendering the component. Sometimes the method itself contains the rendering code but most often the method delegates the rendering to a dedicated renderer! What we need to do is to intercept these method calls and to call our own code which should then be responsible for deciding if the component should be rendered! Intercepting these method call and inserting our own security code in the already compiled jars, is a job for an aspect and we will use the popular aspectj compiler for this.

## How

The steps involved for accomplishing our objective are:

1. Add the aspectj compiler dependency to your pom file
2. Add the jar files to be weaved
3. Write your aspect
4. Write your security code

### Step1

We need to add the aspectj compiler to our pom file. Some projects can consist of more than one pom.xml file, so you have to decide where you want to put the dependency. Adding it to the outermost pom.xml file should be fine! Add the dependency:

```
<dependency>
  <groupId>aspectj</groupId>
  <artifactId>aspectjrt</artifactId>
  <version>1.5.0</version>
</dependency>
```

### Step2

Now we need to add the jar files to be weaved with our custom code. In our case it is the myfaces.jar files but if you would use other third party JSF components, you would do it the same way. The jar files need to be added to a pom file by adding a aspectj plugin configuration. Now the JSF components are used in the webapp so I suggest you add the needed configuration in the pom.xml file of your webapp. Do the following:

```
<plugin>
  <groupId>org.codehaus.mojo</groupId>
  <artifactId>aspectj-maven-plugin</artifactId>
  <configuration>
    <weaveDependencies>
      <weaveDependency>
        <groupId>org.apache.myfaces.core</groupId>
        <artifactId>myfaces-api</artifactId>
      </weaveDependency>
      <weaveDependency>
        <groupId>org.apache.myfaces.core</groupId>
        <artifactId>myfaces-impl</artifactId>
      </weaveDependency>
    </weaveDependencies>
  </configuration>
  <executions>
    <execution>
      <goals>
        <goal>compile</goal>
      </goals>
    </execution>
  </executions>
</plugin>
```

Now in the lines above we first declare the plugin and then we configure the plugin. What happens is that we configured the plugin to aspectj-compile the two libraries `<groupId>org.apache.myfaces.core</groupId> <artifactId>myfaces-api</artifactId>` and `<groupId>org.apache.myfaces.core</groupId> <artifactId>myfaces-impl</artifactId>` with any aspect found in your source code. Your custom code will be weaved into the jars. Remember for this to work you must have the two libraries defined as dependencies somewhere your poms, I have them just below the plugin configuration so my configuration looks like this:

```

<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-war-plugin</artifactId>
    </plugin>
    <plugin>
      <groupId>org.codehaus.mojo</groupId>
      <artifactId>dbunit-maven-plugin</artifactId>
    </plugin>

    <plugin>
      <groupId>org.appfuse</groupId>
      <artifactId>maven-warpath-plugin</artifactId>
    </plugin>
    <plugin>
      <groupId>org.codehaus.mojo</groupId>
      <artifactId>aspectj-maven-plugin</artifactId>
      <configuration>
        <weaveDependencies>
          <weaveDependency>
            <groupId>org.apache.myfaces.core</groupId>
            <artifactId>myfaces-api</artifactId>
          </weaveDependency>
          <weaveDependency>
            <groupId>org.apache.myfaces.core</groupId>
            <artifactId>myfaces-impl</artifactId>
          </weaveDependency>
        </weaveDependencies>
      </configuration>
    </plugin>
    <executions>
      <execution>
        <goals>
          <goal>compile</goal>
        </goals>
      </execution>
    </executions>
  </plugins>
</build>

<dependencies>
  <dependency>
    <groupId>org.apache.myfaces.core</groupId>
    <artifactId>myfaces-api</artifactId>
    <version>1.1.4</version>
  </dependency>
  <dependency>
    <groupId>org.apache.myfaces.core</groupId>
    <artifactId>myfaces-impl</artifactId>
    <version>1.1.4</version>
  </dependency>
</dependencies>

```

Now lets get to the actual aspect!

### Step3

In src->main folder create a folder called aspect. So now you have src->main-->aspect! In the aspect folder create a file called JSFComponentSecurityAspect.aj. Notice the aj extention. Paste the following information into the file!

```

import org.your.webapp.security.JSFComponentSecurityChecker;
import org.your.webapp.security.impl.SomeSecurityChecker;

import javax.faces.component.UIComponent;

public aspect JSFComponentSecurityAspect {
    private JSFComponentSecurityChecker securityChecker = null;

    public void setSecurityChecker(JSFComponentSecurityChecker checker) {
        securityChecker = checker;
    }

    /*
        A point cut to capture all executions of encode methods (encodeBegin, encodeEnd,
        encodeChildren) of any UIComponent.
        hideComponents is the name of a pointcut. A pointcut is the part of the code that we are going
        to advise.
        execution means join on method execution
        void UIComponent+.encode*(..) means any encode method returning void for any class that
        implements UIComponent
        this(component) is a capturing pointcut so we can access the current component
    */
    public pointcut secureComponents(UIComponent component) :
        execution (void UIComponent+.encode*(..) && this(component);

    void around(UIComponent component) : secureComponents(component) {
        //Should be injected by spring - This is a no go :)
        securityChecker = new SomeSecurityChecker();

        /* If this component is not secured, then allow the encode method of the component to execute.
    */
        if (!securityChecker.isSecured(component)){
            proceed(component);
            return;
        }

        /* Check to see if we are allowed to display the component.
        * If we are allowed, then proceed as usual.
        * If not, then don't proceed and setRendered to false.
        */
        if (securityChecker.isAllowed(component)){
            proceed(component);
        } else {
            component.setRendered(false);
        }
    }
}

```

Here we have the aspect which first has a pointcut definition. It will intercept all calls to the encode\* methods and then call the around advice. An around advice means that the code in the advice gets inserted in the actual method call and therefore we have access to the actual UIComponent being called (see the this(component) which will add the current component to the around advice also called context passing). In the around advice we first instantiate the security interface which of course is bad practice, you should let spring do that. Then we ask the security interface implementation if the current component is secured, if not then we call a special aspect method called proceed which means the component will continue to execute normally. If the component is secured we check if the current component is allowed for the current user (in my case it's a user but it could be anything depending on your implementation), if the user is allowed to access this component, proceed is called and the component method continues, otherwise we set the rendered attribute of the component to false which means it will not be rendered.

This is pretty much it, you of course need to define your interface and your implementation of the security interface but this is up to you, every company has its own requirement. In my case I have a table in the database where all the component id's are registered with their id's and then the matching roles. Very much the same way the userrole stuff works! Then I made some administration GUI to manage it.

Here is my interface and some bogus implementation!

```

public interface JSFComponentSecurityChecker {

    /**
     * Checks to see if a component can be executed based on the implementations rules
     *
     * @param component component
     * @return allowed.
     */
    boolean isAllowed(UIComponent component);

    /**
     * Check to see if the component is secured.
     *
     * @param component component
     * @return secured.
     */
    boolean isSecured(UIComponent component);
}

import org.your.webapp.security.JSFComponentSecurityChecker;
import org.acegisecurity.Authentication;
import org.acegisecurity.context.SecurityContextHolder;
import org.acegisecurity.userdetails.UserDetails;
import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;

import javax.faces.component.UIComponent;
import javax.faces.context.FacesContext;

public class SomeSecurityChecker implements JSFComponentSecurityChecker {
    protected final Log log = LogFactory.getLog(getClass());

    public boolean isAllowed(UIComponent component) {
        if (log.isDebugEnabled()) {
            log.debug("Checking Component with id: "+component.getId() );
        }
        //here you should call business code to figure out if the component should be visible/rendered
        for the current user
        //possibly by checking some component/role/user type of table
        //In this dummy implementation it doesn't check properly, it simply returns true to indicate
        //that the component should be rendered

        return true;
    }

    public boolean isSecured(UIComponent component) {
        if (log.isDebugEnabled()) {
            log.debug("Checking if Component with id: "+component.getId() + "IS secure " + true);
        }
        //TODO REMOVE
    }

    //All components are secure but you should have business code that figures out if component is
    secure
    return true;
}
}

```

## Bonus

I made a second aspect that can document the component, that mean because I have a very complex security setup I thought it would be nice that it would be possible to output comments to actual response so that when you do a view source on the resulting HTML page you can see comments if a component wasn't rendered because the current user doesn't have the correct permissions. This is of course only enabled if we are in development so that it will not be visible when running in production! This is only a prototype but I put it here for inspiration. The most difficult part was passing all the different parameters around and it took me some time to figure the aspect syntax out



```

import javax.faces.context.FacesContext;
import javax.faces.component.UIComponent;
import javax.faces.render.Renderer;
import javax.faces.context.ResponseWriter;
import java.io.IOException;

public aspect JSFComponentDocumentator {

    /**
     * Intercept calls to any render component and provide the arguments to the around advice
     */
    public pointcut documentComponent(Renderer renderer, FacesContext context, UIComponent component) :
        execution (void Renderer+.encode*(FacesContext, UIComponent)) && target(renderer)&& args(context,
component);

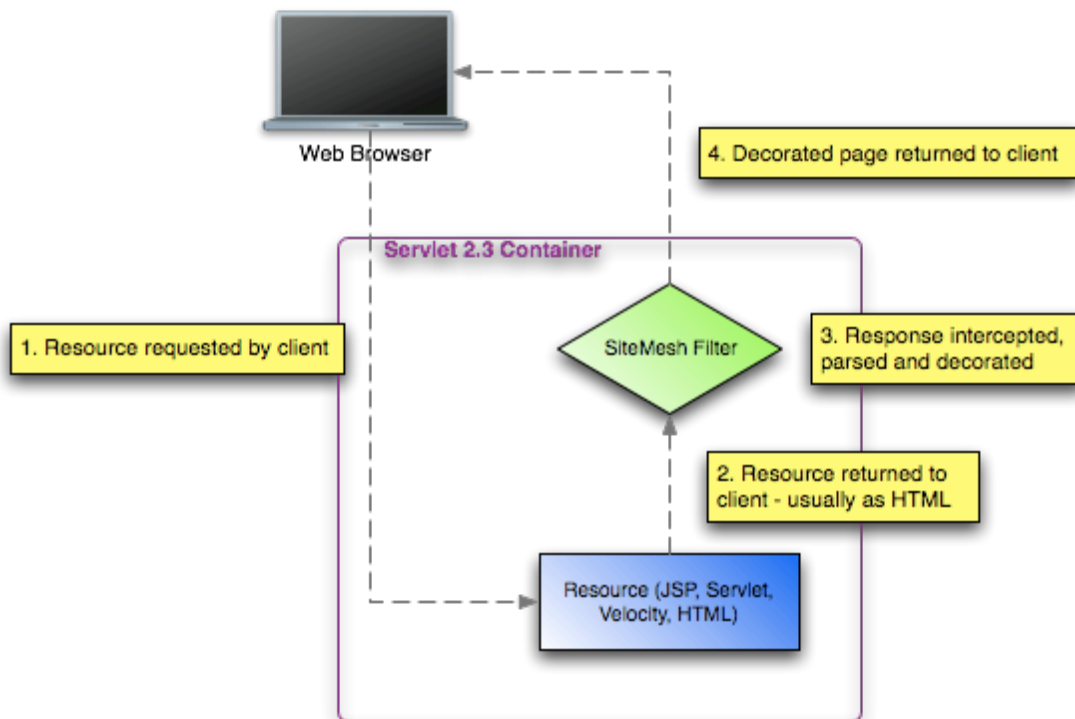
    void around(Renderer renderer, FacesContext context, UIComponent component) :
documentComponent(renderer, context, component) {
        ResponseWriter writer = context.getResponseWriter();
        try {
            //Here you could write information about a component which is not rendered as source
            comments.
            //This could be helpful in development where there could be comments in the html output generated
            //by the renderer. So in a HTML page you could have a comment like: <!-- Component with ID someID is
            not
            //rendered because user had role (somerole) and the required role for this component are (req
            roles)-->
            //Just an example what you could do
            writer.writeComment("Currently Rendering component with ID: " +component.getId());
        } catch (IOException e) {
            e.printStackTrace();
        }

        //Proceed with the render method
        proceed(renderer,context,component);
    }
}

```

## SiteMesh

[SiteMesh](#) is an open-source layout and page decoration framework from the OpenSymphony project. It was originally created over 5 years ago, when Joe Walnes downloaded the first Sun servlet engine and wrote it using servlet chains. Over the years, the basic design has stayed the same; content is intercepted and parsed, and a decorator mapper finds a decorator and merges everything. The figure below shows a simplistic example of how this works.



Skinning is an essential element to every web application. The ability to edit one or two files to change the entire layout of an application is necessary for maintainability. SiteMesh is a simple decoration framework and is very easy to install and configure.

To learn more about SiteMesh, see the following articles:

- [Introduction to SiteMesh](#)
- [Advanced SiteMesh](#)

For the first couple years, AppFuse used Apache Tiles for its templating framework. Since Tiles didn't work with all the frameworks AppFuse supports, [we switched to SiteMesh](#). The transition was [fairly easy](#).

The menu selection logic in AppFuse is done using [SiteMesh](#) and a form of [dependency injection](#).

## Spring

AppFuse uses [Spring](#) for its [dependency injection](#) capabilities, as well as because it simplifies Java EE development and testing. To learn more about Spring and how it works, please read [Introduction to the Spring Framework](#) by Rod Johnson. If you have Spring experience, but haven't worked with Spring 2.0, you'll want to read [Spring 2.0: What's New and Why it Matters](#) by Rod Johnson. For a condensed version, see [What's New in Spring 2.0](#) from Spring's documentation.



There are a number of [books](#) written on Spring. The best ones are [Spring Live](#), [Spring in Action](#) and [Pro Spring](#). Spring in Action is [currently being updated](#) for Spring 2.0. If you're interested in the history of Spring more so than how to develop with it, you'll want to read [J2EE Design and Development](#) by Rod Johnson.

Useful Spring documentation:

- [Spring's Reference Manual](#)
- [Spring Framework API](#)
- [Spring Modules Reference Manual](#)

Spring Live was written by [Matt Raible](#), AppFuse's founder. This book was written for Java developers with some web development experience. If that fits your profile, there's a good chance you'll enjoy this book. [Chapter 2](#) is available as a free download.

## Version Control

## Hosting



## Git Hosting

### Commercial Projects

- [GitHub](#)

### Open-Source Projects

- [GitHub](#)
- [repo.or.cz](#)
- [Gitorious](#)

## Subversion Hosting

### Commercial Projects

- [Beanstalk](#)
- [CVSDude](#)
- [CollabNet](#)

### Open-Source Projects

- [Google Code](#)
- [SourceForge.net](#)
- [CodeHaus](#)
- [Java.Net](#)
- [Kenai](#)

## Mercurial Hosting

### Commercial Projects

### Open-Source Projects

- [Kenai](#)

## Bazaar Hosting

### Commercial Projects

### Open-Source Projects

- [launchpad](#)

## Web Filters

This page is a summary of the web filters configured in AppFuse including the description and purpose of each. The content was taken from [this thread](#) from the AppFuse user mail list and formatted for the wiki site.

Filter	Description	Use in AppFuse
cacheFilter	Can be used to cache requests by mapping it to particular URLs. OSCache also has JSP tags that you can use to cache specific portions of your page.	This is not used--its filter-mapping is commented out.
clickstreamFilter	Captures where the user clicked and can be used for statistics. We've thought about replacing it with MessAdmin - since <a href="#">MessAdmin seems</a> to be a more active project and provides the same functionality and more.	Admin users can see the path taken by users

encodingFilter	Allows foreign characters and i18n support work much better. See this <a href="#">filter's javadocs</a> for more information.	For i18n support
gzipFilter	This compresses the response to the user by up to 70%. This is particularly useful for compressing JavaScript and CSS. Onjava has a good article to read to understand the usefulness of <a href="#">GzipFilter</a> .	To compress responses. Sometimes it is helpful to turn this filter off when testing unexpected responses.
lazyLoadingFilter	Used for Hibernate lazy-loading. If you initialize all your lazy collections in your DAO layer or middle-tier, you won't need this filter.	Disabled in SVN and 2.0-M5 release since we don't need it in the core and it prevents us from easily switching DAO Frameworks.
localeFilter	Allows i18n switching by passing in a "locale" parameter. It also sets the user's locale on a ThreadLocal so you can read it in any layer of the application.	Used in AppFuse for generating i18n messages.
rewriteFilter	Similar to Apache's mod_rewrite and allows you to <a href="#">beautify URLs</a> and all kinds of other things.	It's not used by most frameworks (except Struts 2), but it's available for use.
securityFilter	Delegates to Acegi Security and prevents users from accessing certain URLs.	security.xml defines which roles are allowed to access which URL's
sitemesh	Allows modular composition and decoration of pages with your app	AppFuse uses this to provide support for themes and common decoration elements (headers, footers, menus, etc)
struts-cleanup	According to the <a href="#">Struts 2 Javadocs</a> this is required if you're using SiteMesh with Struts 2	Which we are 😊
struts	The front servlet that dispatches all action requests for Struts 2	Mapped to /* so it can do things like add CSS and JavaScript to the <head> (if you have an <s:head> tag there. The extension that's used is configured in struts.xml.

## Tutorials

To develop an application with AppFuse, you generally create [POJOs](#) and configure how they work with [Spring](#). Follow the steps below to create a master/detail screens that CRUD a Person object. You should create a project using the [QuickStart Guide](#) and setup your [Development Environment](#) before starting these tutorials.

1. Create a [Person Entity](#).
2. Create a [PersonDao](#) in [Hibernate](#), [iBATIS](#) or [JPA](#).
3. Create a [PersonManager](#) to act as a service facade to [PersonDao](#).
4. Create the web tier using [JSF](#), [Struts 2](#), [Spring MVC](#) or [Tapestry](#).



### Help us help you

These tutorials are designed to make it easy to develop applications with open source Java frameworks. If you have difficulty in any sections, or think that things can be clarified, we encourage you to 1) contact the [mailing list](#) or 2) modify the page to make things more clear.

To modify content, [create an account](#), then navigate to the page you want to modify and select **Edit > Edit this page**. The AppFuse Team monitors changes so they will receive notifications of your changes and can correct any mistakes you make.

### Where's the source?

One of the most commonly asked questions about AppFuse 2.x is [where's the source?](#) If you'd like to convert your project so it doesn't rely on AppFuse dependencies and includes AppFuse's source instead, run **mvn appfuse:full-source**.

### CRUD Generation

AppFuse has a [Maven Plugin](#) that allows you to generate all the code in these tutorials. However, we recommend you complete these tutorials before using it so you're familiar with all the code that's being generated.

One of the nice features of AppFuse 2.x is you no longer need to create a DAO or Manager for type-safe CRUD functionality. You merely need to create Spring bean definitions. This means you should be able to easily develop a front-end without worrying about writing code for the backend (until you need customized functionality of course).



### Having trouble choosing a web framework?

If you're having trouble choosing a web framework, read [What Web Application framework should you use?](#) and [Java Web Framework Sweet Spots](#). More information on comparing web frameworks could be found in [Matt Raible's Presentations](#).

# Development Environment

## About this Tutorial

This tutorial describes how to setup your development environment to compile/test/deploy AppFuse using Maven and your favorite IDE ([Eclipse](#), [IDEA](#) or [NetBeans](#)). Eclipse works well for single-module projects, whereas IDEA and NetBeans work better with multi-module projects. Knowledge of Maven is not required to use AppFuse because the tutorials explain how to use it. If you're interested in learning [Maven](#) in-depth, please see [Maven: The Definitive Guide](#) (it's free!).

## Table of Contents

1. [Download](#) - links to download Java, Maven and MySQL
2. [Install](#) - detailed instructions on where to install everything
3. [Configure](#) - how to configure your environment variables
4. [Additional Tips](#) - tools to boost your productivity

## Download

1. [Download](#) Java 5 SE. AppFuse should work fine with Java 6, but Maven [has issues](#) with multi-module projects.
2. [Download](#) Maven 2.0.5+.
3. [Download](#) MySQL 5.0.27+.

Downloading these files to your desktop should work just fine.

## Install

Make sure you have [7-Zip](#) or [WinZip](#) installed (for Windows) and gnutar for OS X before installing these packages. Linux users should be fine with the default tar tools.

1. Create a "Tools" and "SDKs" folder on your hard drive. On Windows, I create these at c:\Tools and c:\SDKs. On \*nix, I usually do /opt/dev/tools and install Java in the default locations. Make sure and install Sun's Java if you're on Linux. Now that it's GPL, it shouldn't be difficult to *apt-get*.
2. Create Environment variables for these folders - SDKS\_HOME and TOOLS\_HOME (optional)
3. Install Java SE (a.k.a. JDK) in the SDKs directory - keeping the directory names intact.
4. Unzip/Install Maven in the Tools directory - "maven-x" is what I use for the directory name, where x is the version number.
5. Install MySQL in the Tools directory.
6. Create a "Source" directory on your hard drive (this is where you'll put all your projects and their source code). On \*nix, I usually create a "dev" folder in my home directory.

At this point, you should have a directory structure that looks something like the following:

```
SDKs
- jdk1.5.0_10
Tools
- maven-2.0.5
- mysql
Source
```

After installing these tools, you'll need to [setup an SMTP server](#) as well. If there's an existing server you'd like to use, simply change the host name in *src/main/resources/mail.properties*.

Now you'll need to configure all these tools so that your operating system knows they're installed.

## Configure

A Windows example is the only one shown here because it's assume the \*nix folks are smart enough to figure it out for their system.

1. To set Environment Variables in Windows, either go to Control Panel -> System or right-click My Computer -> Properties.
2. Click on the *Advanced* Tab and then click the *Environment Variables* button.
3. Put focus on the second box (System Variables) by clicking on one of the existing values.
4. Enter the following variables:
  - HOME = c:\Source
  - SDKS\_HOME = c:\SDKs
  - TOOLS\_HOME = c:\Tools
  - JAVA\_HOME = %SDKS\_HOME%\jdk1.5.0\_10
  - MAVEN\_HOME = %TOOLS\_HOME%\maven-2.0.5

- `MYSQL_HOME = %TOOLS_HOME%\mysql`
- Append to the PATH variable: `%JAVA_HOME%\bin;%MAVEN_HOME%\bin;%MYSQL_HOME%\bin`

You should now be able to open a command prompt and type "java -version", "mvn -version" or "mysql" and not get errors.

## Additional Tips

- Use [Cygwin](#) on Windows for running Maven and doing all command line things. Install it in `$TOOLS_HOME/cygwin`.
- Use [Eclipse](#) or [IDEA](#). Install them in `$TOOLS_HOME/eclipse-x` or `$TOOLS_HOME/idea-x`. Multi-module Maven projects seem to work best in IDEA. NetBeans also has [excellent Maven support](#), but it's not widely used among AppFuse developers and users. In Eclipse, `Ctrl+Shift+R` is your best friend; `Ctrl+Shift+N` in IDEA. See [IDEs](#) for more information on setting up an AppFuse-based project in your favorite IDE ([Eclipse](#), [IDEA](#), or [NetBeans](#)).

If you're starting work at a new client or at your company, you should do the following to help your development process become more efficient.

1. Setup a source control system. [Subversion](#) is highly recommended. Setting up a commit notification system is recommended when working with other developers. [FishEye](#) works great.
2. Setup a bug tracking system. Popular (free) choices are [Bugzilla](#) and [Trac](#). The best one we've seen is [JIRA](#). For more information see [What issue tracking system is best for you?](#)
3. Setup a Wiki. We love [Confluence](#) - but it's not free. Other recommended systems are [JSPWiki](#) and [Trac](#). Trac has a wiki, source control browser and bug-tracking system all-in-one.
4. Setup a development box to host the source control system, the bug tracking system, and a wiki. Install [Hudson](#), [CruiseControl](#), [Lunbuild](#) or [Continuum](#) on this box to do continuous integration for your project. Setting up continuous integration with a Maven 2 project is a piece-of-cake, so you have no excuses! 😊 If you're interested in commercial CI systems, checkout [Pulse](#) or [Bamboo](#). See [Which open source CI tool is best suited for your application's environment?](#) for more information on choosing a CI Server.
5. (optional) Install [Roller](#) and use it to report your daily status and issues. This will allow your client (or supervisor) to track your progress.



### **Buildix Rocks!**

[Buildix](#) is a VMWare image (that can be installed on bare metal) that has Subversion, CruiseControl and Trac all pre-installed. Using it is [highly recommended](#).

If installing and configuring all of this software doesn't boost your productivity, please read [Tips for Productivity and Happiness at Work](#) for more suggestions.

## Installing an SMTP Server

One of the setup requirements for AppFuse is an SMTP server. By far the easiest and most reliable way to configure AppFuse is to have an SMTP service running on `localhost`. This isn't much of a problem for some operating systems where an SMTP server is installed by default (like Linux and OS X distributions), but for new AppFuse users running Windows it has been a recurring issue. So here are a few options you have for installing an SMTP server for your operating system:

### Platform Independent

- [Apache James](#) - Complete mail and news server written in Java

It is a breeze to send mail with James. Simply download and unpack the distribution, cd into the 'james-2.3.0/bin' directory (versions can obviously differ), and execute the `run.sh` (or `run.bat` for Windows) script. (If you run this script on a Linux machine, you must start it as `sudo` so it can bind to the necessary ports.) If you need to configure James to receive mail, please consult their documentation.

### Windows

- [IIS](#) - (NT/2000/XP) Microsoft SMTP server provided with the OS (Another [setup guide](#))

It is important that if you set up the SMTP server that comes with IIS, to turn on relaying capabilities in order for email to be routed correctly within AppFuse. To do this, follow these steps:

1. Open the SMTP properties window
  - a. Access by clicking Start
  - b. Control Panel
  - c. Administrative Tools
  - d. Internet Information Services
  - e. Expand the domain you use for appfuse
  - f. and right click on the Default SMTP Virtual Server
2. Click the Access tab
3. Click the Relay button to bring up the Relay Restrictions window
4. Select the radio option for "Only the list below"
5. Click the Add button to add access to a particular domain or group
6. If using the localhost as your SMTP server as previously suggested, make sure you add 127.0.0.1 here so that the localhost can properly relay mail.

7. Finally, click OK, Apply, and OK to save your changes.  
Also note that if you're behind a firewall or router, you may need to open ports 25 and 2525.
- [FreeSMTP](#) - (9x/NT/2000/XP) Freeware SMTP server that runs from your system tray

## OS X

- [Postfix](#) - Included in the OS (regular & server, if not started: try "sudo postfix start" in your Terminal window)

## Linux

- [Sendmail](#) - Most common SMTP server for Linux
- [Postfix](#) - Easier to configure alternative to the ubiquitous Sendmail



Obviously there are more choices than these. This is meant to be a short list to get people up and running on AppFuse. If you know of another server that should be on this list please add it.

## Using JRebel with IntelliJ IDEA

This page describes how to install [JRebel](#) and configure [IntelliJ IDEA](#) to use it. JRebel is a tool that allows you to compile and reload Java classes without restarting your application. In essence, it makes Java web application development very similar to developing with Rails and Grails (from a save/reload perspective). It gives you zero turnaround, which is something [all web frameworks should support](#).

1. [Download](#) and install IntelliJ IDEA 9 Ultimate Edition (in /Applications on OS X).
2. [Download](#) and install JRebel.
  - a. `java -jar jrebel-setup.jar`
  - b. Install the [JRebel Plugin for IDEA](#). Shortcut: File > Settings > Search for plugins and find JRebel.
3. On OS X, Modify `/etc/launchd.conf` and add the following so M2\_HOME is available to GUI apps. You'll need to reboot after making this change.

```
setenv M2_HOME /opt/tools/maven2
```

More info on this setting is [available on Stack Overflow](#).

4. Modify your project's pom.xml to include the [JRebel Maven plugin](#) (for generating the configuration of which directories and files to watch for changes).

```
<plugin>
  <groupId>org.zeroturnaround</groupId>
  <artifactId>javarebel-maven-plugin</artifactId>
  <version>1.0.5</version>
  <executions>
    <execution>
      <id>generate-rebel-xml</id>
      <phase>process-resources</phase>
      <goals>
        <goal>generate</goal>
      </goals>
    </execution>
  </executions>
</plugin>
```

5. Change your pom.xml so Jetty doesn't reload the app when classes are compiled. Specifically, change `<scanIntervalSeconds>3</scanIntervalSeconds>` to `<scanIntervalSeconds>0</scanIntervalSeconds>`.
6. Use the JRebel icons to start jetty:run in your IDE.



7. (Optional) Set a JREBEL\_HOME environment variable that points to your JRebel install (/Applications/ZeroTurnaround/JRebel on OS X) and set your MAVEN\_OPTS to use JRebel's settings (for using JRebel running it from the command line). For example:

```
export JAVA_OPTS="-Xmx512M -XX:PermSize=256m -XX:MaxPermSize=512m -Djava.awt.headless=true"
export JREBEL_HOME=/Applications/ZeroTurnaround/JRebel
export MAVEN_OPTS="$JAVA_OPTS -noverify -javaagent:$JREBEL_HOME/jrebel.jar"
```

After making these changes, you should be able to compile classes in IDEA and refresh your browser. Log messages like the following should show up in your console.

```
JRebel: Reloading class 'org.appfuse.webapp.action.UserAction'.
```

To simplify things further, you can map Command+S to compile (instead of Shift+F9). Just look for Keymaps in Settings, rename the default one and search for Compile to remap.

## Enterprise Integration

AppFuse should work with most EIS (Enterprise Information Systems). It's usually just a matter of configuration to get an AppFuse-based application to communicate with an EIS.

If you've worked with AppFuse and integrated it into an EIS, feel free to fill in a tutorial (or contribute a tutorial of your own). 😊

## Clustering with Terracotta

### Purpose

This tutorial describes how to integrate [Terracotta](#) (TC) into AppFuse using the [TC Maven Plugin](#). We hence assume that you have a general understanding of what these applications can do. At the end of this tutorial, you should have a TC enabled application capable of scaling up to several thousands of simultaneous users within a few clicks.

Since Terracotta is a library used to cluster data seamlessly amongst many nodes, we'll be sharing parts of the info that are running in every instance of AppFuse amongst n-number of Jetty Servers.

### Setting up the environment

For the following, we'll assume you've already setup your AppFuse environment, if not, please refer to [AppFuse's QuickStart](#) with Spring MVC web framework. Please notice that Terracotta can also be integrated in any other web framework in AppFuse.

Start by creating a new AppFuse Spring MVC project:

```
GroupId: com.mycompany
ArtifactId: myproject
Version: 2.1.0-M1
Web Framework: Spring MVC
```

Create the new project by executing the following generated Maven command:



```
mvn archetype:generate -B -DarchetypeGroupId=org.appfuse.archetypes \
-DarchetypeArtifactId=appfuse-basic-spring-archetype \
-DarchetypeVersion=2.2.0 -DgroupId=com.mycompany -DartifactId=myproject
```

Run the following command inside the project's directory:

```
mvn jetty:run-war
```

This will run the AppFuse application on <http://localhost:8080>.

## Integrating Terracotta

Terracotta runs as a standalone server on a machine. It accepts 'client' connections from remote applications, namely our AppFuse based applications.

In order to tell AppFuse where to find the central Terracotta server, we use the TC configuration file, named "tc-config.xml" by convention. Inside this file, we usually specify the hostname of the TC server, the data to be shared, where to log the server and client logs, which classes to instrument and which modules to include.

Place the following configurations inside a file named "tc-config.xml" file and save it in the same directory that contains the file pom.xml of the AppFuse project.

```
<tc:tc-config xmlns:tc="http://www.terracotta.org/config">

<!--Tell DSO where the Terracotta server can be found;

See the Terracotta DSO Guide for additional information.-->

<servers>
  <server host="%i" name="sample">
    <data>target/terracotta/server/data</data>
    <logs>target/terracotta/server/logs</logs>
    <statistics>target/terracotta/server/statistics</statistics>
  </server>
</servers>
<!--Tell DSO where to put the generated client logs
See the Terracotta DSO Guide for additional information.-->
<clients>
  <logs>target/terracotta/clients/logs/%(tc.nodeName)</logs>
  <statistics>target/terracotta/clients/statistics/%(tc.nodeName)</statistics>
  <modules>
    <!-- this module will automatically tell the Jetty container to share and cluster the session
map -->
    <module name="tim-jetty-6.1" version="2.1.1" />
  </modules>
</clients>
<application>
<dso>
<!--Our app requires these custom objects/classes to be shared - the following declarations
tells DSO which ones they are. When the app runs under DSO, instances of these classes
will broadcast changes in their state.
A good idiom when writing an app that you intend to cluster via TC DSO, is to group the
classes you wish to share under a single package (although if you follow the MVC pattern
this tends to happen naturally) - this way the list of classes you wish to instrument
can be concise-->
<instrumented-classes>
<!--Include all application classes that are clustered-->
<include>
  <class-expression>com.opensymphony.clickstream.Clickstream</class-expression>
</include>
<include>
  <class-expression>com.opensymphony.clickstream.ClickstreamRequest</class-expression>
</include>
```

```

<include>
  <class-expression>org.springframework.security.ui.savedrequest.SavedRequest</class-expression>
</include>
<include>
  <class-expression>org.springframework.security.ui.savedrequest.SavedCookie</class-expression>
</include>
<include>
  <class-expression>org.springframework.security.context.SecurityContextImpl</class-expression>
</include>
<include>
  <class-expression>org.springframework.security.providers.UsernamePasswordAuthenticationToken</class-expression>
<class-expression>org.springframework.security.providers.AbstractAuthenticationToken</class-expression>
<class-expression>org.appfuse.model.User</class-expression>
</include>
<include>
  <class-expression>org.appfuse.model.BaseObject</class-expression>
</include>
<include>
  <class-expression>org.springframework.security.ui.WebAuthenticationDetails</class-expression>
</include>
<include>
  <class-expression>org.appfuse.model.Role</class-expression>
</include>
<include>
  <class-expression>org.appfuse.model.Address</class-expression>
</include>
<include>
  <class-expression>org.hibernate.collection.PersistentSet</class-expression>
</include>
<include>
  <class-expression>org.hibernate.collection.AbstractPersistentCollection</class-expression>
</include>
<include>
  <class-expression>org.springframework.security.BadCredentialsException</class-expression>
</include>
<include>
  <class-expression>org.springframework.security.AuthenticationException</class-expression>
</include>
<include>
  <class-expression>org.springframework.security.SpringSecurityException</class-expression>
</include>
<include>
  <class-expression>org.springframework.core.NestedRuntimeException</class-expression>
</include>
</instrumented-classes>
<additional-boot-jar-classes>
  <include>java.lang.String$CaseInsensitiveComparator</include>
  <include>java.util.Locale</include>
</additional-boot-jar-classes>
<!--Tell DSO which applications in your web container is using DSO-->
<web-applications>
  <web-application>myproject-1.0-SNAPSHOT</web-application>
</web-applications>
</dso>

```

```
</application>
</tc:tc-config>
```

We now need to TC-enable the AppFuse generated project. In order to keep things simple, we'll use the TC Maven plugin which downloads a mock Terracotta server and runs it on the localhost. It will download Terracotta's core libraries and use them to run the servers and nodes. This plugin will NOT use the Terracotta distribution you installed manually. The above distribution ships with an extra Swing GUI called the Dev Console that will allow you to debug the TC memory in real time.

Add the following to the pom.xml file in their corresponding places:

```
<repository>
  <id>terracotta-repository</id>
  <url>http://www.Terracotta.org/download/reflector/maven2</url>
</repository>
<pluginRepository>
  <id>terracotta-repository</id>
  <url>http://www.terracotta.org/download/reflector/maven2</url>
</pluginRepository>
```

```
<!-- this will add the Jetty module library. This library will be used by the container -->
<dependency>
  <artifactId>tim-jetty-6.1</artifactId>
  <version>2.1.1</version>
  <groupId>org.terracotta.modules</groupId>
  <scope>provided</scope>
</dependency>
```

```

<!--
This will create a new profile that will be used in order to run the Terracotta server and two
Terracotta nodes (application server instances).
This will download the Terracotta core libraries (version 3.2.1) that will be used.
This will download the Jetty container also.
-->
<profile>
  <id>jetty6x</id>
  <build>
    <plugins>
      <plugin>
        <artifactId>tc-maven-plugin</artifactId>
        <version>1.5.1</version>
        <groupId>org.terracotta.maven.plugins</groupId>
        <configuration>
          <processes>
            <process nodeName="cargo" count="2">
              <container>
                <containerId>jetty6x</containerId>
                <zipUrlInstaller>
                  <url>http://dist.codehaus.org/jetty/jetty-6.1.22/jetty-6.1.22.zip</url>
                </zipUrlInstaller>
              </container>
            </process>
          </processes>
        </configuration>
      </plugin>
    </plugins>
  </build>
</profile>

```

## Running AppFuse with Terracotta

Now go to your project's directory. Make sure you generated the war file of the project in the target folder. If you don't have the war file, execute the following command to generate it:

```
mvn jetty:run-war
```

Then execute the following command to run the project with Terracotta:

```
mvn -Pjetty6x tc:run
```

If you run the command for the first time then you'll have to wait after getting this because Maven will be downloading the "installs" folder which contains Jetty (24.3 MB).

```
[INFO] Resolving modules: [<xml-fragment name="tim-jetty-6.1" version="2.1.1" xmlns:tc="http://www.terracotta.org/config"/>]
[INFO] Resolving bundle: org.terracotta.modules:excludes-config:3.2.1
[INFO] Resolving bundle: org.terracotta.modules:modules-base:1.2.1
[INFO] Resolving bundle: org.terracotta.modules:guimodels-config:3.2.1
[INFO] Resolving bundle: org.terracotta.modules:jdk15-preinst-config:3.2.1
[INFO] Resolving bundle: org.terracotta.modules:standard-config:3.2.1
[INFO] Resolving location: org.terracotta.modules:tim-jetty-6.1:2.1.1
[INFO] Resolving bundle: org.terracotta.modules:tim-session-common:2.1.1
[INFO] Resolving bundle: org.terracotta.modules:tim-distributed-cache:1.3.1
[INFO] Resolving bundle: org.terracotta.modules:tim-concurrent-collections:1.3.1
[INFO] Starting DSO nodes
[INFO] [dso start] 2010-03-28 13:57:37,180 INFO - Terracotta Server instance has started up as ACTIVE
node on 0:0:0:0:0:0:0:9510 successfully, and is now ready for work.
```

This command will choose the Jetty profile and execute the tc:run using the tc-maven-plugin. This will start one Terracotta server. This will also run two client nodes which will connect automatically to the server. The two client nodes will automatically use the tc-config.xml we created as their configuration files.

Make sure you remove the your old cookie session from your browser.

### Congratulations !!!

You're all set. Your myproject application will be run by two client nodes (listening on different ports):

<http://localhost:8080/myproject-1.0-SNAPSHOT>

<http://localhost:8081/myproject-1.0-SNAPSHOT>

Follow those steps to make sure that Terracotta is clustering your session:

1. go to <http://localhost:8080/myproject-1.0-SNAPSHOT>
2. login using admin/admin
3. go to <http://localhost:8081/myproject-1.0-SNAPSHOT/admin/users.html> (using the other port)
4. if you are not redirected to the login screen then it worked

You can monitor your Terracotta server and the two clients by running the dev console:

Linux:

```
$TC_INSTALL_DIR/bin/dev-console.sh
```

Windows:

```
$TC_INSTALL_DIR/bin/dev-console.bat
```

Configure your loadbalancer to dispatch the load on the 2 nodes. At anytime, if any of the nodes fails, your inmemory data is safely clustered on the TC Server (which can be installed in a highly available mode).

### From here on

You should be able to setup your AppFuse instances to connect to an actual TC Server in no time. To do this, refer to the 'host' parameter inside the config.xml file and change it to point to the newly installed TC Server.

Helpful sources:

- <http://forums.terracotta.org/forums/posts/list/1792.page>
- <http://forge.terracotta.org/releases/projects/tc-Maven-plugin/cargo.html>

Ayman Abou Hamra  
abouhamra@setic.eu  
Setic <http://www.setic.eu>

In this article, we've used

- Maven 2.2.1

- Java: 1.6.0\_10
- Terracotta: 3.2.1
- AppFuse 2.1.0-M1 with the Spring MVC framework
- Jetty 6.1.22

## JBPM Integration

### HOWTO: JBPM process engine + appfuse jsf modular

#### required resources:

- appfuse 2.0.2 jsf modular
- jbpms 3.2.2
- spring-modules jbpms

#### integration:

##### database

use JBPM ddl according to your database. I recommend to put jbpms-db-objects to another schema than your domain-db-objects. Easier for updates.

##### hibernate

```
<hibernate-configuration>
  <session-factory name="java:/jbpmsSessionFactory">

    <mapping class="org.appfuse.model.User"/>
    <mapping class="org.appfuse.model.Role"/>
    ...
  ALL JBPM *.hbm.xml
```

#### give your sessionFactory a name - so you can reference it in JBPM-configuration

Copy the \*.hbm.xml files to your classpath. I recommend a new maven module for it.

#### jbpms.cfg.xml

to your classpath: - jbpms-config

```

<jbpm-configuration>

  <jbpm-context>
    <service name="persistence">
      <factory>
        <bean class="org.jbpm.persistence.db.DbPersistenceServiceFactory">
          <!-- let us handle transactions by aop or @Transactional; so we have to turn jbpm tx off
-->
          <field name="isTransactionEnabled"><false/></field>
          <field name="sessionFactoryJndiName">
            <!-- same as name of your hibernate session factory -->
            <string value="java:/jbpmSessionFactory" />
          </field>
        </bean>
      </factory>
    </service>
    <service name="tx" factory="org.jbpm.tx.TxServiceFactory" />
    <service name="message" factory="org.jbpm.msg.db.DbMessageServiceFactory" />
    <service name="scheduler" factory="org.jbpm.scheduler.db.DbSchedulerServiceFactory" />
    <service name="logging" factory="org.jbpm.logging.db.DbLoggingServiceFactory" />
    <service name="authentication"
factory="org.jbpm.security.authentication.DefaultAuthenticationServiceFactory" />

  </jbpm-context>
  <!-- lets resolve JSF variables in process diagrams -->
  <bean name='jbpm.variable.resolver' class='at.igsoft.jbpm.el.JsfJbpmVariableResolver'
singleton='true' />
</jbpm-configuration>

```

## JSF-EL + JBPM

make sure EL expressions from JSF can be used in JBPM process diagrams:

```

package at.igsoft.jbpm.el;

import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;
import org.appfuse.webapp.util.FacesUtils;
import org.jbpm.jpdl.el.ELException;
import org.jbpm.jpdl.el.impl.JbpmVariableResolver;

/** jbpm can resolve JSF variables with this
use: have in view something like #{myForm.name}
than you can use this in your process diagram */
public class JsfJbpmVariableResolver extends JbpmVariableResolver {
  private static final Log log = LogFactory.getLog(JbpmVariableResolver.class);
  public Object resolveVariable(String name) throws ELException {
    //let jbpm first try to resolve the variable
    Object value = super.resolveVariable(name);
    if (value == null) {
      //jbpm dont know the var, lets see whether jsf (implicitly) spring can resolve
      if (log.isDebugEnabled()) {
        log.debug(String.format("resolving %s to: %s", name, value));
      }
      value = FacesUtils.getManagedBean(name);
    }
    return value;
  }
}

```

\*Note: this class is referenced in the jbpm.cfg.xml

## jbpm + spring (spring-modules)

```

<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.0.xsd"
default-lazy-init="true">

    <!-- helper for reading jBPM process definitions -->
    <bean id="process-todo"
        class="org.springframework.modules.workflow.jbpm31.definition.ProcessDefinitionFactoryBean">
        <property name="definitionLocation"
            value="classpath:process-todo/processdefinition.xml"/>
    </bean>

    <!-- jBPM configuration -->
    <bean id="jbpmConfiguration"
        class="org.springframework.modules.workflow.jbpm31.LocalJbpmConfigurationFactoryBean">
        <property name="sessionFactory" ref="sessionFactory"/>
        <property name="configuration" value="classpath:jbpm.cfg.xml"/>
        <property name="processDefinitions">
            <list>
                <ref local="process-todo"/>
            </list>
        </property>
        <property name="createSchema" value="false"/>
    </bean>

    <!-- jBPM template -->
    <bean id="jbpmTemplate" class="org.springframework.modules.workflow.jbpm31.JbpmTemplate">
        <constructor-arg index="0" ref="jbpmConfiguration"/>
        <constructor-arg index="1" ref="process-todo"/>
        <property name="hibernateTemplate" ref="hibernateTemplate"/>
    </bean>
</beans>

```

**Note: I referenced a specific process definition**

classpath:process-todo/processdefinition.xml

you can use mine (see below) or define your own with jbpm-eclipse-plugin

## hibernate session in view

Make sure you have some open session in view filter active. I usually dont use since i fetch what i need. (web.xml)

## simple sample

### process definition

a simple workflow



```

<?xml version="1.0" encoding="UTF-8"?>
<process-definition
  name="todo"
  xmlns="urn:jbpm.org:jpdl-3.2"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="urn:jbpm.org:jpdl-3.2 http://jbpm.org/xsd/jpdl-3.2.xsd"
>
  <start-state name="start">
    <transition to="todo"></transition>
  </start-state>

  <task-node name="todo">
    <task name="Todo" blocking="true">
      <description>
        #{todoList.description}
      </description>
      <assignment pooled-actors="admin"></assignment>
      <controller></controller>
    </task>
    <transition to="end"></transition>
  </task-node>

  <end-state name="end"></end-state>
</process-definition>

```

In simple words: On starting a new process instance, a new TASK is generated. This task will be shown to all appfuse-admins (role admin). Every admin can assign the TASK to himself. Then the task-assignee can finish the task. After that the process will end.

The expression

```
#{todoList.description}
```

will be taken from JSF.

### view (facelets)

```

<html xmlns="http://www.w3.org/1999/xhtml" xmlns:c="http://java.sun.com/jstl/core"
xmlns:f="http://java.sun.com/jsf/core" xmlns:h="http://java.sun.com/jsf/html"
xmlns:ui="http://java.sun.com/jsf/facelets"
xmlns:a4j="http://richfaces.org/a4j" xmlns:rich="http://richfaces.org/rich"
xmlns:jstl="http://java.sun.com/jsp/jstl/functions"
>

<f:view>
<f:loadBundle var="text" basename="#{basePage.bundleName}"/>
  <head>

    </head>
<body id="bdy">

  <c:set var="taskInstancePriorityList" value="#{todoList.taskInstancePriorityList}"/>
  <a4j:form id="task_list_personal">
    <div>
      <h:outputText value="There are no todo items." rendered="#{empty taskInstancePriorityList}"/>
      <h2><h:outputText value="todo items." rendered="#{not empty taskInstancePriorityList}"/></h2>
      <h:dataTable value="#{taskInstancePriorityList}" var="task" rendered="#{not empty
taskInstancePriorityList}">
        <h:column>
          <f:facet name="header">
            <h:outputText value="Description"/>
          </f:facet>
          <h:inputText value="#{task.description}" style="width: 400"/>
        </h:column>

```

```

        <h:column>
            <f:facet name="header">
                <h:outputText value="Created"/>
            </f:facet>
            <h:outputText value="#{task.taskMgmtInstance.processInstance.start}">
            </h:outputText>
        </h:column>
        <h:column>
            <f:facet name="header">
                <h:outputText value="Priority"/>
            </f:facet>
            <h:inputText value="#{task.priority}" style="width: 30"/>
        </h:column>
        <h:column>
            <f:facet name="header">
                <h:outputText value="Due Date"/>
            </f:facet>
            <h:inputText value="#{task.dueDate}" style="width: 100">
            </h:inputText>
        </h:column>
        <h:column>
            <a4j:commandLink action="#{todoList.done}" value="Done" reRender="task_list_personal">
                <f:param name="taskId" value="#{task.id}"/>
            </a4j:commandLink>
        </h:column>
    </h:dataTable>
</div>
</a4j:form>

<br/>

<c:set var="pooledTaskInstanceList" value="#{todoList.pooledTaskInstanceList}"/>
<a4j:form id="task_list_pooled">
    <div>
        <h:outputText value="There are no assignable items." rendered="#{empty
pooledTaskInstanceList}"/>
        <h2><h:outputText value="assignable items." rendered="#{not empty pooledTaskInstanceList}"
/></h2>
        <h:dataTable value="#{pooledTaskInstanceList}" var="task" rendered="#{not empty
pooledTaskInstanceList}">
            <h:column>
                <f:facet name="header">
                    <h:outputText value="Description"/>
                </f:facet>
                <h:inputText value="#{task.description}" style="width: 400"/>
            </h:column>
            <h:column>
                <f:facet name="header">
                    <h:outputText value="Created"/>
                </f:facet>
                <h:outputText value="#{task.taskMgmtInstance.processInstance.start}">
                </h:outputText>
            </h:column>
            <h:column>
                <f:facet name="header">
                    <h:outputText value="Priority"/>
                </f:facet>
                <h:inputText value="#{task.priority}" style="width: 30"/>
            </h:column>
            <h:column>
                <f:facet name="header">
                    <h:outputText value="Due Date"/>
                </f:facet>
                <h:inputText value="#{task.dueDate}" style="width: 100">
                </h:inputText>
            </h:column>
        </h:dataTable>
    </div>
</a4j:form>

```

```
        <h:column>
        <h:column>
            <a4j:commandLink action="#{todoList.assign}" value="Assign" reRender=
"task_list_personal,task_list_pooled">
                <f:param name="taskId" value="#{task.id}"/>
            </a4j:commandLink>
        </h:column>
        </h:column>
    </h:dataTable>
</div>
</a4j:form>

<a4j:form id="task_new">
    <div>
        <h:inputText value="#{todoList.description}" style="width: 400"/>
        <a4j:commandButton value="Create New Item" action="#{todoList.createTodo}" reRender=
"task_list_pooled"/>
    </div>
</a4j:form>

</body>
```

```
</f:view>
</html>
```

## jsf controller (spring way)

```
package at.igsoft.templtest.webapp.list;

import java.util.HashMap;
import java.util.List;
import java.util.Map;

import org.apache.commons.lang.StringUtils;
import org.jbpm.graph.def.ProcessDefinition;
import org.jbpm.graph.exe.ProcessInstance;
import org.jbpm.taskmgmt.exe.TaskInstance;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.transaction.annotation.Transactional;
import org.springframework.util.Assert;
import org.springframework.modules.workflow.jbpm31.JbpmTemplate;

import at.igsoft.jbpm.util.JbpmAppfuseUtil;
import at.igsoft.templtest.webapp.action.IqsBasePage;

@SuppressWarnings("unchecked")
@Controller //spring bean
public class TodoList extends IqsBasePage { //my custom basepage
    private static final long serialVersionUID = 1L;
    @Autowired private JbpmTemplate jbpm; //like pityful jsf injection mech. but better

    private List<TaskInstance> taskInstancePriorityList; //tasks for current user
    private List<TaskInstance> pooledTaskInstanceList; //pooled tasks not yet assigned, but assignable to
    current user
    private String description; //field in UI for new tasks

    public TodoList() {}

    @Transactional //ok this is very unlike the appfuse way - i just like seam too ;-)
    public String done() {
        Long id = getParameterId("taskId");
        TaskInstance task = (TaskInstance) jbpm.getHibernateTemplate().get(TaskInstance.class, id);
        task.start();
        task.end();
        jbpm.getHibernateTemplate().save(task);
        //trigger view refresh
        taskInstancePriorityList = null;
        pooledTaskInstanceList = null;
        return null;
    }

    /** assign given task to current user */
    @Transactional
    public String assign() {
        Long id = getParameterId("taskId");
        TaskInstance task = (TaskInstance) jbpm.getHibernateTemplate().get(TaskInstance.class, id);
        task.setActorId(getCurrentUsername());
        jbpm.getHibernateTemplate().save(task);
        log.info(String.format("assigned task: %s to user: %s", task.getId(), getCurrentUsername()));
        //trigger view refresh
        taskInstancePriorityList = null;
        pooledTaskInstanceList = null;
        return null;
    }

    @Transactional
    public String createTodo() {
        ProcessDefinition pdef = jbpm.getProcessDefinition();
        Map<String, Object> params = new HashMap<String, Object>();
```

```

    Assert.isTrue(StringUtils.isEmpty(description), "no desc from ui");
    params.put("#{todoList.description}", description);
    ProcessInstance p = pdef.createProcessInstance(params);
    p.signal();
    jbp.getHibernateTemplate().save(p); //
log.info(String.format("new process %s started by user %s", pdef.getName(), getCurrentUsername()));
    //trigger view refresh
taskInstancePriorityList = null;
    pooledTaskInstanceList = null;
    return null;
}
@Transactional
public List<TaskInstance> getTaskInstancePriorityList() {
    if (taskInstancePriorityList == null) {
        //init
taskInstancePriorityList = jbp.findTaskInstances(getCurrentUsername());
    }
    return taskInstancePriorityList;
}
public void setTaskInstancePriorityList(
    List<TaskInstance> taskInstancePriorityList) {
    this.taskInstancePriorityList = taskInstancePriorityList;
}
@Transactional
public List<TaskInstance> getPooledTaskInstanceList() {
    if (pooledTaskInstanceList == null) {
        pooledTaskInstanceList = jbp.findPooledTaskInstances(JbpAppfuseUtil.getActorAndGroupIds());
    }
    return pooledTaskInstanceList;
}
public void setPooledTaskInstanceList(List<TaskInstance> pooledTaskInstanceList) {
    this.pooledTaskInstanceList = pooledTaskInstanceList;
}
public String getDescription() {
    return description;
}
public void setDescription(String description) {
    this.description = description;
}

```

```
}  
}
```

### some pity helper class to get all known usernames + roles for pooled actors

```
package at.igsoft.jbpm.util;  
  
import java.util.ArrayList;  
import java.util.List;  
  
import org.springframework.security.Authentication;  
import org.springframework.security.GrantedAuthority;  
import org.springframework.security.context.SecurityContextHolder;  
  
public class JbpmAppfuseUtil {  
    public static List<String> getActorAndGroupIds() {  
        List<String> result = new ArrayList<String>();  
        Authentication currentUser = SecurityContextHolder.getContext().getAuthentication();  
        result.add(currentUser.getName());  
        GrantedAuthority[] auths = currentUser.getAuthorities();  
        for (int i = 0; i < auths.length; i++) {  
            GrantedAuthority auth = auths[i];  
            result.add(auth.getAuthority());  
        }  
        return result;  
    }  
}
```

### thoughts about transaction + db object separation

Usually you have some domain tables + some jbpm tables.

- I don't want to intermix them
- I want transactional behaviour between jbpm + application-domain

possible solutions:

#### put jbpm tables to different schema in same database. example with postgres:

I put my domain-tables into public-schema and jbpm-tables to jbpm schema.

Then I alter all jbpm \*.hbm.xml files with SEARCH&REPLACE over files (jedit) and prefix all table names with "jbpm." Same goes for Foreign Key references.

#### jbpm tables to different database than domain-app-tables

use 2PC and XA with some nice jta-application server like jboss. Or tomcat/jetty + jotm/... (jta-impl for "lean" app-servers)

I posted some working configuration for seam: <http://www.seamframework.org/Community/TransactionQuestionInJbpmContextNoTransation>

## Migration Guide

The 2.x releases below are listed from most recent to least.

### Latest Release

- [Release Notes 2.1.0](#) - Easier to use and much faster. JSF 2, Tapestry 5, JPA 2 and Spring 3. CXF, RESTful services and many, many efficiency improvements.

### Past Releases

- [Release Notes 2.0.2](#) - Lots of bug fixes, library upgrades, and ability to customize code generation templates.
- [Release Notes 2.0.1](#) - Lots of bug fixes, improvements to code generation and upgraded to Spring 2.5.
- [Release Notes 2.0](#) - The road to AppFuse 2.0 has been a long journey through Mavenland, annotations and generics. We're pleased to announce that we're finally finished after 13 months of development. This release contains a number of bug fixes for AMP, an upgrade to

- Tapestry 4.1.3, the addition of Tacos, support for Oracle and changes to prevent XSS attacks.
- [Release Notes 2.0 RC1](#) - The major features in this release are JSF 1.2, Tapestry 4.1 and improved Code Generation. In addition, we've addressed over 100 issues in preparation for our final 2.0 release.
- [Release Notes 2.0 M5](#) - The major features in this release are Code Generation, Full Source and XFire support.
- [Release Notes 2.0 M4](#) - We were hoping to get AMP's code generation and XFire integrated in this release. While we're still working on those features, there were already quite a few improvements over M3, so we decided to release early. We hope to have both AMP and XFire integration completed for 2.0 M5.
- [Release Notes 2.0 M3](#) - The major feature in this release is **documentation!** All of the [web tutorials](#) tutorials have been written and tested. In addition, we figured out how to get native2ascii to work with Maven 2 - so all i18n bundles and supported languages (now including Turkish!) should work. Besides those key features, this is primarily a bug fix release. We still plan on creating a Maven plugin for code generation, as well as integrating XFire.
- [Release Notes 2.0 M2](#) - We've reached quite a milestone in the development in AppFuse 2.0. This is a release that we hope to use to flush out issues and help make AppFuse 2.0 a solid release. The major things missing from this release are code generation (AppGen) and web services (XFire) support.

## Upgrading from 1.x to 2.x

Upgrading a 1.x application to 2.x shouldn't be too difficult, it's all Java and XML after all! The first thing you should be aware of is that AppFuse 2.x doesn't support Struts 1. This means if your AppFuse 1.x application uses Struts 1.x, it's probably not worth upgrading. In fact, continuing to use your 1.x application as-is may be the best solution. The only *good* reason to try to upgrade your application to 2.x is if you want to use Maven instead of Ant.

To upgrade, you'll want to create a new project using the [QuickStart Guide](#). Then you'll want to run **mvn appfuse:full-source** to convert your new project to contain all of AppFuse source (like 1.x did by default).



After you've setup your new project, check it into source control so you can rollback if you make any mistakes. Creating a backup copy on your hard-drive is also a recommended solution.


















































1. Copy Java source files from `src/dao`, `src/service` and `src/web` to `src/main/java`. If you're using a modular-structured project, copy `src/dao/**` and `src/service/**` to **`core/src/main/java`**.
2. Change your Model objects from using mapping files to using JPA Annotations.
3. Copy web files (CSS, JavaScript, etc.) to `src/main/webapp` (**`web/src/main/webapp`** for modular projects).

We realize this is not the step-by-step guide most folks are looking for. Please ask on the [user mailing list](#) if you have any troubles. Also, there are several AppFuse Developers who'd be happy to help with your migration. Please [contact Matt](#) if you'd like to contract one of them to assist you.
















## Release Notes 2.1.0 M1

### Detailed Changelog





















AppFuse Issues (45 issues)				
Type	Key	Summary	Status	Resolution
	APF-1166	Upgrade to Canoo WebTest 3.0	 Resolved	Fixed
	APF-1165	Change from Core JSF Validator to Shale Validator	 Resolved	Fixed
	APF-1164	Remove OSCache	 Resolved	Fixed
	APF-1158	Fix for JSF "Upload A File" bug	 Resolved	Fixed
	APF-1153	Use skipTests instead of maven.test.skip	 Resolved	Fixed
	APF-1150	Remove User parameter from UserManager.getUsers()	 Resolved	Fixed
	APF-1145	PMD has failures (concentrated on struts basic)	 Resolved	Fixed
	APF-1144	checkstyle errors (focus on struts 2.0 basic)	 Resolved	Fixed
	APF-1135	Upgrade to UrlRewriteFilter 3.1.0	 Resolved	Fixed
				

	APF-1131	Remove Clickstream	Resolved	Fixed
	APF-1129	Upgrade to Cargo 1.0-beta-2 to support tomcat6x	 Resolved	Fixed
	APF-1128	Upgrade to SiteMesh 2.4	 Resolved	Fixed
	APF-1127	When using mvn appfuse:remove command in struts2 basic archetype, it removes all validation files (xml)	 Resolved	Fixed
	APF-1125	Upgrade from XFire to CXF	 Resolved	Fixed
	APF-1119	Update BaseDaoTestCase to use AbstractTransactionalJUnit4SpringContextTests	 Resolved	Fixed
	APF-1117	appfuse:full-source removes .svn/tmp folder causing svn client to fail	 Resolved	Fixed
	APF-1116	Synch with latest tapestry5-appfuse	 Resolved	Fixed
	APF-1115	displaytag message is corrupted when language is in Tranditional Chinese	 Resolved	Fixed
	APF-1114	Upgrade to Tapestry 5	 Resolved	Fixed
	APF-1112	static ResourceBundle in BaseManagerTestCase	 Resolved	Fixed
	APF-1110	Errors when creating default appfuse project and importing it in Eclipse	 Resolved	Fixed
	APF-1109	Old Xerces version bug when using JPA as dao.framework	 Resolved	Fixed
	APF-1105	Change to use Spring annotations (@Repository, @Service and @Autowired) in service and data modules	 Resolved	Fixed
	APF-1101	Upgrade to maven-resources-plugin 2.3	 Resolved	Fixed
	APF-1100	Error in security.xml XML comments, suggesting user comment out wrong line if not using JSF	 Resolved	Fixed
	APF-1099	speed up db population in dbunit	 Resolved	Fixed
	APF-1096	Upgrade to Spring Security 2.0.4	 Resolved	Fixed
	APF-1094	Upgrade to Spring 2.5.5	 Resolved	Fixed
	APF-1089	Upgrade Hibernate: Core 3.3.0.SP1, Annotations 3.4.0.GA and EntityManager 3.4.0.GA	 Resolved	Fixed
	APF-1086	Upgrade to EhCache 1.5.0	 Resolved	Fixed
	APF-1084	Issue starting up appfuse once mvn appfuse:full-source is run	 Resolved	Cannot Reproduce
	APF-1083	Upgrade to Ant 1.7.1	 Resolved	Fixed
	APF-1081	appfuse:gen goal does not work in core project	 Resolved	Fixed
	APF-1073	Profiles seem to be broken in Appfuse 2.0.2, Error executing database operation: CLEAN_INSERT	 Resolved	Fixed



	APF-1072	Struts demo code in appfuse-demos-2.0.2.tar.gz needs to be updated for Appfuse 2.0.2	 Resolved	Fixed
	APF-1070	Link to "All Streams" on page admin/viewstream.jsp is broken, linking to /admin/clickstreams.jspjsp	 Resolved	Fixed
	APF-1066	JSF + iBATIS: NoClassDefFoundError: org/apache/commons/collections/map/LinkedMap	 Resolved	Fixed
	APF-1046	Upgrade Cargo to use Tomcat 6.0.18	 Resolved	Fixed
	APF-1018	Convert JSP files to facelets in JSF version of AppFuse 1.x	 Resolved	Won't Fix
	APF-950	Subversion-with-AppFuse mini-tutorial	 Resolved	Fixed
	APF-936	BaseDaoTestCase#populate( Object ) does not work	 Resolved	Fixed
	APF-684	Upgrade to Velocity 1.5	 Resolved	Won't Fix
	APF-531	Change to Ajax4JSF's FastFilter for faster page loading	 Resolved	Won't Fix
	APF-365	Model Classes are missing testcases	 Resolved	Won't Fix
	APF-116	When CATALINA_BASE is defined it should be used instead of CATALINA_HOME	 Resolved	Fixed





#### AppFuse Light Issues (10 issues)















































Type	Key	Summary	Status	Resolution
	EQX-207	Upgrade to SiteMesh 2.4	 Resolved	Fixed
	EQX-206	Upgrade to JWebUnit 2.1	 Resolved	Fixed
	EQX-204	Validator in UserFormController doesn't work	 Resolved	Fixed
	EQX-202	Add Automatic Validation with annotations to Spring MVC	 Resolved	Fixed
	EQX-201	Develop system to share web assets and reduce duplication	 Resolved	Fixed
	EQX-200	Move from Ant to Maven Modular project	 Resolved	Fixed
	EQX-199	Add Ajaxified Body to all Web Frameworks	 Resolved	Fixed
	EQX-198	Upgrade to jWebUnit 2.0	 Resolved	Fixed
	EQX-197	Change to use AppFuse Backend (a.k.a. remove OJB, Spring JDBC and JDO)	 Resolved	Fixed
	EQX-68	Create (and publish) archetypes of AppFuse Light combinations	 Resolved	Fixed



















































## Release Notes 2.1.0 M2





























### Detailed Changelog

#### AppFuse Issues (65 issues)







Type	Key	Summary	Status	Resolution
	APF-1211	Maven WarPath plugin does not work correctly with Maven 3	 Resolved	Fixed
	APF-1210	Remove AspectJ Plugin	 Resolved	Fixed

	APF-1204	package name wrong "double webapp"	 Resolved	Fixed
	APF-1203	drop-down menu disappears when the pointer passes over another tab	 Resolved	Fixed
	APF-1202	appfuse:gen not fully autowired	 Resolved	Fixed
	APF-1201	Upgrade to Tomcat 6.0.29	 Resolved	Fixed
	APF-1200	upgrade checkstyle from 4.4 to 5	 Resolved	Fixed
	APF-1199	test error after execute mvn appfuse:full-source	 Resolved	Cannot Reproduce
	APF-1197	calendar position is wrong in AppFuse JSF	 Resolved	Fixed
	APF-1193	Upgrade to Spring 3.0.5 and Spring Security 3.0.4	 Resolved	Fixed
	APF-1189	It seems pom.xml is not updating after running command mvn appfuse:full-source	 Resolved	Fixed
	APF-1188	Upgrade to Tomcat 6.0.24 for Memory Protection	 Resolved	Fixed
	APF-1183	Appfuse blocks loading of alternative properties files	 Resolved	Fixed
	APF-1182	Upgrade to Spring Security 3.0	 Resolved	Fixed
	APF-1179	Upgrade to Spring 3.0	 Resolved	Fixed
	APF-1177	Basic Struts Archetype results in two webapps directories in src/main/java	 Resolved	Fixed
	APF-1176	mvn appfuse:full-source in appfuse-modular archetypes does not include calculated properties in pom.xml	 Resolved	Fixed
	APF-1175	mvn appfuse:full-source in appfuse-modular archetypes produces duplicated tag modules	 Resolved	Fixed
	APF-1172	Generated Web tests won't work when entities have instance variable prefixed with underscore	 Resolved	Fixed
	APF-1171	Bump version of Compass to 2.2.0	 Resolved	Fixed
	APF-1170	Upgrade to Hibernate 3.3.2.GA	 Resolved	Fixed
	APF-1169	Upgrade to Spring 2.5.6.SEC01	 Resolved	Fixed
	APF-1168	appfuse:full-source can't handle milestone releases	 Resolved	Fixed
	APF-1159	Add Polish versions of properties and errors	 Resolved	Fixed
	APF-1154	Editing (some) users sometimes results in the user being inserted again..	 Resolved	Fixed
	APF-1151	appfuse gen-core not working for JPA	 Resolved	Cannot Reproduce
		Error executing ant tasks - Wrong document title found!. Expected value ". *Menú Principal.*" but		

	APF-1148	got "Registro   AppFuse"		Resolved	Fixed
	APF-1147	mailengine does not throw exception		Resolved	Won't Fix
	APF-1146	to improve code coverage add the following patches		Resolved	Fixed
	APF-1130	Upgrade to Struts 2.1		Resolved	Fixed
	APF-1121	Upgrade to H2 1.0.79		Resolved	Fixed
	APF-1120	mvn appfuse:full-source on a appfuse-basic-struts projects generates a duplicate hibernate.cfg.xml in test resources		Resolved	Fixed
	APF-1113	Encryption problem for Role_Admin users		Resolved	Cannot Reproduce
	APF-1091	internet explorer problem with commandLink having request parameter with name "id"		Resolved	Fixed
	APF-1082	Objects are committed to database in web module tests generated using AppFuse Maven Plugin		Resolved	Fixed
	APF-1078	JBoss Richfaces integration issue -- clicking add button on users.xhtml does not navigate to userForm.xhtml page.		Resolved	Fixed
	APF-1076	Better Ant support		Resolved	Won't Fix
	APF-1075	Problem in edit and save profile after form validation		Resolved	Fixed
	APF-1069	Error creating a new user.		Resolved	Fixed
	APF-1054	Support ICEfaces 1.7.0		Resolved	Fixed
	APF-1044	Pages always show up in English although browser's preferred language is another language		Resolved	Cannot Reproduce
	APF-1037	Blank Page when providing a base expression in a JSP		Resolved	Fixed
	APF-1034	CLEAN_INSERT error with projectnames with dots in		Resolved	Fixed
	APF-1025	XML and locale properties files should be UTF-8 encoded - but are ansi		Resolved	Fixed
	APF-1020	SiteMesh decorates Ajax requests		Resolved	Fixed
	APF-1002	Add facility to control Hibernate SQL Logging from pom.xml and/or profiles.xml		Resolved	Won't Fix
	APF-1001	Page not Found using appfuse:gen to generate: Spring Modular		Resolved	Fixed
	APF-995	IllegalArgumentException when running mvn appfuse:full-source -- Illegal character in path at index 18: file:/C:/Dokumente und Einstellungen		Resolved	Fixed
	APF-992	pom.version is not replaced in ApplicationResources_(language code).properties		Resolved	Fixed
	APF-984	Upgrade Spring MVC Controllers to use annotations		Resolved	Fixed
	APF-979	server side validation and inputHidden fields causes problems		Resolved	Fixed

	APF-970	mvn appfuse:full-source fails with Illegal characters in path	 Resolved	Fixed
	APF-963	DWR does not work out of the box	 Resolved	Fixed
	APF-952	InterBase support: Add primary key support using sequences	 Resolved	Won't Fix
	APF-920	mvn appfuse:full-source failed with svn: Connection timed out: connect	 Resolved	Won't Fix
	APF-914	Cobertura doesn't work when aspectj-maven-plugin is enabled	 Resolved	Fixed
	APF-901	Reduce possibility of memory leaks	 Resolved	Won't Fix
	APF-890	Creation of a separate appfuse-dao module for DAO api (interfaces)	 Resolved	Won't Fix
	APF-831	File displaytag_zh_CN.properties included in appfuse-web-common-2.0-m5.war is wrong	 Resolved	Cannot Reproduce
	APF-804	NPE in XWorkConverter.loadConversionProperties after adding DateConverter	 Resolved	Fixed
	APF-658	Document how to make Resin 3.x aware of Spring 2.0 when using Cargo	 Resolved	Won't Fix
	APF-645	Warpath classpath not resolving in Netbeans 5.5	 Resolved	Fixed
	APF-528	Create a web application that allows users to create an AppFuse or Equinox application from a web page	 Resolved	Fixed
	APF-419	Change UserSQL.xml to automatically fetch roles instead of manually doing it	 Resolved	Won't Fix
	APF-158	Add Beandoc support	 Resolved	Won't Fix

#### AppFuse Light Issues (3 issues)

Type	Key	Summary	Status	Resolution
	EQX-208	Upgrade to Spring 3.0.2	 Open	Unresolved
	EQX-205	Submitting non-ASCII charactes in Freemarker leads to garbled characters	 Open	Unresolved
	EQX-203	Move web modules to AppFuse SVN	 Open	Unresolved

## Release Notes 2.1.0

This release contains several bug fixes as well REST support, upgrades (JPA 2, CXF, JSF 2, Tapestry 5, Spring 3) and restructured archetypes that make everything **much** faster to run. In addition, there's a new **appfuse-ws** archetype that leverages [Enunciate](#). Below is a highlighted list of significant enhancements since 2.0.2:

### AppFuse

- [\[APF-1221\]](#) - Upgraded to JSF 2
- [\[APF-1114\]](#) - Upgraded to Tapestry 5
- [\[APF-1125\]](#) - Upgraded from XFire to CXF
- [\[APF-897\]](#) - Added appfuse-ws as an archetype
- [\[APF-1244\]](#) - Added support for RESTful Services
- [\[APF-1193\]](#) - Upgraded to Spring 3.0.5 and Spring Security 3.0.5
- [\[APF-1171\]](#) - Upgraded Compass to 2.2.0
- [\[APF-267\]](#) - Added search feature to list screen generation
- [\[APF-1212\]](#) - Changed all web frameworks to use Extensionless URLs
- [\[APF-1218\]](#) - Upgraded to JPA 2.0
- [\[APF-1195\]](#) - Upgraded to Hibernate 3.6.1
- [\[APF-1105\]](#) - Changed to use Spring annotations (@Repository, @Service and @Autowired) in service and data modules

- [\[APF-984\]](#) - Upgraded Spring MVC Controllers to use annotations
- [\[APF-1130\]](#) - Upgraded to Struts 2.1
- [\[APF-1131\]](#) - Removed Clickstream
- [\[APF-1164\]](#) - Removed OSCache
- [\[APF-1166\]](#) - Upgraded to Canoo WebTest 3.0
- [\[APF-970\]](#) - Fixed appfuse:full-source when spaces in project path
- [\[APF-1201\]](#) - Upgraded to Tomcat 6.0.29 for Memory Protection
- [\[APF-1012\]](#) - Extended appfuse maven plugin to allow sub-packages
- [\[APF-1119\]](#) and [APF-1245](#) - Upgraded to JUnit 4 for all tests

## AppFuse Light

- [\[EQX-68\]](#) - Created archetypes of AppFuse Light combinations
- [\[EQX-197\]](#) - Changed to use AppFuse Backend (a.k.a. remove OJB, Spring JDBC and JDO)
- [\[EQX-197\]](#) - Added Ajaxified Body to all Web Frameworks
- [\[EQX-200\]](#) - Moved from Ant to a Maven Modular project
- [\[EQX-209\]](#) - Upgraded Wicket to 1.4.15
- [\[EQX-213\]](#) - Upgraded to JSF 2
- [\[EQX-214\]](#) - Integrated Extensionless URLs
- [\[EQX-210\]](#) - Integrated AMP into AppFuse Light

In addition, a number of blog posts were written about features that went into this release while it was being developed:

- [Adding Search to AppFuse with Compass](#)
- [Upgrading to JSF 2](#)
- [Fixing XSS in JSP 2](#)
- [Implementing Extensionless URLs with Tapestry, Spring MVC, Struts 2 and JSF](#)
- [AppFuse 2.1 Milestone 2 Released](#)
- [Using JRebel with IntelliJ IDEA on OS X](#)
- [AppFuse 2.1 Milestone 1 Released](#)
- [AppFuse Light converted to Maven modules, upgraded to Tapestry 5 and Stripes 1.5](#)
- [My Experience with Java REST Frameworks \(specifically Jersey and CXF\)](#)
- [Upgrading Hibernate to 3.4.0 and AppFuse for Tapestry 5](#)

Demos for this release can be viewed at <http://demo.appfuse.org>. If you don't see a list of AppFuse and AppFuse Light demos, try clearing your browser's cache (shift + reload).

Please see the [Upgrade Guide](#) below or the [QuickStart Guide](#) to get started with this release. Individual issues fixed can be seen in the [changelog](#), as well as the release notes for [2.1.0 M1](#) and [2.1.0 M2](#).

## Upgrade Guide

There are many things that changed between AppFuse 2.0.2 and AppFuse 2.1. Most significant, archetypes now include all the source for the web modules so using `jetty:run` and your IDE will work much smoother now. The backend is still embedded in JARs, enabling you to choose with persistence framework (Hibernate, iBATIS or JPA) you'd like to use. If you want to modify the source for that, [add the core classes to your project](#) or run `appfuse:full-source`.



### Diff is your friend

The easiest way to upgrade is likely to create a new project using 2.1.0, then compare the top-level directory of your project with the new one.

- [Beyond Compare](#) is a fabulous diff tool for Windows users.
- [WinMerge](#) is a great open source visual diff and merging tool for Windows.
- [SmartSynchronize](#) is a multi-platform file and directory compare tool.

The [tutorial applications](#) have been upgraded from 2.0.2 to 2.1.0.

## The AppFuse Maven Plugin

This plugin currently does two things: 1) code generation for CRUD and 2) allows you to convert your project to use AppFuse's source instead of using its binary dependencies.

### Generating CRUD with AMP

You can run the following command to generate CRUD screens/classes for a POJO:

```
mvn appfuse:gen -Dentity=Name
```



### Web Tests Generation Bug

We used incorrect logic when deciding if WebTest support was included in your project. For this reason, you may see the following error when generating code:

```
[info] [AppFuse] Project doesn't use Canoo WebTest, disabling UI test generation.  
[info] [AppFuse] Support for jWebUnit will be added in a future release.  
[info] [AppFuse] See http://issues.appfuse.org/browse/EQX-215 for more information.
```

To fix this problem, perform one of the following steps:

- Create an empty file at `target/appfuse/generated-sources/src/test/resources/{Class}-web-tests.xml`.
- Upgrade the `appfuse-maven-plugin` in your project to use version `2.1.1-SNAPSHOT`.

After doing this, you should be able to run `mvn appfuse:gen -Dentity={Class}` successfully.

If you don't specify the entity name, you will be prompted for it. Currently, if a `@Column` has `"nullable = false"`, a "required field" validation rule will be generated as part of the web framework's validation configuration. This command will also install the generated code, unless you specify `-DdisableInstallation=true`.

If your entity is not defined in `hibernate.cfg.xml`, it will be added.

In a modular project, these commands must be run in the "core" and "web" modules. The plugin is smart enough to figure out when it should/should not generate stuff based on the packaging type (jar vs. war). If you want to generate specific code in a "war" project, you can use **gen-core** or **gen-web**.



### Removing Code

You can run `mvn appfuse:remove` to remove the artifacts installed by `appfuse:gen`.

There's also a goal that allows you to generate model objects from database tables.

```
appfuse:gen-model
```

This goal will install the generated files into your source tree, unless you specify `-DdisableInstallation=true`. After running this command, you can use **appfuse:gen** to generate CRUD classes/tests/screens for this object.

If you want to customize the code generation templates (written in FreeMarker), you can copy them into your project using the following command:

```
appfuse:copy-templates
```

### Installing AppFuse's source into your project

Creating a project with no dependencies on AppFuse is very easy. After you've create a new project, run the following command:

```
mvn appfuse:full-source
```



### Issues with Maven 3

This command **won't work with Maven 3**. Please use Maven 2.2.1 to run this command (you only have to do it once per project).

This goal will convert your project to use all of AppFuse's source and remove all dependencies on AppFuse.



























What the full-source plugin does:



























1. Exports all sources from Subversion into your project. It reads the `dao.framework` and `web.framework` properties to determine what you need.











2. Calculates dependencies by reading pom.xml files from the various AppFuse modules. It replaces your dependencies with these new ones. The order of the dependencies added is alphabetical based on groupId.
3. Reads properties from the root AppFuse pom.xml and adds the ones that don't exist to your project. The order of the properties added is alphabetical.
4. Renames packages from `org.appfuse` to your project's groupId.

## Detailed Changelog

Also see the release notes from [2.1.0 M1](#) and [2.1.0 M2](#) for previous improvements as part of this release.

AppFuse Issues (33 issues)				
Type	Key	Summary	Status	Resolution
	APF-1245	Update web unit tests to JUnit 4	 Resolved	Fixed
	APF-1244	Add support for RESTful Services	 Resolved	Fixed
	APF-1242	Remove Eclipse and IDEA Plugins	 Resolved	Fixed
	APF-1238	Cancelling on Signup tries to do a save	 Resolved	Fixed
	APF-1232	Several tutorial pages at <a href="http://www.appfuse.com">http://www.appfuse.com</a> are showing up with blank content	 Resolved	Fixed
	APF-1230	Update demos on <a href="http://demo.appfuse.org">demo.appfuse.org</a> with 2.1.0 release	 Resolved	Fixed
	APF-1229	Update Tutorials and Demos to use the latest release	 Resolved	Fixed
	APF-1228	Change JSP EL to escape XML by default	 Resolved	Fixed
	APF-1225	Upgrade to Tapestry 5.2.4	 Resolved	Fixed
	APF-1224	Reload Options doesn't display a message in JSF	 Resolved	Fixed
	APF-1221	Upgrade to JSF 2	 Resolved	Fixed
	APF-1218	Upgrade to JPA 2.0	 Resolved	Fixed
	APF-1217	mvn site fails because of the aspectj-maven-plugin not exist!	 Resolved	Fixed
	APF-1214	i18n - encoding "8859_" and Resource filtering doesn't work correctly.	 Resolved	Fixed
	APF-1213	Fix AppFuse Archetype Signing Issues	 Resolved	Fixed
	APF-1212	Change all web frameworks to use Extensionless URLs	 Resolved	Fixed
	APF-1195	Upgrade to Hibernate 3.6	 Resolved	Fixed
	APF-1184	java.lang.IllegalStateException: The PluginDescriptor for the plugin Plugin [org.apache.maven.plugins:maven-eclipse-plugin] was not found.	 Resolved	Fixed
	APF-1173	Create instructions for creating a local repository	 Open	Unresolved
	APF-1157	can't use slashes in action name, they are always use for namespace	 Resolved	Fixed

	APF-1133	"appfuse:remove" doesn't remove JSP generated files in a Spring project	 Resolved	Fixed
	APF-1126	The "prod" profile defined in the parent pom should be moved to the core pom in a modular application	 Resolved	Fixed
	APF-1108	Check if Cobertura is working in Archetypes	 Resolved	Fixed
	APF-1092	appfuse automatically logs out after signup	 Resolved	Fixed
	APF-1088	If class name has embedded capitals, AppGen code fails UI tests, and generated hyperlinks are incorrect.	 Resolved	Fixed
	APF-1019	JPA appfuse project results in a javax.persistence.OptimisticLockException when saving a User a second time (sequentially)	 Resolved	Fixed
	APF-1012	Extend appfuse maven plugin to allow sub-packages	 Resolved	Fixed
	APF-924	Poorman's dynamic finder: findAllWhere(....)	 Resolved	Fixed
	APF-897	Add appfuse-ws as an archetype	 Resolved	Fixed
	APF-843	Change menu selection logic to use CSS	 Resolved	Won't Fix
	APF-576	Introduce new exception class when an Object is not found	 Resolved	Won't Fix
	APF-349	remove confirmPassword from User domain class	 Resolved	Won't Fix
	APF-267	Add search feature to list screen generation	 Resolved	Fixed

AppFuse Light Issues (5 issues)				
Type	Key	Summary	Status	Resolution
	EQX-214	Integrate Extensionless URLs	 Resolved	Fixed
	EQX-213	Upgrade to JSF 2	 Resolved	Fixed
	EQX-212	Update demos on demo2.appfuse.org with 2.1.0 release	 Open	Unresolved
	EQX-210	Integrate AMP into AppFuse Light	 Resolved	Fixed
	EQX-209	Upgrade Wicket to 1.4.15 (patch)	 Resolved	Fixed

## Persistence

### About this Tutorial

This tutorial creates a new database table and the Java code to access that table.

You create an object and then some more classes to persist (save/retrieve/delete) that object from the database. In Java speak, this object is called a Plain Old Java Object (POJO). This object basically represents a database table. With AppFuse 1.x, you typically created a DAO and JUnit Test to persist this POJO. However, with AppFuse 2.x, there is a Generics-based DAO and Manager that will CRUD all objects for you. The only time you'll need to create DAOs is when you want custom behavior or if you need finders.

AppFuse uses [Hibernate](#) for its default persistence layer. Hibernate is an Object/Relational (O/R) Framework that allows you to relate your Java Objects to database tables. It allows you to very easily perform CRUD (Create, Retrieve, Update, Delete) on your objects.





### iBatis and JPA

You can also use [iBatis](#) or [JPA](#) as a persistence framework option. To use iBatis with AppFuse, see the [Using iBatis](#) tutorial. To use JPA implementation, see the [Using JPA](#) tutorial.

To get started creating a new Object and table in AppFuse's project structure, please complete the instructions below.

## Table of Contents

1. [Create a new POJO and add JPA Annotations](#)
2. [Create a new database table from the object using Maven](#)

## Create a new POJO and add JPA Annotations

The first thing you need to do is create an object to persist. Create a simple "Person" object (in the `src/main/java/**/model` directory for the basic archetypes or the `core/src/main/java/**/model` directory for the modular archetypes) that has an id, a firstName and a lastName (as properties). For recommend package-naming conventions, see the [FAQ](#). These tutorials use "org.appfuse.tutorial" as the root package name.

```
package org.appfuse.tutorial.model;

import org.appfuse.model.BaseObject;

import javax.persistence.Entity;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.GeneratedValue;
import javax.persistence.Column;

public class Person extends BaseObject {
    private Long id;
    private String firstName;
    private String lastName;

    /*
     * Generate your getters and setters using your favorite IDE:
     * In Eclipse:
     * Right-click -> Source -> Generate Getters and Setters
     */
}
```



Extending `BaseObject` is optional, but recommended as a good practice to force creation of `toString()`, `equals()` and `hashCode()` methods. If you plan to put this object into the user's session, or expose it through a web service, you should implement `java.io.Serializable` as well.

To generate `toString()`, `equals()` and `hashCode()` methods, you can use [Commonclipse](#). Another Eclipse Plugin you can use is [Commons4E](#). When generating or writing `equals()` and `hashCode()` methods, you don't want to include your object's primary key. See Hibernate's [Equals and Hascode](#) for more information.



### IntelliJ Users

If you're using [IntelliJ IDEA](#), you can generate `equals()` and `hashCode()`, but not `toString()`. There is a [ToStringPlugin](#) that works reasonably well.



### Netbeans Users

If you're using [Netbeans 6.0](#), you can generate `equals()` and `hashCode()`, using key combination `Alt+Insert` -> chose override method -> `hashCode`, `equals`. You can also use [CodeGen](#) plugin

Now that you have this POJO created, you need to add [JPA annotations](#). These annotations are used by Hibernate to map objects tables and properties (variables) columns.



If you're new to Hibernate and Annotations, you might want to read [An Introduction to Hibernate 3 Annotations](#) by John Ferguson Smart.

First of all, add an `@Entity` annotation that signifies what table this object relates to. The "name" is optional; the class name is used if it's not specified. Make sure you import annotations from `javax.persistence.*` rather than from Hibernate.

```
@Entity
public class Person extends BaseObject {
```



If you specify a name value for your `@Entity` annotation (for example `@Entity(name="person")`), this will be the alias for HQL queries. If you don't specify this value, the name will match the short name of your class (`Person`). If you want to change the table name that's generated, use the `@Table` annotation with a "name" value.

You also have to add an `@Id` annotation to signify the primary key. The `@GeneratedValue` annotation should also be specified to indicate the primary key generation strategy.

```
@Id @GeneratedValue(strategy = GenerationType.AUTO)
public Long getId() {
    return this.id;
}
```

For the rest of the fields, you aren't required to annotate them unless you want to 1) exclude them from being persisted (with `@Transient`) or 2) want to change their column name or other attributes. To change the column names, use the `@Column` annotation. Add the `@Column` annotation to both the `getFirstName()` and `getLastName()` methods.

```
@Column(name="first_name", length=50)
public String getFirstName() {
    return this.firstName;
}
...
@Column(name="last_name", length=50)
public String getLastName() {
    return this.lastName;
}
```



#### Annotations on fields

You can also put JPA annotations on fields instead of *getters*. However, you should be aware that if you add field-level annotations, *method-level annotations will be ignored*.

## Create a new database table from the object using Maven

Open `src/main/resources/hibernate.cfg.xml` for the basic archetypes (or `core/src/main/resources/hibernate.cfg.xml` for the modular archetypes) and register your `Person` object with the following XML:

```
<mapping class="org.appfuse.tutorial.model.Person"/>
```

Save the file and run `mvn test-compile hibernate3:hbm2ddl` from the command line. For modular projects, make sure you run this command from the "core" directory. This will generate your database schema with the "person" table included.

```
create table person (id bigint not null auto_increment, first_name varchar(50), primary key (id))
type=InnoDB;
```

After you've created a POJO and generated a schema from it, how do you persist that object? AppFuse ships with GenericDao implementations that makes it possible to CRUD any object. In addition to this [Generics](#)-based class, there is a UniversalDao that does the same thing. The major difference between the two is you'll need to cast to your specified type for the UniversalDao. However, it also doesn't require you to define any new Spring beans, so there are benefits and drawbacks. For these tutorials, you will learn how to program using the GenericDao.

Please choose the persistence framework you'd like to use to continue:



If you don't know which is better for your project, please read [Hibernate vs. iBATIS](#).

## AppFuse Core Classes

### *About this Tutorial*

There is going to come a time in the development of your project when you will need to make changes to one or more of the AppFuse core model classes. This tutorial takes you through the seeding of your project with the source code for these classes, and gives some guidelines as to what you can and what you cannot change without breaking AppFuse.



#### **Be Careful**

Please be careful when changing the AppFuse model classes. The functionality in the security layer as well as many of the controllers rely on these classes exposing a core set of attributes. The old adage applies: test everything thoroughly!

### *Before you Start*

In order to complete the tutorial you need to have a recent version of the Subversion command line client installed. Binaries and install instructions for the client are available [here](#).

### *Get the Model Class Code*

The process varies slightly according to which AppFuse archetype you are using:

1. If you are using one of the basic archetypes, open a command prompt at the root of the project (on the directory that contains the pom.xml).
2. If you are using one of the modular archetypes, open a command prompt at the root of the *core* module (on the directory that holds the pom.xml file for the core module).

Now enter the following at the command prompt:

```
svn export --force https://svn.java.net/svn/appfuse-svn/trunk/data/common/src --username guest
```

If all goes well, a set of source files have been downloaded into the src folder of your project. These are the AppFuse core model classes - open them in your favourite IDE and spend some time browsing through them. The ones you are interested in are held in the package org.appfuse.model. The download also includes some interfaces from the org.appfuse.dao package - please leave these as they are.



#### **Leave Package and Class Names As They Are**


You are free to change the class attributes and methods as you see fit, but you must leave the class package and names unchanged. AppFuse depends on it!

### *Exclude the AppFuse Data Common Package*

You now have all the core AppFuse classes building in your own project, so to avoid any possible classpath clashes you need to exclude the data

common module bundled as part of AppFuse.

Add the following <exclusions>..</exclusions> to the existing dependencies in pom.xml. For basic archetypes, add them in pom.xml at the project root.

 Note that the first dependency below is <type>warpath</type>, not <type>war</type> (which you should find immediately before the warpath one).

```
<dependency>
  <groupId>org.appfuse</groupId>
  <artifactId>appfuse-${web.framework}</artifactId>
  <version>${appfuse.version}</version>
  <type>warpath</type>
  <exclusions>
    <exclusion>
      <groupId>org.appfuse</groupId>
      <artifactId>appfuse-data-common</artifactId>
    </exclusion>
  </exclusions>
</dependency>

...

<dependency>
  <groupId>org.appfuse</groupId>
  <artifactId>appfuse-${dao.framework}</artifactId>
  <version>${appfuse.version}</version>
  <exclusions>
    <exclusion>
      <groupId>org.appfuse</groupId>
      <artifactId>appfuse-data-common</artifactId>
    </exclusion>
  </exclusions>
</dependency>
```

For modular archetypes, add the following <exclusions>..</exclusions> to the existing dependency in core/pom.xml:

```
<dependency>
  <groupId>org.appfuse</groupId>
  <artifactId>appfuse-${dao.framework}</artifactId>
  <version>${appfuse.version}</version>
  <exclusions>
    <exclusion>
      <groupId>org.appfuse</groupId>
      <artifactId>appfuse-data-common</artifactId>
    </exclusion>
  </exclusions>
</dependency>
```

**Model Class Attribute Dependencies**

Much of the functionality that ships with AppFuse depends on the presence of a core set of attributes on the model classes. The following table lists these dependencies.

Function	Controller	Attributes/Interface/Method
Security	ACEGI	User class must implement org.acegisecurity.userdetails.UserDetails
User Profile	UserAction/UserFormController/UserForm	User.password, User.version, User.username, User.confirmPassword, User.email, User.roles, User.addRole(), User.getFullName(), Role.name

Password Hint	PasswordHintAction/PasswordHintController/PasswordHint	User.email, User.username, User.passwordHint
Active User List	ActiveUserList	User.username, User.getFullName() (JSF only)
User Signup	SignupAction/SignupController/SignupForm	User.addRole(), User.password, User.username, User.email, User.getFullName()

Note that a dependency does not mean that you absolutely cannot remove/change the related attribute - it simply means that that piece of functionality will cease to work. Only change the attribute or method if you do not need the functionality in your application!

## Using Hibernate

### About this Tutorial

This tutorial shows two things:

1. You don't need to write DAOs if you just need generic CRUD functionality.
2. How to write DAOs when you need custom functionality.

If you're new to Hibernate, you might want to read the [Hibernate Reference Guide](#) before starting this tutorial.

### Table of Contents

1. [Register a personDao bean definition](#)
2. [Create a DAO Test to test finder functionality](#)
3. [Create a DAO Interface and implementation](#)
4. [Run the DAO Test](#)



#### Source Code

The code for this tutorial is located in the "tutorial-hibernate" module of the [appfuse-demos](#) project on Google Code. Use the following command to check it out from Subversion:

```
svn checkout http://appfuse-demos.googlecode.com/svn/trunk/tutorial-hibernate
```

### Register a personDao bean definition

AppFuse 2.x doesn't require you to write a DAO to persist a POJO. You can use the [GenericDaoHibernate](#) class if all you need is CRUD on an object:

To register a personDao bean, open `src/main/webapp/WEB-INF/applicationContext.xml` (or `core/src/main/resources/applicationContext.xml` for a modular archetype) and add the following to it:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

    <bean id="personDao" class="org.appfuse.dao.hibernate.GenericDaoHibernate">
        <constructor-arg value="org.appfuse.tutorial.model.Person"/>
    </bean>
</beans>
```

After doing this, you can use this bean as a dependency of an object by adding the following setter method:

```
@Autowired
public void setPersonDao(GenericDao<Person, Long> personDao) {
    this.personDao = personDao;
}
```

If you need more than just CRUD functionality, you'll want to continue reading below. If not, you can continue to [Creating new Managers](#). This is a tutorial for creating Business Facades, which are similar to [Session Facades](#), but don't use EJBs. These facades are used to provide communication from the front-end to the DAO layer.

### Create a DAO Test to test finder functionality

Now you'll create a `DaoTest` to test that your DAO works. "Wait a minute," you say, "I haven't created a DAO!" You are correct. However, I've found that [Test-Driven Development](#) breeds higher quality software. For years, I thought **write your test before your class** was hogwash. It just seemed stupid. Then I tried it and I found that it works great. The only reason I do test-driven stuff now is because I've found it rapidly speeds up the process of software development.

To start, create a `PersonDaoTest.java` class in your `src/test/java/**/dao` directory (or `core/src/test/java/**/dao` directory for a modular archetype). This class should extend `org.appfuse.dao.BaseDaoTestCase`, a subclass of Spring's [AbstractTransactionalJUnit4SpringContextTests](#). This parent class is used to load Spring's `ApplicationContext` (since Spring binds interfaces to implementations), and for (optionally) loading a `.properties` file that has the same name as your `*Test.class`. In this example, if you put a `PersonDaoTest.properties` file in `src/test/resources/org/appfuse/tutorial/dao`, this file's properties will be available via an `"rb"` variable.

```
package org.appfuse.tutorial.dao;

import org.appfuse.dao.BaseDaoTestCase;
import org.appfuse.tutorial.model.Person;
import org.junit.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.dao.DataAccessException;
import org.springframework.test.annotation.ExpectedException;

import java.util.List;

import static org.junit.Assert.*;

public class PersonDaoTest extends BaseDaoTestCase {
    @Autowired
    private PersonDao personDao;
}
```

The code you see above is what you need for a basic Spring integration test that initializes and configures an implementation of `PersonDao`. Spring will use autowiring to set the "personDao" bean as a dependency of this class.

Now you need test that the finder method works in your DAO. AppFuse uses JUnit 4, which allows you to indicate test methods with a `@Test` annotation. Add the following method to your `PersonDaoTest.java` file:

```
@Test
public void testFindPersonByLastName() throws Exception {
    List<Person> people = personDao.findByLastName("Raible");
    assertTrue(people.size() > 0);
}
```

You'll notice that this method relies on pre-existing data in order to pass. The [DbUnit Maven Plugin](#) is used to populate the database with test data before the tests are run, so you can simply add the new table/record to the `src/test/resources/sample-data.xml` file (or `core/src/test/resources/sample-data.xml` for a modular archetype).

```
<table name='person'>
  <column>id</column>
  <column>first_name</column>
  <column>last_name</column>
  <row>
    <value>1</value>
    <value>Matt</value>
    <value>Raible</value>
  </row>
</table>
```



If you reformat the XML in sample-data.xml, make sure no line breaks are added inside the tag values. I was surprised when `<password>xxxxxxx</password>` was broken into three lines. Since the password was no longer decryptable with extra tabs and newline characters, I could no longer log in under any name. This is a feature of DBUnit that allows adding any characters to the database content.

Since the PersonDao you're about to write includes CRUD functionality, you can also write a test to verify CRUD works properly.

```
@Test
@ExpectedException(DataAccessException.class)
public void testAddAndRemovePerson() throws Exception {
    Person person = new Person();
    person.setFirstName("Country");
    person.setLastName("Bry");

    person = personDao.save(person);
    flush();

    person = personDao.get(person.getId());

    assertEquals("Country", person.getFirstName());
    assertNotNull(person.getId());

    log.debug("removing person...");

    personDao.remove(person.getId());
    flush();

    // should throw DataAccessException
    personDao.get(person.getId());
}
```

In the above example, you can see that `person.set*(value)` is being called to populate the Person object before saving it. This is easy in this example, but it could get quite cumbersome if you're persisting an object with 10 required fields. This is why a `ResourceBundle` exists in `BaseDaoTestCase`. Simply create a `PersonDaoTest.properties` file in the same directory as `PersonDaoTest.java` and define your property values inside it:

```
firstName=Matt
lastName=Raible
```



I tend to just hard-code test values into Java code - but the `.properties` file is an option that works great for large objects.

Then, rather than calling `person.set*` to populate your objects, you can use the `BaseDaoTestCase.populate(java.lang.Object)` method:

```
Person person = new Person();
person = (Person) populate(person);
```

At this point, the `PersonDaoTest` class won't compile yet because there is no `PersonDao.class` in your classpath, you need to create it.

## Create a DAO Interface and implementation

Create a `PersonDao.java` interface in the `src/main/java/**/dao` (or `core/src/main/java/**/dao` for a modular archetype) directory and specify the finder method for any implementation classes.

```
package org.appfuse.tutorial.dao;

import org.appfuse.dao.GenericDao;
import org.appfuse.tutorial.model.Person;

import java.util.List;

public interface PersonDao extends GenericDao<Person, Long> {
    public List<Person> findByLastName(String lastName);
}
```

Notice in the class above there is no exception on the method signature. This is due to the power of [Spring](#) and how it wraps Exceptions with `RuntimeExceptions`. At this point, you should be able to compile all your code using your IDE or `mvn test-compile`. However, if you try to run `mvn test -Dtest=PersonDaoTest`, you will get an error:

```
Running org.appfuse.tutorial.dao.PersonDaoTest
ERROR [main] TestContextManager.prepareTestInstance(324) | Caught exception while allowing
TestExecutionListener
[org.springframework.test.context.support.DependencyInjectionTestExecutionListener@eac619] to prepare
test instance
[org.appfuse.tutorial.dao.PersonDaoTest@73cb2c]
org.springframework.beans.factory.BeanCreationException: Error creating bean with name
'org.appfuse.tutorial.dao.PersonDaoTest': Injection of autowired
dependencies failed; nested exception is org.springframework.beans.factory.BeanCreationException:
Could not autowire field: private
org.appfuse.tutorial.dao.PersonDao org.appfuse.tutorial.dao.PersonDaoTest.personDao; nested exception
is
org.springframework.beans.factory.NoSuchBeanDefinitionException: No matching bean of type
[org.appfuse.tutorial.dao.PersonDao] found for dependency: expected at
least 1 bean which qualifies as autowire candidate for this dependency. Dependency annotations:
{@org.springframework.beans.factory.annotation.Autowired(required=true)}
```



### Showing errors in your console

To show testing errors in your console, append `-Dsurefire.useFile=false` to your `mvn test` command.

This is an error message from Spring - indicating that you need to specify a bean named "personDao". To do that you need to create the `PersonDao` implementation.

Create a `PersonDaoHibernate` class that implements the finder method in `PersonDao`. To do this, create a new class in `src/main/java/**/dao/hibernate` (or `core/src/main/java/**/dao/hibernate` for the modular archetype) and name it `PersonDaoHibernate.java`. It should extend `GenericDaoHibernate` and implement `PersonDao`. *Javadocs eliminated for brevity.*



```

package org.appfuse.tutorial.dao.hibernate;

import java.util.List;

import org.appfuse.dao.hibernate.GenericDaoHibernate;
import org.appfuse.tutorial.model.Person;
import org.appfuse.tutorial.dao.PersonDao;

import org.springframework.stereotype.Repository;

@Repository("personDao")
public class PersonDaoHibernate extends GenericDaoHibernate<Person, Long> implements PersonDao {

    public PersonDaoHibernate() {
        super(Person.class);
    }

    public List<Person> findByLastName(String lastName) {
        return getHibernateTemplate().find("from Person where lastName=?", lastName);
    }
}

```

Now, if you try to run **mvn test -Dtest=PersonDaoTest**, it should pass. The `@Repository` annotation indicates this is a Spring bean. The following XML (in your `applicationContext.xml` file) scans for classes with these annotations. If you are writing your own Dao and Dao interface, be sure to comment out or remove the reference to the generic Dao in the `applicationContext.xml` file in `src/main/webapp/WEB-INF` (or `core/src/main/resources` for a modular archetype).

```

<!-- Activates scanning of @Repository and @Service -->
<context:component-scan base-package="org.appfuse.tutorial"/>

```

If you don't like annotations, you can also use XML. To do this, you need to configure Spring so it knows that `PersonDaoHibernate` is the implementation of `PersonDao`. Open the `applicationContext.xml` file in `src/main/webapp/WEB-INF` (or `core/src/main/resources` for a modular archetype) and add the following XML to it:

```

<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

    <bean id="personDao" class="org.appfuse.tutorial.dao.hibernate.PersonDaoHibernate"/>
</beans>

```

## Run the DAO Test

Save all your edited files and try running **mvn test -Dtest=PersonDaoTest** one more time.

Yeah Baby, Yeah:  
**BUILD SUCCESSFUL**  
 Total time: 12 seconds



### Open Session In View Filter

If you want to use Hibernate's lazy-loading feature in your application, you'll need to uncomment the `lazyLoadingFilter` (and its mapping) in `web.xml`.

---

*Next Up: **Part II: Creating new Managers** - A HowTo for creating Business Facades, which are similar to **Session Facades**, but don't use EJBs. These facades are used to provide communication from the front-end to the DAO layer.*

## Java 5 Enums Persistence with Hibernate

## About this Tutorial

If you need to map Java 5 Enums to Hibernate using JPA annotations this is you are looking for.

## Table of Contents

1. [Writing the JPA annotations](#)
2. [Mapping Java 5 Enums to Hibernate](#)

## Writing the JPA annotations

First, a sample POJO from AppFuse Persistence tutorial to demonstrate how to use the annotations.

```
package org.appfuse.tutorial.model;

@Entity
@Table(name="t_person")
public class Person extends BaseObject {

    // Enumerations -----
    public enum Sex{

        MALE(1),
        FEMALE(2);

        private int value;

        Sex(int value) {
            this.value = value;
        }

        // the identifierMethod
        public int toInt() {
            return value;
        }

        // the valueOfMethod
        public static Sex fromInt(int value) {
            switch(value) {
                case 1: return MALE;
                case 2: return FEMALE;
                default:
                    return UNKNOWN;
            }
        }

        public String toString() {
            switch(this) {
                case MALE:
                    return "Male";
                case FEMALE:
                    return "Female";
            }
        }
    }

    // Attributes -----

    @Id
    @Column(name= "person_id")
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;

    @Column(name="person_firstname", length = 50, nullable = false)
```

```
private String firstName;

@Column(name="person_lastname", length = 50, nullable = false)
private String lastName;

@Column(name= "person_sex", columnDefinition="integer", nullable = true)
@Type(
    type = "org.appfuse.tutorial.common.hibernate.GenericEnumUserType",
    parameters = {
        @Parameter(
            name = "enumClass",
            value = "org.appfuse.tutorial.model.Person$Sex"),
        @Parameter(
            name = "identifierMethod",
            value = "toInt"),
        @Parameter(
            name = "valueOfMethod",
            value = "fromInt")
    }
)
private Sex sex;

/*
 * Getters and Setters ...
 */
```

```
    */  
}
```

## Mapping Java 5 Enums to Hibernate

Now, use the following **Flexible solution - working version** to map the Sex attribute. Thanks to Martin Kersten, from the article [Java 5 EnumUserType](#).

```
package org.appfuse.tutorial.common.hibernate;  
  
import java.io.Serializable;  
import java.lang.reflect.Method;  
import java.sql.PreparedStatement;  
import java.sql.ResultSet;  
import java.sql.SQLException;  
import java.util.Properties;  
  
import org.hibernate.HibernateException;  
import org.hibernate.type.NullableType;  
import org.hibernate.type.TypeFactory;  
import org.hibernate.usertype.ParameterizedType;  
import org.hibernate.usertype.UserType;  
  
public class GenericEnumUserType implements UserType, ParameterizedType {  
    private static final String DEFAULT_IDENTIFIER_METHOD_NAME = "name";  
    private static final String DEFAULT_VALUE_OF_METHOD_NAME = "valueOf";  
  
    private Class<? extends Enum> enumClass;  
    private Class<?> identifierType;  
    private Method identifierMethod;  
    private Method valueOfMethod;  
    private NullableType type;  
    private int[] sqlTypes;  
  
    public void setParameterValues(Properties parameters) {  
        String enumClassName = parameters.getProperty("enumClass");  
        try {  
            enumClass = Class.forName(enumClassName).asSubclass(Enum.class);  
        } catch (ClassNotFoundException cfne) {  
            throw new HibernateException("Enum class not found", cfne);  
        }  
  
        String identifierMethodName = parameters.getProperty("identifierMethod",  
            DEFAULT_IDENTIFIER_METHOD_NAME);  
  
        try {  
            identifierMethod = enumClass.getMethod(identifierMethodName, new Class[0]);  
            identifierType = identifierMethod.getReturnType();  
        } catch (Exception e) {  
            throw new HibernateException("Failed to obtain identifier method", e);  
        }  
  
        type = (NullableType) TypeFactory.basic(identifierType.getName());  
  
        if (type == null)  
            throw new HibernateException("Unsupported identifier type " + identifierType.getName());  
  
        sqlTypes = new int[] { type.sqlType() };  
  
        String valueOfMethodName = parameters.getProperty("valueOfMethod",  
            DEFAULT_VALUE_OF_METHOD_NAME);  
  
        try {  
            valueOfMethod = enumClass.getMethod(valueOfMethodName, new Class[] { identifierType });  
        }
```

```

        } catch (Exception e) {
            throw new HibernateException("Failed to obtain valueOf method", e);
        }
    }

    public Class returnedClass() {
        return enumClass;
    }

    public Object nullSafeGet(ResultSet rs, String[] names, Object owner) throws HibernateException,
        SQLException {
        Object identifier = type.get(rs, names[0]);
        if (identifier == null) {
            return null;
        }

        try {
            return valueOfMethod.invoke(enumClass, new Object[] { identifier });
        } catch (Exception e) {
            throw new HibernateException("Exception while invoking valueOf method '" +
                valueOfMethod.getName() + "' of " +
                    "enumeration class '" + enumClass + "'", e);
        }
    }

    public void nullSafeSet(PreparedStatement st, Object value, int index) throws HibernateException,
        SQLException {
        try {
            if (value == null) {
                st.setNull(index, type.sqlType());
            } else {
                Object identifier = identifierMethod.invoke(value, new Object[0]);
                type.set(st, identifier, index);
            }
        } catch (Exception e) {
            throw new HibernateException("Exception while invoking identifierMethod '" +
                identifierMethod.getName() + "' of " +
                    "enumeration class '" + enumClass + "'", e);
        }
    }

    public int[] sqlTypes() {
        return sqlTypes;
    }

    public Object assemble(Serializable cached, Object owner) throws HibernateException {
        return cached;
    }

    public Object deepCopy(Object value) throws HibernateException {
        return value;
    }

    public Serializable disassemble(Object value) throws HibernateException {
        return (Serializable) value;
    }

    public boolean equals(Object x, Object y) throws HibernateException {
        return x == y;
    }

    public int hashCode(Object x) throws HibernateException {
        return x.hashCode();
    }

    public boolean isMutable() {
        return false;
    }

    public Object replace(Object original, Object target, Object owner) throws HibernateException {

```

```
return original;
```

```
}  
}
```

## Managing Hibernate through MBeans

### Managing Hibernate through MBeans

Two files need to be modified:

- `src/main/resources/hibernate.cfg.xml` where the following line should be added in the session-factory tag:

hibernate.cfg.xml
<pre>&lt;property name="hibernate.generate_statistics"&gt;true&lt;/property&gt;</pre>

- `src/main/resources/applicationContext-dao.xml`, just before the comment "If you want to be able to do simple CRUD...":

applicationContext-dao.xml
<pre>&lt;bean id="jmxExporter"   class="org.springframework.jmx.export.MBeanExporter"&gt;   &lt;property name="beans"&gt;     &lt;map&gt;       &lt;entry key="Hibernate:name=statistics"&gt;         &lt;ref local="statisticsBean" /&gt;       &lt;/entry&gt;     &lt;/map&gt;   &lt;/property&gt; &lt;/bean&gt;  &lt;bean id="statisticsBean" class="org.hibernate.jmx.StatisticsService"&gt;   &lt;property name="statisticsEnabled"&gt;     &lt;value&gt;true&lt;/value&gt;   &lt;/property&gt;   &lt;property name="sessionFactory" ref="sessionFactory" /&gt; &lt;/bean&gt;</pre>

Once the package recreated (`mvn package` or `mvn jetty:run-war`) and the server restarted, a Hibernate entry appears among the other MBeans...

For Tomcat, the following options should be provided to the JVM (the port of course may be changed):

```
-Dcom.sun.management.jmxremote.port=9002  
-Dcom.sun.management.jmxremote.authenticate=false -Dcom.sun.management.jmxremote.ssl=false
```

## Using iBATIS

### About this Tutorial

This tutorial will show you two things:

1. You don't need to write DAOs if you just need generic CRUD functionality.
2. How to write DAOs when you need custom functionality.

If you're new to iBATIS, you might want to read the [iBATIS Reference Guide](#) before starting this tutorial.

### Table of Contents

1. Setup your project to use iBATIS
2. Register a personDao bean definition

3. [Create a DAO Test to test finder functionality](#)
4. [Create a DAO Interface and implementation](#)
5. [Run the DAO Test](#)

## Setup your project to use iBATIS

To change a newly-created project to use iBATIS instead of Hibernate (the default), you'll need to perform the following steps:

1. Change the `<dao.framework>` property in your `pom.xml` to be **ibatis** instead of **hibernate**.
2. The `*SQL.xml` mapping files are currently MySQL-only, so if you want to use them with another database, you'll want to download them into your project and put them in a `src/main/resources/sqlmaps` directory. You can right-click, save as the following links: [LookupSQL.xml](#), [RoleSQL.xml](#) and [UserSQL.xml](#).
3. At this point, if you use JPA annotations on your POJOs, you can use the Hibernate3 Maven Plugin to generate your schema. To do this, change `<implementation>annotationconfiguration</implementation>` to `<implementation>jpaconfiguration</implementation>` in your `pom.xml`. If you'd like more control of your schema, see the instructions below.
4. Delete `src/main/resources/hibernate.cfg.xml`. If you're using the `sql-maven-plugin` (step #2 below), you can delete the `src/main/resources/META-INF` directory from your project as well. If you want to use the [AppFuse Maven Plugin](#) to generate code, you will need to keep `hibernate.cfg.xml` in your project (at least for the M5 release).

## Using SQL Maven Plugin to Generate Schema

1. Download the schema creation DDL and put it in `src/test/resources`: [MySQL](#) or [PostgreSQL](#).
2. Remove the `hibernate3-maven-plugin` from your `pom.xml` and replace it with the `sql-maven-plugin`:

```
<plugin>
  <groupId>org.codehaus.mojo</groupId>
  <artifactId>sql-maven-plugin</artifactId>
  <version>1.4</version>
  <configuration>
    <driver>${jdbc.driverClassName}</driver>
    <username>${jdbc.username}</username>
    <password>${jdbc.password}</password>
    <url>${jdbc.url}</url>
    <autocommit>true</autocommit>
    <skip>${skipTests}</skip>
  </configuration>
  <executions>
    <execution>
      <id>create-schema</id>
      <phase>process-test-resources</phase>
      <goals>
        <goal>execute</goal>
      </goals>
      <configuration>
        <autocommit>true</autocommit>
        <srcFiles>
          <srcFile>src/test/resources/${jdbc.groupId}-schema.sql</srcFile>
        </srcFiles>
      </configuration>
    </execution>
  </executions>
  <dependencies>
    <dependency>
      <groupId>${jdbc.groupId}</groupId>
      <artifactId>${jdbc.artifactId}</artifactId>
      <version>${jdbc.version}</version>
    </dependency>
  </dependencies>
</plugin>
```





### Source Code

The code for this tutorial is located in the "tutorial-ibatis" module of the [appfuse-demos](#) project on Google Code. Use the following command to check it out from Subversion:

```
svn checkout http://appfuse-demos.googlecode.com/svn/trunk/tutorial-ibatis
```

## Register a personDao bean definition

AppFuse 2.x doesn't require you to write a DAO to persist a POJO. You can use the [GenericDaoiBatis](#) class if all you need is CRUD on an object:

To register a personDao bean, open *src/main/webapp/WEB-INF/applicationContext.xml* (or *core/src/main/resources/applicationContext.xml* for a modular archetype) and add the following to it:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

    <bean id="personDao" class="org.appfuse.dao.ibatis.GenericDaoiBatis">
        <constructor-arg value="org.appfuse.tutorial.model.Person"/>
        <property name="dataSource" ref="dataSource"/>
        <property name="sqlMapClient" ref="sqlMapClient"/>
    </bean>
</beans>
```

You will also need to create a *PersonSQL.xml* file in *src/main/resources/sqlmaps*:

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE sqlMap PUBLIC "-//ibatis.com//DTD SQL Map 2.0//EN"
"http://ibatis.apache.org/dtd/sql-map-2.dtd">

<sqlMap namespace="PersonSQL">
  <typeAlias alias="person" type="org.appfuse.tutorial.model.Person"/>

  <parameterMap id="addParam" class="person">
    <parameter property="firstName" jdbcType="VARCHAR" javaType="java.lang.String"/>
    <parameter property="lastName" jdbcType="VARCHAR" javaType="java.lang.String"/>
  </parameterMap>

  <parameterMap id="updateParam" class="person">
    <parameter property="id" jdbcType="INTEGER" javaType="java.lang.Long"/>
    <parameter property="firstName" jdbcType="VARCHAR" javaType="java.lang.String"/>
    <parameter property="lastName" jdbcType="VARCHAR" javaType="java.lang.String"/>
  </parameterMap>

  <resultMap id="personResult" class="person">
    <result property="id" column="id"/>
    <result property="firstName" column="first_name"/>
    <result property="lastName" column="last_name"/>
  </resultMap>

  <select id="getPersons" resultMap="personResult">
    <![CDATA[
      select * from person
    ]]>
  </select>

  <select id="getPerson" resultMap="personResult">
    <![CDATA[
      select * from person where id = #value#
    ]]>
  </select>

  <insert id="addPerson" parameterMap="addParam">
    <![CDATA[
      insert into person (first_name,last_name) values ( ?,? )
    ]]>
    <selectKey resultClass="java.lang.Long" keyProperty="id" type="post">
      SELECT LAST_INSERT_ID() AS id
    </selectKey>
  </insert>

  <update id="updatePerson" parameterMap="updateParam">
    <![CDATA[
      update person set first_name = ?, last_name = ? where id = ?
    ]]>
  </update>

  <delete id="deletePerson">
    <![CDATA[
      delete from person where id = #value#
    ]]>
  </delete>
</sqlMap>

```

Next, add a reference to the new SQL Map in *src/main/resources/sql-map-config.xml*:

```

<sqlMap resource="sqlmaps/PersonSQL.xml"/>

```

If you're using the **sql-maven-plugin**, you'll also need to add a "create table" statement to *src/test/resources/mysql-schema.sql*:

```
drop table if exists person;

CREATE TABLE person (
    id int(8) auto_increment,
    first_name varchar(50) NOT NULL,
    last_name varchar(50) NOT NULL,
    PRIMARY KEY (id)
);
```

After doing this, you can use this bean on an object by adding the following setter method:

```
public void setPersonDao(GenericDao<Person, Long> personDao) {
    this.personDao = personDao;
}
```

If you need more than just CRUD functionality, you'll want to continue reading below. If not, you can continue to [Creating new Managers](#). This is a tutorial for creating Business Facades, which are similar to [Session Facades](#), but don't use EJBs. These facades are used to provide communication from the front-end to the DAO layer.

### Create a DAO Test to test finder functionality

Now you'll create a `DaoTest` to test that your DAO works. "Wait a minute," you say, "I haven't created a DAO!" You are correct. However, I've found that [Test-Driven Development](#) breeds higher quality software. For years, I thought **write your test before your class** was hogwash. It just seemed stupid. Then I tried it and I found that it works great. The only reason I do test-driven stuff now is because I've found it rapidly speeds up the process of software development.

To start, create a `PersonDaoTest.java` class in your `src/test/java/**/dao` directory (or `core/src/test/java/**/dao` directory for a modular archetype). This class should extend `org.appfuse.dao.BaseDaoTestCase`, a subclass of Spring's [AbstractTransactionalJUnit4SpringContextTests](#). This parent class is used to load Spring's `ApplicationContext` (since Spring binds interfaces to implementations), and for (optionally) loading a `.properties` file that has the same name as your `*Test.class`. In this example, if you put a `PersonDaoTest.properties` file in `src/test/resources/org/appfuse/tutorial/dao`, this file's properties will be available via an `"rb"` variable.

```
package org.appfuse.tutorial.dao;

import org.appfuse.dao.BaseDaoTestCase;
import org.appfuse.tutorial.model.Person;
import org.junit.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.dao.DataAccessException;
import org.springframework.test.annotation.ExpectedException;

import java.util.List;

import static org.junit.Assert.*;

public class PersonDaoTest extends BaseDaoTestCase {
    @Autowired
    private PersonDao personDao;
}
```

The code you see above is what you need for a basic Spring integration test that initializes and configures an implementation of `PersonDao`. Spring will use autowiring to set the `"personDao"` bean as a dependency of this class.

Now you need test that the finder method works in your DAO. AppFuse uses JUnit 4, which allows you to indicate test methods with a `@Test` annotation. Add the following method to your `PersonDaoTest.java` file:

```
@Test
public void testFindPersonByLastName() throws Exception {
    List<Person> people = personDao.findByLastName("Raible");
    assertTrue(people.size() > 0);
}
```

You'll notice that this method relies on pre-existing data in order to pass. The [DbUnit Maven Plugin](#) is used to populate the database with test data before the tests are run, so you can simply add the new table/record to the `src/test/resources/sample-data.xml` file (or `core/src/test/resources/sample-data.xml` for a modular archetype).

```
<table name='person'>
  <column>id</column>
  <column>first_name</column>
  <column>last_name</column>
  <row>
    <value>1</value>
    <value>Matt</value>
    <value>Raible</value>
  </row>
</table>
```

Since the `PersonDao` you're about to write includes CRUD functionality, you can also write a test to verify CRUD works properly.

```
@Test
@ExpectedException(DataAccessException.class)
public void testAddAndRemovePerson() throws Exception {
    Person person = new Person();
    person.setFirstName("Country");
    person.setLastName("Bry");

    person = personDao.save(person);

    person = personDao.get(person.getId());

    assertEquals("Country", person.getFirstName());
    assertNotNull(person.getId());

    log.debug("removing person...");

    personDao.remove(person.getId());

    // should throw DataAccessException
    personDao.get(person.getId());
}
```

In the above example, you can see that `person.set*(value)` is being called to populate the `Person` object before saving it. This is easy in this example, but it could get quite cumbersome if you're persisting an object with 10 required fields. This is why a `ResourceBundle` exists in `BaseDaoTestCase`. Simply create a `PersonDaoTest.properties` file in the same directory as `PersonDaoTest.java` and define your property values inside it:

```
firstName=Matt
lastName=Raible
```



I tend to just hard-code test values into Java code - but the `.properties` file is an option that works great for large objects.

Then, rather than calling `person.set*` to populate your objects, you can use the `BaseDaoTestCase.populate(java.lang.Object)` method:

```
Person person = new Person();
person = (Person) populate(person);
```

At this point, the `PersonDaoTest` class won't compile yet because there is no `PersonDao.class` in your classpath, you need to create it.

## Create a DAO Interface and implementation

Create a `PersonDao.java` interface in the `src/main/java/**/dao` (or `core/src/main/java/**/dao` for a modular archetype) directory and specify the finder method for any implementation classes.

```
package org.appfuse.tutorial.dao;

import org.appfuse.dao.GenericDao;
import org.appfuse.tutorial.model.Person;

import java.util.List;

public interface PersonDao extends GenericDao<Person, Long> {
    public List<Person> findByLastName(String lastName);
}
```

Notice in the class above there is no exception on the method signature. This is due to the power of [Spring](#) and how it wraps Exceptions with `RuntimeExceptions`. At this point, you should be able to compile all your code using your IDE or `mvn test-compile`. However, if you try to run `mvn test -Dtest=PersonDaoTest`, you will get an error:

```
Running org.appfuse.tutorial.dao.PersonDaoTest
ERROR [main] TestContextManager.prepareTestInstance(324) | Caught exception while allowing
TestExecutionListener
[org.springframework.test.context.support.DependencyInjectionTestExecutionListener@5332ca] to prepare
test instance
[org.appfuse.tutorial.dao.PersonDaoTest@2b7632]
org.springframework.beans.factory.BeanCreationException: Error creating bean with name
'org.appfuse.tutorial.dao.PersonDaoTest': Injection of autowired
dependencies failed; nested exception is org.springframework.beans.factory.BeanCreationException:
Could not autowire field: private
org.appfuse.tutorial.dao.PersonDao org.appfuse.tutorial.dao.PersonDaoTest.personDao; nested exception
is
org.springframework.beans.factory.NoSuchBeanDefinitionException: No matching bean of type
[org.appfuse.tutorial.dao.PersonDao] found for dependency: expected at
least 1 bean which qualifies as autowire candidate for this dependency. Dependency annotations:
{@org.springframework.beans.factory.annotation.Autowired(required=true)}
```



### Showing errors in your console

To show testing errors in your console, append `-Dsurefire.useFile=false` to your `mvn test` command.

This is an error message from Spring - indicating that you need to specify a bean named "personDao". To do that you need to create the `PersonDao` implementation.

Create a `PersonDaoiBatis` class that implements the finder method in `PersonDao`. To do this, create a new class in `src/main/java/**/dao/ibatis` (or `core/src/main/java/**/dao/ibatis` for the modular archetype) and name it `PersonDaoiBatis.java`. It should extend `GenericDaoiBatis` and implement `PersonDao`. *Javadocs eliminated for brevity.*

```

package org.appfuse.tutorial.dao.ibatis;

import org.appfuse.tutorial.dao.PersonDao;
import org.appfuse.tutorial.model.Person;
import org.appfuse.dao.ibatis.GenericDaoiBatis;

import java.util.List;

import org.springframework.stereotype.Repository;

@Repository("personDao")
public class PersonDaoiBatis extends GenericDaoiBatis<Person, Long> implements PersonDao {

    public PersonDaoiBatis() {
        super(Person.class);
    }

    @SuppressWarnings("unchecked")
    public List<Person> findByLastName(String lastName) {
        return (List<Person>) getSqlMapClientTemplate().queryForList("findByLastName", lastName);
    }
}

```

You'll also need to add the **findByLastName** query to your `PersonSQL.xml`:

```

<select id="findByLastName" resultMap="personResult">
    select * from person where last_name = #value#
</select>

```

Now, if you try to run **mvn test -Dtest=PersonDaoTest**, it should pass. The `@Repository` annotation indicates this is a Spring bean. The following XML (in your `applicationContext.xml` file) scans for classes with these annotations. If you are writing your own Dao and Dao interface, be sure to comment out or remove the reference to the generic Dao in the `applicationContext.xml` file in `src/main/webapp/WEB-INF` (or `core/src/main/resources` for a modular archetype).

```

<!-- Activates scanning of @Repository and @Service -->
<context:component-scan base-package="org.appfuse.tutorial"/>

```

If you don't like annotations, you can also use XML. To do this, you need to configure Spring so it knows that `PersonDaoHibernate` is the implementation of `PersonDao`. Open the `applicationContext.xml` file in `src/main/webapp/WEB-INF` (or `core/src/main/resources` for a modular archetype) and add the following XML to it:

```

<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

    <bean id="personDao" class="org.appfuse.tutorial.dao.hibernate.PersonDaoHibernate"/>
</beans>

```

## Run the DAO Test

Save all your edited files and try running **mvn test -Dtest=PersonDaoTest** one more time.

**Yeah Baby, Yeah:**  
**BUILD SUCCESSFUL**  
 Total time: 10 seconds

\*Incidentally, you can try running the tests from your IDE. Make sure you re-import your project files after changing the pom dependencies.

---

*Next Up: **Part II: Creating new Managers*** - A HowTo for creating Business Facades, which are similar to [Session Facades](#), but don't use EJBs. These facades are used to provide communication from the front-end to the DAO layer.

## Using JPA

### About this Tutorial

This tutorial will show you two things:

1. You don't need to write DAOs if you just need generic CRUD functionality.
2. How to write DAOs when you need custom functionality.

If you're new to JPA, you might want to read the [JPA Reference Guide](#) before starting this tutorial.

### Table of Contents

1. [Setup your project to use JPA](#)
2. [Register a personDao bean definition](#)
3. [Create a DAO Test to test finder functionality](#)
4. [Create a DAO Interface and implementation](#)
5. [Run the DAO Test](#)



#### Source Code

The code for this tutorial is located in the "tutorial-jpa" module of the [appfuse-demos](#) project on Google Code. Use the following command to check it out from Subversion:

```
svn checkout http://appfuse-demos.googlecode.com/svn/trunk/tutorial-jpa
```

### Setup your project to use JPA

To change a newly-created project to use JPA instead of Hibernate (the default), you'll need to perform the following steps:

1. Change the `<dao.framework>` property in your `pom.xml` (in a modular project, the top level `pom.xml`) to be **jpa** instead of **hibernate**.
2. Modify the configuration for the Hibernate3 Maven Plugin: change `<implementation>annotationconfiguration</implementation>` to `<implementation>jpaconfiguration</implementation>` in your `pom.xml` (in a modular project, `core/pom.xml`).
3. Delete the `hibernate.cfg.xml` file in `src/main/resources` (in a modular project, `core/src/main/resources/hibernate.cfg.xml`).

### Register a personDao bean definition

AppFuse 2.x doesn't require you to write a DAO to persist a POJO. You can use the [GenericDaoJpa](#) class if all you need is CRUD on an object:

To register a personDao bean, open `src/main/webapp/WEB-INF/applicationContext.xml` (or `core/src/main/resources/applicationContext.xml` for a modular archetype) and add the following to it:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

    <bean id="personDao" class="org.appfuse.dao.jpa.GenericDaoJpa">
        <constructor-arg value="org.appfuse.tutorial.model.Person"/>
    </bean>
</beans>
```

After doing this, you can use this bean on an object by adding the following setter method:

```

public void setPersonDao(GenericDao<Person, Long> personDao) {
    this.personDao = personDao;
}

```

If you need more than just CRUD functionality, you'll want to continue reading below. If not, you can continue to [Creating new Managers](#). This is a tutorial for creating Business Facades, which are similar to [Session Facades](#), but don't use EJBs. These facades are used to provide communication from the front-end to the DAO layer.

## Create a DAO Test to test finder functionality

Now you'll create a `DaoTest` to test that your DAO works. "Wait a minute," you say, "I haven't created a DAO!" You are correct. However, I've found that [Test-Driven Development](#) breeds higher quality software. For years, I thought **write your test before your class** was hogwash. It just seemed stupid. Then I tried it and I found that it works great. The only reason I do test-driven stuff now is because I've found it rapidly speeds up the process of software development.

To start, create a `PersonDaoTest.java` class in your `src/test/java/**/dao` directory (or `core/src/test/java/**/dao` directory for a modular archetype). This class should extend `org.appfuse.dao.BaseDaoTestCase`, a subclass of Spring's [AbstractTransactionalJUnit4SpringContextTests](#). This parent class is used to load Spring's `ApplicationContext` (since Spring binds interfaces to implementations), and for (optionally) loading a `.properties` file that has the same name as your `*Test.class`. In this example, if you put a `PersonDaoTest.properties` file in `src/test/resources/org/appfuse/tutorial/dao`, this file's properties will be available via an `"rb"` variable.

```

package org.appfuse.tutorial.dao;

import org.appfuse.dao.BaseDaoTestCase;
import org.appfuse.tutorial.model.Person;
import org.junit.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.dao.DataAccessException;
import org.springframework.test.annotation.ExpectedException;

import java.util.List;

import static org.junit.Assert.*;

public class PersonDaoTest extends BaseDaoTestCase {
    @Autowired
    private PersonDao personDao;
}

```

The code you see above is what you need for a basic Spring integration test that initializes and configures an implementation of `PersonDao`. Spring will use autowiring to set the `"personDao"` bean as a dependency of this class.

Now you need test that the finder method works in your DAO. AppFuse uses JUnit 4, which allows you to indicate test methods with a `@Test` annotation. Add the following method to your `PersonDaoTest.java` file:

```

@Test
public void testFindPersonByLastName() throws Exception {
    List<Person> people = personDao.findByLastName("Raible");
    assertTrue(people.size() > 0);
}

```

You'll notice that this method relies on pre-existing data in order to pass. The [DbUnit Maven Plugin](#) is used to populate the database with test data before the tests are run, so you can simply add the new table/record to the `src/test/resources/sample-data.xml` file (or `core/src/test/resources/sample-data.xml` for a modular archetype).



```
<table name='person'>
  <column>id</column>
  <column>first_name</column>
  <column>last_name</column>
  <row>
    <value>1</value>
    <value>Matt</value>
    <value>Raible</value>
  </row>
</table>
```

Since the `PersonDao` you're about to write includes CRUD functionality, you can also write a test to verify CRUD works properly.

```
@Test
@ExpectedException(DataAccessException.class)
public void testAddAndRemovePerson() throws Exception {
    Person person = new Person();
    person.setFirstName("John");
    person.setLastName("Elway");

    person = personDao.save(person);

    person = personDao.get(person.getId());

    assertEquals("John", person.getFirstName());
    assertNotNull(person.getId());

    log.debug("removing person...");

    personDao.remove(person.getId());

    // should throw DataAccessException
    personDao.get(person.getId());
}
```

In the above example, you can see that `person.set*(value)` is being called to populate the `Person` object before saving it. This is easy in this example, but it could get quite cumbersome if you're persisting an object with 10 required fields. This is why a `ResourceBundle` exists in `BaseDaoTestCase`. Simply create a `PersonDaoTest.properties` file in the same directory as `PersonDaoTest.java` and define your property values inside it:

```
firstName=Matt
lastName=Raible
```



I tend to just hard-code test values into Java code - but the `.properties` file is an option that works great for large objects.

Then, rather than calling `person.set*` to populate your objects, you can use the `BaseDaoTestCase.populate(java.lang.Object)` method:

```
Person person = new Person();
person = (Person) populate(person);
```

At this point, the `PersonDaoTest` class won't compile yet because there is no `PersonDao.class` in your classpath, you need to create it.

## Create a DAO Interface and implementation

Create a `PersonDao.java` interface in the `src/main/java/**/dao` (or `core/src/main/java/**/dao` for a modular archetype) directory and specify the finder method for any implementation classes.

```

package org.appfuse.tutorial.dao;

import org.appfuse.dao.GenericDao;
import org.appfuse.tutorial.model.Person;

import java.util.List;

public interface PersonDao extends GenericDao<Person, Long> {
    public List<Person> findByLastName(String lastName);
}

```

Notice in the class above there is no exception on the method signature. This is due to the power of [Spring](#) and how it wraps Exceptions with RuntimeExceptions. At this point, you should be able to compile all your code using your IDE or **mvn test-compile**. However, if you try to run **mvn test -Dtest=PersonDaoTest**, you will get an error:

```

Running org.appfuse.tutorial.dao.PersonDaoTest
ERROR [main] TestContextManager.prepareTestInstance(324) | Caught exception while allowing
TestExecutionListener
[org.springframework.test.context.support.DependencyInjectionTestExecutionListener@5535b7] to prepare
test instance
[org.appfuse.tutorial.dao.PersonDaoTest@39b974]
org.springframework.beans.factory.BeanCreationException: Error creating bean with name
'org.appfuse.tutorial.dao.PersonDaoTest': Injection of autowired
dependencies failed; nested exception is org.springframework.beans.factory.BeanCreationException:
Could not autowire field: private
org.appfuse.tutorial.dao.PersonDao org.appfuse.tutorial.dao.PersonDaoTest.personDao; nested exception
is
org.springframework.beans.factory.NoSuchBeanDefinitionException: No matching bean of type
[org.appfuse.tutorial.dao.PersonDao] found for dependency: expected at
least 1 bean which qualifies as autowire candidate for this dependency. Dependency annotations:
{@org.springframework.beans.factory.annotation.Autowired(required=true)}

```



#### Showing errors in your console

To show testing errors in your console, append **-Dsurefire.useFile=false** to your **mvn test** command.

This is an error message from Spring - indicating that you need to specify a bean named "personDao". To do that you need to create the PersonDao implementation.

Create a PersonDaoJpa class that implements the finder method in PersonDao. To do this, create a new class in *src/main/java/\*\*/dao/jpa* (or *\_core/src/main/java/\*\*/dao/jpa* for the modular archetype) and name it PersonDaoJpa.java. It should extend GenericDaoJpa and implement PersonDao. *Javadocs eliminated for brevity.*

```

package org.appfuse.tutorial.dao.jpa;

import org.appfuse.dao.jpa.GenericDaoJpa;
import org.appfuse.tutorial.dao.PersonDao;
import org.appfuse.tutorial.model.Person;
import org.springframework.stereotype.Repository;

import javax.persistence.Query;
import java.util.List;

@Repository("personDao")
public class PersonDaoJpa extends GenericDaoJpa<Person, Long> implements PersonDao {

    public PersonDaoJpa() {
        super(Person.class);
    }

    @SuppressWarnings("unchecked")
    public List<Person> findByLastName(String lastName) {
        Query q = getEntityManager().createQuery("select p from Person p where p.lastName=?");
        q.setParameter(1, lastName);
        return q.getResultList();
    }
}

```

Now, if you try to run **mvn test -Dtest=PersonDaoTest**, it should pass. The `@Repository` annotation indicates this is a Spring bean. The following XML (in your `applicationContext.xml` file) scans for classes with these annotations. If you are writing your own Dao and Dao interface, be sure to comment out or remove the reference to the generic Dao in the `applicationContext.xml` file in `src/main/webapp/WEB-INF` (or `core/src/main/resources` for a modular archetype).

```

<!-- Activates scanning of @Repository and @Service -->
<context:component-scan base-package="org.appfuse.tutorial"/>

```

If you don't like annotations, you can also use XML. To do this, you need to configure Spring so it knows that `PersonDaoHibernate` is the implementation of `PersonDao`. Open the `applicationContext.xml` file in `src/main/webapp/WEB-INF` (or `core/src/main/resources` for a modular archetype) and add the following XML to it:

```

<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

    <bean id="personDao" class="org.appfuse.tutorial.dao.jpa.PersonDaoJpa"/>
</beans>

```

## Run the DAO Test

Save all your edited files and try running **mvn test -Dtest=PersonDaoTest** one more time.

**Yeah Baby, Yeah:**

**BUILD SUCCESSFUL**  
Total time: 11 seconds

---

*Next Up: **Part II: Creating new Managers** - A HowTo for creating Business Facades, which are similar to [Session Facades](#), but don't use EJBs. These facades are used to provide communication from the front-end to the DAO layer.*

## Services

## About this Tutorial

This tutorial creates a Business Facade class (and a JUnit Test) to talk to the DAO you created in the [Persistence Tutorial](#).

In the context of AppFuse, this is called a Manager class. Its main responsibility is to act as a bridge between the persistence (DAO) layer and the web layer. It's also useful for de-coupling your presentation layer from your database layer (i.e. for Swing apps and web services). Managers should also be where you put any business logic for your application.

You will learn two things:

1. You don't need to write Managers if you just need generic CRUD functionality.
2. How to write Managers when you need custom functionality.

## Table of Contents

1. [Register a personManager bean definition](#)
2. [Create a Manager Interface](#)
3. [Create a Manager Test to test finder functionality](#)
4. [Create a Manager Implementation](#)
5. [Run the Manager Test](#)
6. [Register your Manager Implementation](#)
7. [Write the Manager Test using EasyMock](#)



### Source Code

The code for this tutorial is located in the "tutorial-service" module of the [appfuse-demos](#) project on Google Code. Use the following command to check it out from Subversion:

```
svn checkout http://appfuse-demos.googlecode.com/svn/trunk/tutorial-service
```

## Register a personManager bean definition

AppFuse 2.x doesn't require you to write a Manager to persist a POJO. You can use the [GenericManager](#) class if all you need is CRUD on an object. To register a personManager bean, its best to wrap the personDao bean.



### Transactions

All `service.*Manager` beans will automatically be configured by Spring to wrap PROPAGATION\_REQUIRED transactions around their method executions. This is done by the following Spring AOP configuration in `appfuse-service.jar`.

```
<aop:config>
  ...
  <aop:advisor id="managerTx" advice-ref="txAdvice"
    pointcut="execution(* *..service.*Manager.*(..))" order="2"/>
</aop:config>
```

Open your `src/main/webapp/WEB-INF/applicationContext.xml` and replace the personDao bean with the following.

**Hibernate:**

```
<bean id="personManager" class="org.appfuse.service.impl.GenericManagerImpl">
  <constructor-arg>
    <bean class="org.appfuse.dao.hibernate.GenericDaoHibernate" autowire="byType">
      <constructor-arg value="org.appfuse.tutorial.model.Person" />
    </bean>
  </constructor-arg>
</bean>
```



### iBatis and JPA Configuration

For iBatis and JPA, you'll need to change the nested DAO that gets injected into `GenericManagerImpl`:

- **iBatis:** `<bean class="org.appfuse.dao.ibatis.GenericDaoiBatis" ...`
- **JPA:** `<bean class="org.appfuse.dao.jpa.GenericDaoJpa" ...`

If you wrote the `PersonDao` interface and implementation in the previous tutorial, you'll want to use the following for your `personManager` bean definition. If you don't, your `PersonDaoTest` will fail because there's no longer an exposed `personDao` bean.

```
<bean id="personManager" class="org.appfuse.service.impl.GenericManagerImpl">
  <constructor-arg ref="personDao"/>
</bean>
```

Once you've created a `personManager` bean definition, you can use this bean on an object by adding the following setter method:

```
public void setPersonManager(GenericManager<Person, Long> personManager) {
    this.personManager = personManager;
}
```



### The Spring Framework

To learn more about how Spring works, please see the [Spring Reference Guide](#).

That's it! To persist an object with AppFuse 2.x, all you need to do is:

1. Create a POJO with annotations.
2. Register it with your persistence framework.
3. Write some XML to register a type-safe class to persist it, or write **no** XML to use the Universal\* classes.

Not only did you write data access code, you also used interfaces to abstract your implementation. This means that it should be possible to replace your persistence framework at any time.



You can also perform all 3 steps with the [AppFuse Maven Plugin](#).

If you need more than just CRUD functionality, you'll want to continue reading below. From here, you can [expose your manager as a web service](#) or continue [writing your web application](#).

## Create a Manager Interface

First off, create a `PersonManager` interface (in the `src/main/java/**/service` directory - you may need to create this first) and specify the finder method for any implementation classes.

```
package org.appfuse.tutorial.service;

import org.appfuse.service.GenericManager;
import org.appfuse.tutorial.model.Person;

import java.util.List;

public interface PersonManager extends GenericManager<Person, Long> {
    List<Person> findByLastName(String lastName);
}
```

## Create a Manager Test to test finder functionality

Now you know what you want your manager to do, so it's time to write the tests. In this example, you will use mock objects to isolate your tests

from external dependencies. This means you do not need a database set up to run these tests, and it will be easier to test different scenarios. In a large project, you could even test your Manager class before someone else finished writing the DAO implementation.

You can write your own mock objects, but here you are going to use [jMock](#). Another widely use Mock Object library is [EasyMock](#).



Mockito is quickly becoming the most popular mocking framework for Java and we hope to support it in a future release.

Create `PersonManagerImplTest` in `src/test/java/**/service/impl`.

```
package org.appfuse.tutorial.service.impl;

import org.appfuse.service.impl.BaseManagerMockTestCase;
import org.appfuse.tutorial.dao.PersonDao;
import org.appfuse.tutorial.model.Person;
import org.appfuse.tutorial.service.impl.PersonManagerImpl;
import org.jmock.Expectations;
import org.junit.After;
import org.junit.Before;
import org.junit.Test;

import java.util.ArrayList;
import java.util.List;

import static org.junit.Assert.*;

public class PersonManagerImplTest extends BaseManagerMockTestCase {
    private PersonManagerImpl manager = null;
    private PersonDao dao = null;

    @Before
    public void setUp() {
        dao = context.mock(PersonDao.class);
        manager = new PersonManagerImpl(dao);
    }

    @After
    public void tearDown() {
        manager = null;
    }

    @Test
    public void testGetPerson() {
        log.debug("testing get...");

        final Long id = 7L;
        final Person person = new Person();

        // set expected behavior on dao
        context.checking(new Expectations() {{
            one(dao).get(with(equal(id)));
            will(returnValue(person));
        }});

        Person result = manager.get(id);
        assertEquals(person, result);
    }

    @Test
    public void testGetPersons() {
        log.debug("testing getAll...");

        final List persons = new ArrayList();

        // set expected behavior on dao
        context.checking(new Expectations() {{
            one(dao).getAll();
        }});
    }
}
```

```

        will(returnValue(persons));
    });

    List result = manager.getAll();

    assertEquals(persons, result);
}

@Test
public void testSavePerson() {
    log.debug("testing save...");

    final Person person = new Person();
    // enter all required fields

    // set expected behavior on dao
    context.checking(new Expectations() {{
        one(dao).save(with(same(person)));
    }});

    manager.save(person);
}

@Test
public void testRemovePerson() {
    log.debug("testing remove...");

    final Long id = -1L;

    // set expected behavior on dao
    context.checking(new Expectations() {{
        one(dao).remove(with(equal(id)));
    }});

    manager.remove(id);
}

```

```
}  
}
```

This will not compile, as you have not created the `PersonManagerImpl` class it tests.

## Create a Manager Implementation

The next step is to create a `PersonManagerImpl` class that implements the methods in `PersonManager`.

Create `PersonManagerImpl.java` in `src/main/java/**/service/impl`.

```
package org.appfuse.tutorial.service.impl;  
  
import org.appfuse.tutorial.dao.PersonDao;  
import org.appfuse.tutorial.model.Person;  
import org.appfuse.tutorial.service.PersonManager;  
import org.appfuse.service.impl.GenericManagerImpl;  
  
import java.util.List;  
  
public class PersonManagerImpl extends GenericManagerImpl<Person, Long> implements PersonManager {  
    PersonDao personDao;  
  
    public PersonManagerImpl(PersonDao personDao) {  
        super(personDao);  
        this.personDao = personDao;  
    }  
  
    public List<Person> findByLastName(String lastName) {  
        return personDao.findByLastName(lastName);  
    }  
}
```

Now before you run your tests, review your test class to make sure that it will test all possible conditions.

## Run the Manager Test

Save all your edited files and try running `mvn test -Dtest=PersonManagerImplTest`.

## Register your Manager Implementation

There are two ways to register your `personManager` bean at this point: 1) using a `@Service` annotation or 2) with XML. To use annotations, simply add `@Service("personManager")` as a class level parameter:

```
import org.springframework.stereotype.Service;  
  
@Service("personManager")  
public class PersonManagerImpl extends GenericManagerImpl<Person, Long> implements PersonManager {
```

If you'd prefer to use XML, open your `src/main/webapp/WEB-INF/applicationContext.xml` file and replace the `personManager` bean with the following:

```
<bean id="personManager" class="org.appfuse.tutorial.service.impl.PersonManagerImpl">  
    <constructor-arg ref="personDao" />  
</bean>
```





The Web application tutorials assume that you will be using the GenericManager. If you follow them after making this change, you will need to change all the references in their code from the GenericManager to your new PersonManager interface. For example:

```
@Autowired
private PersonManager personManager;
```

That's it. If you want to see how to use EasyMock instead of JMock, then carry on reading. If not, you can [expose your manager as a web service](#) or continue to [writing your web application](#).

## Using EasyMock instead of jMock in Unit Tests

Sooner or later, you will want to add extra dependencies into your project.

AppFuse currently ships with JMock, but not [EasyMock](#), so this section will show you how to add EasyMock as a project dependency.

Edit your *pom.xml* file in your project's top level directory. Add EasyMock as a dependency in the <dependencies> element:

```
<dependencies>
    ...
    <dependency>
        <groupId>org.easymock</groupId>
        <artifactId>easymock</artifactId>
        <version>2.2</version>
        <scope>test</scope>
    </dependency>
</dependencies>
```

EasyMock is only required during testing, and you don't want it packaged in your application, so it's restricted it to a scope of *test*.



After removing jMock and adding EasyMock to your pom.xml, you'll want to regenerate project files for your IDE (**mvn install eclipse:eclipse** for Eclipse or **mvn idea:idea** for IDEA).

Edit the PersonManagerImplTest class you wrote above so it looks as follows:

```
package org.appfuse.tutorial.service.impl;

import junit.framework.TestCase;
import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;
import org.appfuse.tutorial.dao.PersonDao;
import org.appfuse.tutorial.model.Person;
import org.appfuse.tutorial.service.impl.PersonManagerImpl;

import java.util.ArrayList;
import java.util.List;

import static org.easymock.EasyMock.*;

public class PersonManagerImplTest extends TestCase {
    private final Log log = LogFactory.getLog(PersonManagerImplTest.class);
    private PersonManagerImpl manager = null;
    private PersonDao dao = null;
    private Person person = null;

    protected void setUp() throws Exception {
        log.debug("setUpDao for PersonManagerImplTest");
        dao = createStrictMock(PersonDao.class);
    }
}
```

```

        manager = new PersonManagerImpl((PersonDao) dao);
    }

    public void testGetPerson() {
        log.debug("testing getPerson");

        Long id = 7L;
        person = new Person();

        // set expected behavior on dao
        expect(dao.get(id)).andReturn(person);
        replay(dao);

        Person result = manager.get(id);
        assertSame(person, result);
        verify(dao);
    }

    public void testGetPersons() {
        log.debug("testing getPersons");

        List people = new ArrayList();

        // set expected behavior on dao
        expect(dao.getAll()).andReturn(people);
        replay(dao);

        List result = manager.getAll();
        assertSame(people, result);
        verify(dao);
    }

    public void testGetByLastName() {
        log.debug("testing getByLastName");

        List people = new ArrayList();
        String lastName = "Smith";

        // set expected behavior on dao
        expect(dao.findByLastName(lastName)).andReturn(people);
        replay(dao);

        List result = manager.findByLastName(lastName);
        assertSame(people, result);
        verify(dao);
    }

    public void testSavePerson() {
        log.debug("testing savePerson");

        person = new Person();

        // set expected behavior on dao
        expect(dao.save(person)).andReturn(person);
        replay(dao);

        manager.save(person);
        verify(dao);
    }

    public void testRemovePerson() {
        log.debug("testing removePerson");

        Long id = 11L;
        person = new Person();

        // set expected behavior on dao
        dao.remove(id);
        replay(dao);
    }

```

```
manager.remove(id);  
verify(dao);
```

```
}  
}
```

Note that this class extends `junit.framework.TestCase` and not an EasyMock class. This makes EasyMock a good choice for annotation based test frameworks such as [JUnit 4](#) and [TestNG](#). Unfortunately (at the time of writing) Maven does not work properly with these frameworks.

Now check everything works by running `mvn test -Dtest=PersonManagerImplTest` again.

From here, you can [expose your manager as a web service](#) or continue [writing your web application](#).

## Web Services

### About this Tutorial

This tutorial exposes your Manager classes (or any other classes) as a web service with [CXF](#), a proven Open-Source Services Framework. It assumes you completed the [Services](#) tutorial and wrote a custom `PersonManager` class and `PersonManagerImpl` implementation.

### Table of Contents

1. Expose your Manager as a Web Service
  - a. SOAP
  - b. REST
2. Securing your Web Service



#### Source Code

The code for this tutorial is located in the "tutorial-service" module of the [appfuse-demos](#) project on Google Code. Use the following command to check it out from Subversion:

```
svn checkout http://appfuse-demos.googlecode.com/svn/trunk/tutorial-service
```

### Expose your Manager as a Web Service

Thanks to annotations, it's very easy to expose your existing classes as web services. In fact, there's a couple services that are already exposed in your AppFuse application. For the SOAP-based `UserService`, you can see the WSDL by running `mvn jetty:run` and navigating to <http://localhost:8080/services/UserService?wsdl>. You can view the REST equivalent (a WADL file) at [http://localhost:8080/services/api?\\_wadl&\\_type=xml](http://localhost:8080/services/api?_wadl&_type=xml). All web services can be viewed at <http://localhost:8080/services/>.

### Exposing a SOAP Service

To expose your `PersonManager` (and its implementation) as a SOAP service, add a `@WebService` annotation to the interface.

```
package org.appfuse.tutorial.service;  
  
import org.appfuse.service.GenericManager;  
import org.appfuse.tutorial.model.Person;  
  
import javax.jws.WebService;  
import java.util.List;  
  
@WebService  
public interface PersonManager extends GenericManager<Person, Long> {  
    public List<Person> findByLastName(String lastName);  
}
```

Then add the same annotation to your `PersonManagerImpl`, specifying the service name and endpointInterface. You'll also need to add a public, no-args constructor to satisfy JAX-WS.

```

package org.appfuse.tutorial.service.impl;

import org.appfuse.tutorial.dao.PersonDao;
import org.appfuse.tutorial.model.Person;
import org.appfuse.tutorial.service.PersonManager;
import org.appfuse.service.impl.GenericManagerImpl;

import javax.jws.WebService;
import java.util.List;

@Service("personManager")
@WebService(serviceName = "PersonService", endpointInterface =
"org.appfuse.tutorial.service.PersonManager")
public class PersonManagerImpl extends GenericManagerImpl<Person, Long> implements PersonManager {
    PersonDao personDao;

    public PersonManagerImpl() {};

    public PersonManagerImpl(PersonDao personDao) {
        super(personDao);
        this.personDao = personDao;
    }

    public List<Person> findByLastName(String lastName) {
        return personDao.findByLastName(lastName);
    }
}

```

Open your `src/main/webapp/WEB-INF/cxf-servlet.xml` file and add the following declaration to tell CXF about your web service endpoint:

```

<jaxws:endpoint id="personService" implementor="#personManager" address="/PersonService"/>

```

After making these changes, you should be able to run **mvn jetty:run** and view the WSDL for your PersonManager at <http://localhost:8080/services/PersonService?wsdl>.

To continue creating your web application, please see the [web tutorials](#).

### Exposing a REST Service

To expose your `PersonManager` (and its implementation) as a REST service, add a `@PATH` annotation to the interface and annotations to the method to indicate its path and supported methods. NOTE: This example includes the aforementioned `@WebService` annotation because it's perfectly acceptable to use both in the same files.

```

package org.appfuse.tutorial.service;

import org.appfuse.service.GenericManager;
import org.appfuse.tutorial.model.Person;

import javax.jws.WebService;
import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.PathParam;
import java.util.List;

@WebService
@Path("/")
public interface PersonManager extends GenericManager<Person, Long> {
    @Path("/person/{lastname}")
    @GET
    List<Person> findByLastName(@PathParam("lastname") String lastName);

    @Path("/people")
    @GET
    List<Person> getPeople();
}

```

With JAX-RS, you don't need any additional annotations on your implementation. However, you'll still need to add the implementation of the `getPeople()` method and to tell CXF about it.

Add the `getPeople()` method's implementation to your `PersonManagerImpl`.

```

public List<Person> getPeople() {
    return personDao.getAll();
}

```

Next, open your `src/main/webapp/WEB-INF/cxf-servlet.xml` file and add the following XML in the `<jaxrs:serviceBeans>` section:

```

<ref bean="personManager"/>

```

After making these changes, you should be able to run `mvn jetty:run` and view all the people in your database at <http://localhost:8080/services/api/people>.



If you have trouble getting this to work, please subscribe to the [mailing list](#) and send us the issue you're experiencing.

## Security

Web Services in AppFuse are unsecure by default. To secure them, the easiest thing to do is add `/services/**=ROLE_NAME` to your `WEB-INF/security.xml` file. For more information, see Enunciate's tutorial on [Securing Web Services](#)

## Limit exposed methods

When you add the annotations to your service classes, by default all public methods are exposed as web services. This might not be ideal. If you would like to limit the methods that are exposed, then do the following.

- i. Remove the `@WebService` from the service interface class (i.e `PersonManager`)
- ii. Remove the endpoint attribute in the service implementation class (i.e `PersonManagerImpl`) so that it now look like this  
`@WebService(serviceName = "PersonService")`
- iii. Add `@WebMethod(exclude=true)` for each public method in the implemented class (i.e `PersonManagerImpl`) that you would not like exposed as a web service. This includes methods defined in parent classes as well (i.e `GenericManagerImpl`)
- iv. Add `@WebMethod` on each public method that you would like to expose as a webservice.

## Web

Web development in 2007 is a lot different than it was earlier this decade. Web applications are getting so rich that they're starting to resemble desktop applications, both with their functionality and their [ability to work offline](#). While many folks say the web browser's days are numbered, I don't buy it. The internet was built on web pages and web applications. It's not going away, and kick-ass internet applications are more than possible with today's technologies. The only limitation is your imagination.

Before you get started developing web applications, there's a few things you absolutely *must* do:

1. [Get Firefox](#). Download, install and use daily. Your application will work on IE, but it's **much** easier to develop for Firefox and tweak for IE (and Safari).
2. [Install Firebug](#). This plugin is probably one of the best things that ever happened to web development.



If you're going to be a web developer, you should spend some time learning the power of XHTML, CSS and JavaScript. You should learn the Ajax toolkit that your preferred web framework uses and/or supports. Especially if you're getting paid to develop the application. Try your best to get paid to learn. I recommend spending an hour each morning reading, when you first get to work. Books are better than websites and blogs because they focus your mind and you're not as tempted to wander.

Now comes the hard part when you're doing Java web development. Which framework do you choose?

Please choose a web framework from the list below to continue:

- [JSF](#)
- [Struts 2](#)
- [Spring MVC](#)
- [Tapestry](#)



#### Having trouble choosing a web framework?

If you're having trouble choosing a web framework, read [What Web Application framework should you use?](#) and [Java Web Framework Sweet Spots](#) (PDF). More information on comparing web frameworks can be found [here](#).

We hope to add Stripes and Wicket as supported web frameworks in the next few months.

## Using JSF

### About this Tutorial

This tutorial creates master/detail screens with [Java Server Faces](#) (JSF). The JSF implementation used in AppFuse is [MyFaces](#), but you should be able to use any 2.0+ JSF implementation. The list (master) screen will have the ability to sort columns, as well as page 25 records at a time. The form (detail) screen will use a nifty CSS form layout (courtesy of [Wufoo](#)). You will also configure client and server-side validation to improve your users' experience.



#### IntelliJ IDEA Rocks

We highly recommend using IntelliJ IDEA when developing web applications in Java. Not only is its Java and JUnit support fantastic, but it has excellent CSS and JavaScript support. [Using JRebel with IDEA](#) is likely to provide you with the most pleasant Java development experiences you've ever had.



This tutorial assumes you've created a project with the **appfuse-basic-jsf** archetype and have already completed the [Persistence](#) and [Services](#) tutorials. If you're using the **appfuse-modular-jsf** archetype, please morph your mind into using the *web* module as the root directory. If you created your project with a different web framework than JSF, you're likely to be confused and nothing will work in this tutorial. 😊

### Table of Contents

1. [Introduction to JSF](#)
2. [Create a PersonListTest](#)
3. [Create a PersonList class that will fetch people](#)
4. [Create persons.xhtml to show search results](#)
5. [Create a PersonFormTest and PersonForm for edit\(\), save\(\) and delete\(\) methods](#)
6. [Create personForm.xhtml to edit a person](#)
7. [Configure Validation](#)
8. [Create a Canoo WebTest to test browser-like actions](#)
9. [Add link to menu](#)





### Source Code

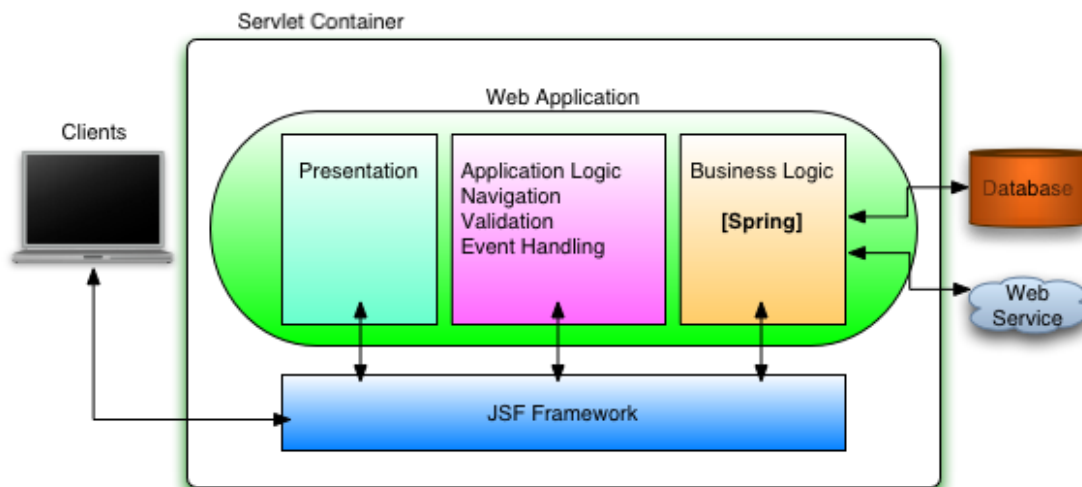
The code for this tutorial is located in the "tutorial-jsf" module of the [appfuse-demos](#) project on Google Code. Use the following command to check it out from Subversion:

```
svn checkout http://appfuse-demos.googlecode.com/svn/trunk/tutorial-jsf
```

## Introduction to JSF

JavaServer Faces (JSF) is a component-based, event-driven web framework. According to Sun Microsystems's [JSF Overview](#), JSF technology includes a set of APIs for representing UI components and managing their state, handling events and input validation, defining page navigation, and supporting internationalization and accessibility.

The figure below shows how JSF fits into a web application's architecture.



One of JSF's prominent features is the ability to wire client-generated events to server-side event handlers. For example, when a user clicks a link or a button, methods in a class can be called. These methods can be *listeners* or *actions*. Listeners typically alter the state of a page-backing Java class or *managed bean*. They can alter the JSF life cycle, but they do not typically control navigation. *Actions* are no-argument methods that return a string that signifies where to go next. Returning null from an action means, "Stay on the same page."



### Learning JSF

If you want a more in-depth learning experience, I suggest you read [David Geary's Core JSF](#). I had it close by my side and used it frequently while integrating JSF into AppFuse. Thanks for the help David and Cay (co-author)!

## Create a PersonListTest

Testing is an important part of any application, and testing a JSF application isn't too difficult. In most web frameworks, an "Action" of some sort contains the controller logic. However, with JSF, they're commonly referred to as "Managed Beans". The methods within these beans are called actions. This tutorial is not going to teach you a whole lot about how JSF works, but it will get you up and running quickly with it.

Managed Beans in JSF follow a [command pattern](#) similar to the one used by [Struts 2 Actions](#). Since these classes are simple POJOs and don't depend on the Servlet API at all, they're very easy to test. AppFuse does most of the hard work for you, meaning it initializes the FacesContext and allows you to grab pre-configured managed beans. [Shale](#) has a [testing framework](#) that AppFuse uses to simplify testing JSF beans even more.

Create a `PersonListTest.java` class in `src/test/java/**/webapp/action`:



```

package org.appfuse.tutorial.webapp.action;

import org.appfuse.service.GenericManager;
import org.appfuse.tutorial.model.Person;

import org.junit.After;
import org.junit.Before;
import org.junit.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;

import static org.junit.Assert.*;

public class PersonListTest extends BasePageTestCase {
    private PersonList bean;
    @Autowired @Qualifier("personManager")
    private GenericManager<Person, Long> personManager;

    @Override
    @Before
    @SuppressWarnings("unchecked")
    public void onSetUp() {
        super.onSetUp();
        bean = new PersonList();
        bean.setPersonManager(personManager);

        // add a test person to the database
        Person person = new Person();
        person.setFirstName("Abbie Loo");
        person.setLastName("Raible");
        personManager.save(person);
    }

    @Override
    @After
    public void onTearDown() {
        super.onTearDown();
        bean = null;
    }

    @Test
    public void testSearch() throws Exception {
        assertTrue(bean.getPersons().size() >= 1);
        assertFalse(bean.hasErrors());
    }
}

```

This class will not compile until you create the `PersonList` class.

### Create a `PersonList` class that will fetch people

Create a `PersonList.java` file in `src/main/java/**/webapp/action`:

```

package org.appfuse.tutorial.webapp.action;

import java.io.Serializable;
import java.util.List;

import org.appfuse.tutorial.model.Person;
import org.appfuse.service.GenericManager;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;

public class PersonList extends BasePage implements Serializable {
    private GenericManager<Person, Long> personManager;

    @Autowired
    public void setPersonManager(@Qualifier("personManager") GenericManager<Person, Long>
personManager) {
        this.personManager = personManager;
    }

    public PersonList() {
        setSortColumn("id"); // sets the default sort column
    }

    public List getPersons() {
        return sort(personManager.getAll());
    }
}

```

The `sort()` method (called in the `getPersons()` method) is in the `BasePage` class. This method makes it possible for you to sort columns in the UI.

Now you need to register this as a managed-bean with JSF. Spring's `SpringBeanFacesELResolver` (defined in `faces-config.xml`) makes this easy to do. Simply add the following annotations at the class-level:

```

import org.springframework.context.annotation.Scope;
import org.springframework.stereotype.Component;

@Scope("request")
@Component("personList")
public class PersonList extends BasePage implements Serializable {

```

If you run `mvn test -Dtest=PersonListTest`, your test should pass.

Nice!

**BUILD SUCCESSFUL**

Total time: 11 seconds

## Create persons.xhtml to show search results

Create a `src/main/webapp/persons.xhtml` page to display the list of people:

```

<html xmlns="http://www.w3.org/1999/xhtml" xmlns:c="http://java.sun.com/jstl/core"
xmlns:f="http://java.sun.com/jsf/core" xmlns:h="http://java.sun.com/jsf/html"
xmlns:ui="http://java.sun.com/jsf/facelets" xmlns:t="http://myfaces.apache.org/tomahawk">

<f:view>
<f:loadBundle var="text" basename="#{personList.bundleName}"/>
<head>
    <title>#{text['personList.title']}</title>
    <meta name="heading" content="#{text['personList.heading']}"/>
    <meta name="menu" content="PersonMenu"/>
</head>

```

```

<body id="personList">
<h:form id="editPerson">

<h:commandButton value="#{text['button.add']}" action="add" id="add" immediate="true" styleClass=
"button"/>
<h:commandButton value="#{text['button.done']}" action="mainMenu" id="cancel" immediate="true"
styleClass="button" style="margin-left: 5px"/>

<!-- Error from this table is caused by http://issues.apache.org/jira/browse/TOMAHAWK-466 -->
<t:dataTable id="persons" var="person" style="margin-top: 10px"
    value="#{personList.persons}" rows="25" sortColumn="#{personList.sortColumn}"
    sortAscending="#{personList.sortAscending}" styleClass="scrollerTable table"
    headerClass="standardTable_Header" rowClasses="standardTable_Row1,standardTable_Row2"
    columnClasses=
"standardTable_Column,standardTable_Column,standardTable_Column,standardTable_Column,standardTable_Column
">

    <t:column>
        <f:facet name="header">
            <t:commandSortHeader columnName="id" arrow="true">
                <h:outputText value="#{text['person.id']}" />
            </t:commandSortHeader>
        </f:facet>
        <h:commandLink action="#{personForm.edit}" value="#{person.id}">
            <f:param name="id" value="#{person.id}" />
            <f:param name="from" value="list" />
        </h:commandLink>
    </t:column>

    <t:column>
        <f:facet name="header">
            <t:commandSortHeader columnName="firstName" arrow="true">
                <h:outputText value="#{text['person.firstName']}" />
            </t:commandSortHeader>
        </f:facet>
        <h:outputText value="#{person.firstName}" escape="true"/>
    </t:column>

    <t:column>
        <f:facet name="header">
            <t:commandSortHeader columnName="lastName" arrow="true">
                <h:outputText value="#{text['person.lastName']}" />
            </t:commandSortHeader>
        </f:facet>
        <h:outputText value="#{person.lastName}" escape="true"/>
    </t:column>

</t:dataTable>

<ui:include src="/common/tableFooter.xhtml">
    <ui:param name="tableName" value="persons"/>
</ui:include>

<script type="text/javascript">
    highlightTableRows("editPerson:persons");
</script>

</h:form>
</body>

```

```
</f:view>
</html>
```

Add a navigation-rule in `src/main/webapp/WEB-INF/faces-config.xml` that routes clicking on the Add button to the `personForm.xhtml` you'll create in step 5.

```
<navigation-rule>
  <from-view-id>/persons.xhtml</from-view-id>
  <navigation-case>
    <from-outcome>add</from-outcome>
    <to-view-id>/personForm.xhtml</to-view-id>
  </navigation-case>
</navigation-rule>
```

Open `src/main/resources/ApplicationResources.properties` and add 18n keys/values for the various "person" properties:

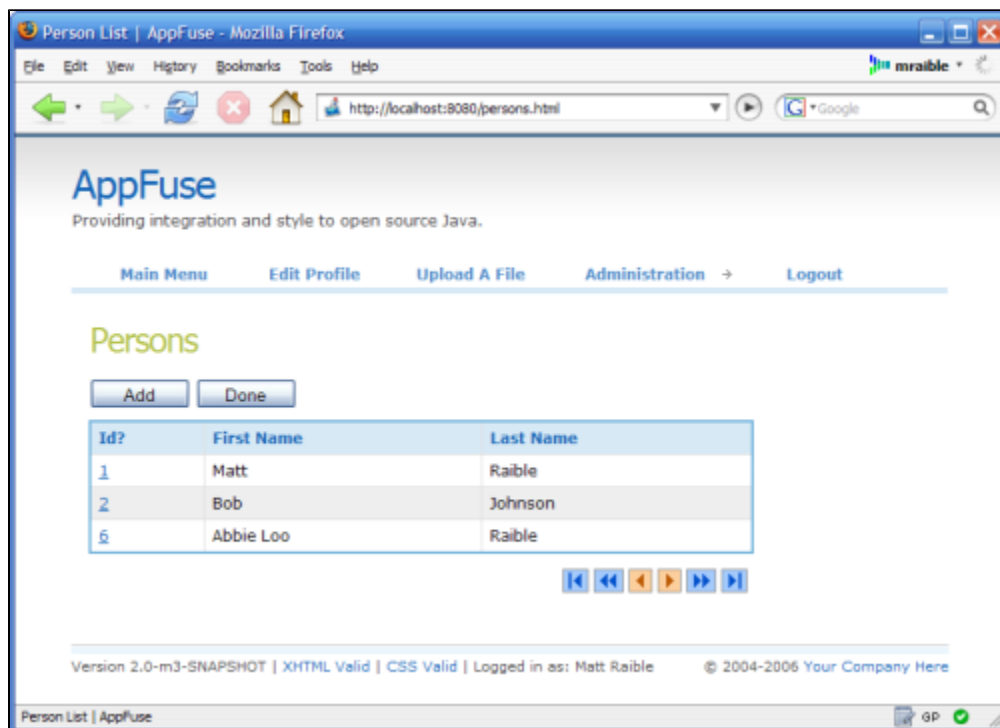
```
# -- person form --
person.id=Id
person.firstName=First Name
person.lastName=Last Name

person.added=Person has been added successfully.
person.updated=Person has been updated successfully.
person.deleted=Person has been deleted successfully.

# -- person list page --
personList.title=Person List
personList.heading=Persons

# -- person detail page --
personDetail.title=Person Detail
personDetail.heading=Person Information
```

Run `mvn jetty:run` and open `http://localhost:8080/persons` in your browser. Login with **admin/admin** and you should see a screen similar to the figure below.



Security settings for AppFuse specify that most url-patterns should be protected (except for /signup and /passwordHint). This guarantees that clients must go through JSF's FacesServlet to get to view pages.



### CSS Customization

If you want to customize the CSS for a particular page, you can add `<body id="pageName"/>` to the top of the file. This will be slurped up by SiteMesh and put into the final page. You can then customize your CSS on a page-by-page basis using something like the following:

```
body#pageName element.class { background-color: blue }
```

## Create a PersonFormTest and PersonForm for edit(), save() and delete() methods

To start creating the detail screen, create a `PersonFormTest.java` class in `src/test/java/**/webapp/action`:

```
package org.appfuse.tutorial.webapp.action;

import org.appfuse.service.GenericManager;
import org.appfuse.tutorial.model.Person;
import org.junit.After;
import org.junit.Before;
import org.junit.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;

import static org.junit.Assert.*;

public class PersonFormTest extends BasePageTestCase {
    private PersonForm bean;
    @Autowired @Qualifier("personManager")
    private GenericManager<Person, Long> personManager;

    @Override
    @Before
    public void onSetUp() {
        super.onSetUp();
        bean = new PersonForm();
        bean.setPersonManager(personManager);
    }

    @Override
    @After
    public void onTearDown() {
        super.onTearDown();
        bean = null;
    }

    @Test
    public void testAdd() throws Exception {
        Person person = new Person();
        // set required fields
        person.setFirstName("firstName");
        person.setLastName("lastName");
        bean.setPerson(person);

        assertEquals("list", bean.save());
        assertFalse(bean.hasErrors());
    }

    @Test
    public void testEdit() throws Exception {
        log.debug("testing edit...");
        bean.setId(1L);
    }
}
```

```
        assertEquals("edit", bean.edit());
        assertNotNull(bean.getPerson());
        assertFalse(bean.hasErrors());
    }

    @Test
    public void testSave() {
        bean.setId(1L);

        assertEquals("edit", bean.edit());
        assertNotNull(bean.getPerson());
        Person person = bean.getPerson();

        // update fields
        person.setFirstName("firstName");
        person.setLastName("lastName");
        bean.setPerson(person);

        assertEquals("edit", bean.save());
        assertFalse(bean.hasErrors());
    }

    @Test
    public void testRemove() throws Exception {
        Person person = new Person();
        person.setId(2L);
        bean.setPerson(person);

        assertEquals("list", bean.delete());
        assertFalse(bean.hasErrors());
    }
}
```

```
}  
}
```

Nothing will compile at this point; you need to create the `PersonForm` that you're referring to in this test.

In `src/main/java/**/webapp/action`, create a `PersonForm.java` class that extends AppFuse's [BasePage](#). Populate it with the following code:

```

package org.appfuse.tutorial.webapp.action;

import java.io.Serializable;
import org.appfuse.tutorial.model.Person;
import org.appfuse.service.GenericManager;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.context.annotation.Scope;
import org.springframework.stereotype.Component;

@Scope("request")
@Component("personForm")
public class PersonForm extends BasePage implements Serializable {
    private GenericManager<Person, Long> personManager;
    private Person person = new Person();
    private Long id;

    @Autowired
    public void setPersonManager(@Qualifier("personManager") GenericManager<Person, Long> manager) {
        this.personManager = manager;
    }

    public Person getPerson() {
        return person;
    }

    public void setPerson(Person person) {
        this.person = person;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String delete() {
        personManager.remove(person.getId());
        addMessage("person.deleted");

        return "list";
    }

    public String edit() {
        if (id == null) {
            id = new Long(getParameter("id"));
        }
        person = personManager.get(id);

        return "edit";
    }

    public String save() {
        boolean isNew = (person.getId() == null);
        personManager.save(person);

        String key = (isNew) ? "person.added" : "person.updated";
        addMessage(key);

        if (isNew) {
            return "list";
        } else {
            return "edit";
        }
    }
}

```

You might notice a number of keys in this file - "person.deleted", "person.added" and "person.updated". These are all keys that need to be in your i18n bundle (ApplicationResources.properties). You should've added these at the beginning of this tutorial. If you want to customize



these messages, to add a person's name or something, simply add a {0} placeholder in the key's message and then use the `addMessage(key, string.replace)` method.

Near the top of `faces-config.xml`, add navigation-rules that controls page flow after actions are executed:

```
<navigation-rule>
  <from-view-id>/personForm.xhtml</from-view-id>
  <navigation-case>
    <from-outcome>cancel</from-outcome>
    <to-view-id>/persons.xhtml</to-view-id>
    <redirect/>
  </navigation-case>
  <navigation-case>
    <from-outcome>edit</from-outcome>
    <to-view-id>/personForm.xhtml</to-view-id>
  </navigation-case>
  <navigation-case>
    <from-outcome>list</from-outcome>
    <to-view-id>/persons.xhtml</to-view-id>
    <redirect/>
  </navigation-case>
</navigation-rule>
```

You'll also need to modify the navigation-rule you added for `persons.xhtml` so an "edit" result routes to the `personForm.xhtml` as well. The first part of the following XML should already exist in `faces-config.xml`.

```
<navigation-rule>
  <from-view-id>/persons.xhtml</from-view-id>
  <navigation-case>
    <from-outcome>add</from-outcome>
    <to-view-id>/personForm.xhtml</to-view-id>
  </navigation-case>
  <navigation-case>
    <from-outcome>edit</from-outcome>
    <to-view-id>/personForm.xhtml</to-view-id>
  </navigation-case>
</navigation-rule>
```

If you look at your `PersonFormTest`, all the tests depend on having a record with `id=1` in the database (and `testRemove` depends on `id=2`), so let's add those records to our sample data file (`src/test/resources/sample-data.xml`). Adding it at the bottom should work - order is not important since it (currently) does not relate to any other tables. If you already have this table, make sure the 2nd record exists so `testRemove()` doesn't fail.

```
<table name='person'>
  <column>id</column>
  <column>first_name</column>
  <column>last_name</column>
  <row>
    <value>1</value>
    <value>Matt</value>
    <value>Raible</value>
  </row>
  <row>
    <value>2</value>
    <value>Bob</value>
    <value>Johnson</value>
  </row>
</table>
```

DbUnit loads this file before you run any tests, so these records will be available to your `PersonFormTest` class. Save all your files and run the tests in `PersonFormTest` using the command `mvn test -Dtest=PersonFormTest`.

**BUILD SUCCESSFUL**

Total time: 13 seconds

## Create personForm.xhtml to edit a person

Create a `src/main/webapp/personForm.xhtml` page to display the form:

```
<html xmlns="http://www.w3.org/1999/xhtml" xmlns:c="http://java.sun.com/jstl/core"
xmlns:f="http://java.sun.com/jsf/core" xmlns:h="http://java.sun.com/jsf/html"
xmlns:ui="http://java.sun.com/jsf/facelets" xmlns:t="http://myfaces.apache.org/tomahawk"
xmlns:v="http://shale.apache.org/validator">

<f:view>
<f:loadBundle var="text" basename="#{personForm.bundleName}"/>
  <head>
    <title>#{text['personDetail.title']}</title>
    <meta name="heading" content="#{text['personDetail.heading']}/>
    <meta name="menu" content="PersonMenu"/>
  </head>
<body id="personForm">

<h:form id="personForm">
<h:inputHidden value="#{personForm.person.id}" id="id"/>

<h:panelGrid columns="3">

  <h:outputLabel styleClass="desc" for="firstName" value="#{text['person.firstName']}/>
  <h:inputText styleClass="text medium" id="firstName" value="#{personForm.person.firstName}"/>
  <t:message for="firstName" styleClass="fieldError"/>

  <h:outputLabel styleClass="desc" for="lastName" value="#{text['person.lastName']}/>

  <h:inputText styleClass="text medium" id="lastName" value="#{personForm.person.lastName}"/>
  <t:message for="lastName" styleClass="fieldError"/>

  <h:panelGroup styleClass="buttonBar bottom">
    <h:commandButton value="#{text['button.save']}" action="#{personForm.save}" id="save"
styleClass="button"/>

    <c:if test="${not empty personForm.person.id}">
      <h:commandButton value="#{text['button.delete']}" action="#{personForm.delete}"
        id="delete" styleClass="button"/>
    </c:if>

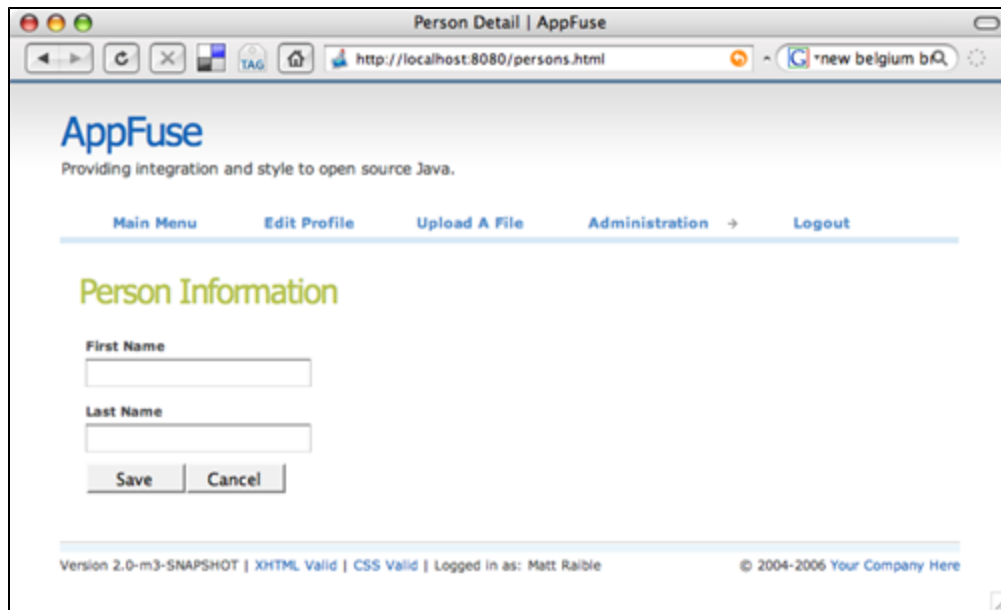
    <h:commandButton value="#{text['button.cancel']}" action="cancel" immediate="true"
      id="cancel" styleClass="button"/>
  </h:panelGroup>
  <h:outputText/><h:outputText/>
</h:panelGrid>
</h:form>

<script type="text/javascript">
  $('#personForm:firstName').focus();
  highlightFormElements();
</script>

</body>
</f:view>
</html>
```

JSF reduces the amount of HTML you have to write for a form. Normally, JSF's `<h:panelGrid>` handles rendering an HTML `<table>` for you. However, since AppFuse uses Wufoo's (much prettier) form layout, we've modified this tag so it renders `<ul>` and `<li>` tags instead.

Run `mvn jetty:run`, open your browser to <http://localhost:8080/persons>, and click on the **Add** button.



Fill in the first name and last name fields and click the Save button. This should route you to the list screen, where a success message flashes and the new person displays in the list.



### Displaying success messages

The `src/main/webapp/common/messages.jsp` file in AppFuse renders the success message in this screen. This file is included in `decorators/default.jsp`. It also handles displaying validation errors:

```
<%-- Error Messages --%>
<c:if test="${not empty errors}">
  <div class="error" id="errorMessages">
    <c:forEach var="error" items="${errors}">
      "
        alt="<fmt:message key="icon.warning"/>" class="icon" />
      <c:out value="${error}"/><br />
    </c:forEach>
  </div>
  <c:remove var="errors" scope="session"/>
</c:if>

<%-- Success Messages --%>
<c:if test="${not empty messages}">
  <div class="message" id="successMessages">
    <c:forEach var="msg" items="${messages}">
      "
        alt="<fmt:message key="icon.information"/>" class="icon" />
      <c:out value="${msg}"/><br />
    </c:forEach>
  </div>
  <c:remove var="messages" scope="session"/>
</c:if>
```

## Configure Validation

Implementing validation with JSF is quite simple. For server-side validation, all you need to do is add **required="true"** to your `<h:inputText>` tags. For client-side validation, you need to add a nested `<v:commonsValidator>` tag. Other validations (besides required) can be specified by nested validation tags in your `inputText` tags.

In the `personForm.xhtml` you created, there is no validation-related information. Therefore, perform the following steps to make `firstName` and `lastName` required fields.

1. Add `required="true"` to the `inputText` fields in `personForm.xhtml`. This makes these fields required on the server-side.
2. Add a `<v:commonsValidator client="true">` tag to these fields to enable client-side validation (code below).

3. Add a `<v:validatorScript>` tag after your form to print out the JavaScript functions for client-side validation.
4. Add an `onSubmit()` event handler to the `<h:form>` tag.
5. Add `onclick()` event handlers to the Cancel and Delete buttons to disable validation.

Below is the revised contents of the `<h:form>` tag with these changes. Replace the `<h:form>` in your `personForm.xhtml` with these changes.

```
<h:form id="personForm" onsubmit="return validatePersonForm(this)">
<h:inputHidden value="#{personForm.person.id}" id="id"/>

<h:panelGrid columns="3">
  <h:outputLabel styleClass="desc" for="firstName" value="#{text['person.firstName']}" />
  <h:inputText styleClass="text medium" id="firstName" value="#{personForm.person.firstName}"
required="true">
    <v:commonsValidator client="true" type="required" arg="#{text['person.firstName']}" />
  </h:inputText>
  <t:message for="firstName" styleClass="fieldError" />

  <h:outputLabel styleClass="desc" for="lastName" value="#{text['person.lastName']}" />
  <h:inputText styleClass="text medium" id="lastName" value="#{personForm.person.lastName}"
required="true">
    <v:commonsValidator client="true" type="required" arg="#{text['person.lastName']}" />
  </h:inputText>
  <t:message for="lastName" styleClass="fieldError" />

  <h:panelGroup styleClass="buttonBar bottom">
    <h:commandButton value="#{text['button.save']}" action="#{personForm.save}" id="save"
styleClass="button" />

    <c:if test="${not empty personForm.person.id}">
      <h:commandButton value="#{text['button.delete']}" action="#{personForm.delete}"
id="delete" styleClass="button" onclick="bCancel=true; return confirmDelete('Person')"/>
    </c:if>

    <h:commandButton value="#{text['button.cancel']}" action="cancel" immediate="true"
id="cancel" styleClass="button" onclick="bCancel=true" />
  </h:panelGroup>
  <h:outputText/><h:outputText/>
</h:panelGrid>
</h:form>

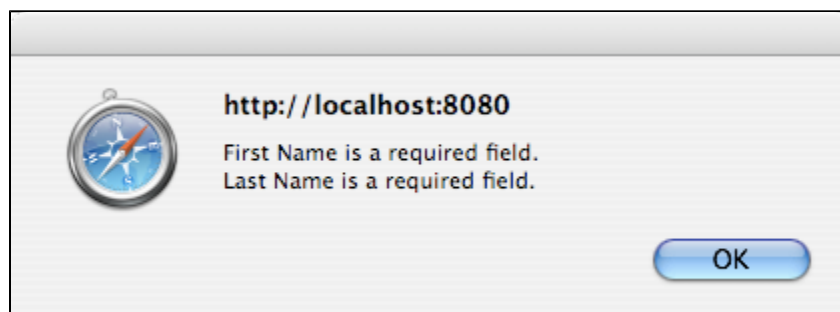
<v:validatorScript functionName="validateUserForm"/>
```



The commonsValidator and client-side validation is from [Shale's Validator Integration](#).

AppFuse contains a custom [LabelRenderer](#) for the `<h:outputLabel>` tag. This render adds asterisks for required fields.

After saving all your files and running `mvn jetty:run`, validation should kick in when you try to save this form. To test, go to <http://localhost:8080/personForm> and try to add a new user with no first or last name. You should get the following JavaScript alert:



To make sure things are *really* working as expected, you can turn off JavaScript and ensure the server-side validation is working. This is easy in [Firefox](#) (my favorite browser), just go to Tools Options Web Features and uncheck "Enable JavaScript". Now if you clear the fields and save the form, you should see the following:

## Person Information

First Name \*

This is a required field.

Last Name \*

This is a required field.

Save

Cancel

### Create a Canoo WebTest to see browser-like actions

The next (optional) step in this tutorial is to create a [Canoo WebTest](#) to test your UI. This step is optional, because you can run the same tests manually through your browser. Regardless, it's a good idea to automate as much of your testing as possible.

You can use the following URLs to test the different actions for adding, editing and saving a user.

- Add - <http://localhost:8080/personForm>.
- Edit - Go to <http://localhost:8080/persons> and click on an existing record.
- Delete - Use the edit link above and click on the Delete button.
- Save - Use the edit link above and click the Save button.



#### WebTest Recorder

There is a [WebTest Recorder](#) Firefox plugin that allows you to record your tests, rather than manually writing them.

Canoo tests are pretty slick in that they're simply configured in an XML file. To add tests for add, edit, save and delete, open `src/test/resources/web-tests.xml` and add the following XML. You'll notice that this fragment has a target named "PersonTests" that runs all the related tests.

```
<target name="PersonTests"
  depends="SearchPeople,EditPerson,SavePerson,AddPerson,DeletePerson"
  description="Call and executes all person test cases (targets)">
  <echo>Successfully ran all Person UI tests!</echo>
</target>

<!-- Verify the people list screen displays without errors -->
<target name="SearchPeople" description="Tests search for and displaying all people">
  <webtest name="searchPeople">
    &config;
    <steps>
      &login;
      <invoke description="View Persons List" url="/persons"/>
      <verifytitle description="we should see the personList title"
        text=".*${personList.title}.*" regex="true"/>
    </steps>
  </webtest>
</target>

<!-- Verify the edit person screen displays without errors -->
<target name="EditPerson" description="Tests editing an existing Person's information">
  <webtest name="editPerson">
    &config;
    <steps>
      &login;
      <invoke description="View Persons List" url="/persons"/>
      <clicklink label="1"/>
    </steps>
  </webtest>
</target>
```

```

        <verifytitle description="we should see the personDetail title"
            text=".*${personDetail.title}.*" regex="true"/>
    </steps>
</webtest>
</target>

<!-- Edit a person and then save -->
<target name="SavePerson" description="Tests editing and saving a person">
    <webtest name="savePerson">
        &config;
        <steps>
            &login;
            <invoke description="View Person List" url="/persons"/>
            <clicklink label="1"/>
            <verifytitle description="we should see the personDetail title"
                text=".*${personDetail.title}.*" regex="true"/>
            <!-- update some of the required fields -->

            <clickbutton label="${button.save}" description="Click Save"/>
            <verifytitle description="Page re-appears if save successful"
                text=".*${personDetail.title}.*" regex="true"/>
            <verifytext description="verify success message" text="${person.updated}"/>
        </steps>
    </webtest>
</target>

<!-- Add a new Person -->
<target name="AddPerson" description="Adds a new Person">
    <webtest name="addPerson">
        &config;
        <steps>
            &login;
            <invoke description="View Person List" url="/persons"/>
            <clickbutton description="Click the 'Add' button" label="${button.add}"/>
            <verifytitle description="we should see the personDetail title"
                text=".*${personDetail.title}.*" regex="true"/>

            <!-- enter required fields -->
            <setinputfield description="set firstName" name="personForm:firstName" value="Jack"/>
            <setinputfield description="set lastName" name="personForm:lastName" value="Raible"/>
            <clickbutton label="${button.save}" description="Click button 'Save'"/>
            <verifytitle description="Person List appears if save successful"
                text=".*${personList.title}.*" regex="true"/>
            <verifytext description="verify success message" text="${person.added}"/>
        </steps>
    </webtest>
</target>

<!-- Delete existing person -->
<target name="DeletePerson" description="Deletes existing Person">
    <webtest name="deletePerson">
        &config;
        <steps>
            &login;
            <invoke description="View Person List" url="/persons"/>
            <clicklink label="2"/>
            <prepareDialogResponse description="Confirm delete" dialogType="confirm" response="true"/>
            <clickbutton label="${button.delete}" description="Click button 'Delete'"/>
            <verifyNoDialogResponses/>
            <verifytitle description="display Person List" text=".*${personList.title}.*" regex="true"
/>

            <verifytext description="verify success message" text="${person.deleted}"/>
        </steps>
    </webtest>
</target>

```

```

    </webtest>
</target>

```

To include the PersonTests when all Canoo tests are run, add it as a dependency to the "run-all-tests" target in *src/test/resources/web-test.xml*.

```

<target name="run-all-tests"
    depends=
    "Login,Logout,PasswordHint,Signup,UserTests,StaticPages,WebServices,DWR,FileUpload,PersonTests"
    description="Call and executes all test cases (targets)"/>

```

After adding this, you should be able to run **mvn verify** and have these tests execute. If this command results in "BUILD SUCCESSFUL" - nice work!

## Add link to menu

The last step is to make the list, add, edit and delete functions visible to the user. The simplest way is to add a new link to the list of links in *src/main/webapp/WEB-INF/pages/mainMenu.jsp*:

```

<li>
    <a href="persons">#{text['menu.viewPeople']}</a>
</li>

```

Where `menu.viewPeople` is an entry in *src/main/resources/ApplicationResources.properties*.

```

menu.viewPeople=View People

```

The other (more likely) alternative is that you'll want to add it to the menu. To do this, add the following to *src/main/webapp/WEB-INF/menu-config.xml*:

```

<Menu name="PeopleMenu" title="menu.viewPeople" page="/persons"/>

```

Make sure the above XML is inside the `<Menus>` tag, but not within another `<Menu>`. Then create *src/main/webapp/common/menu.jsp* and add the following code to it.

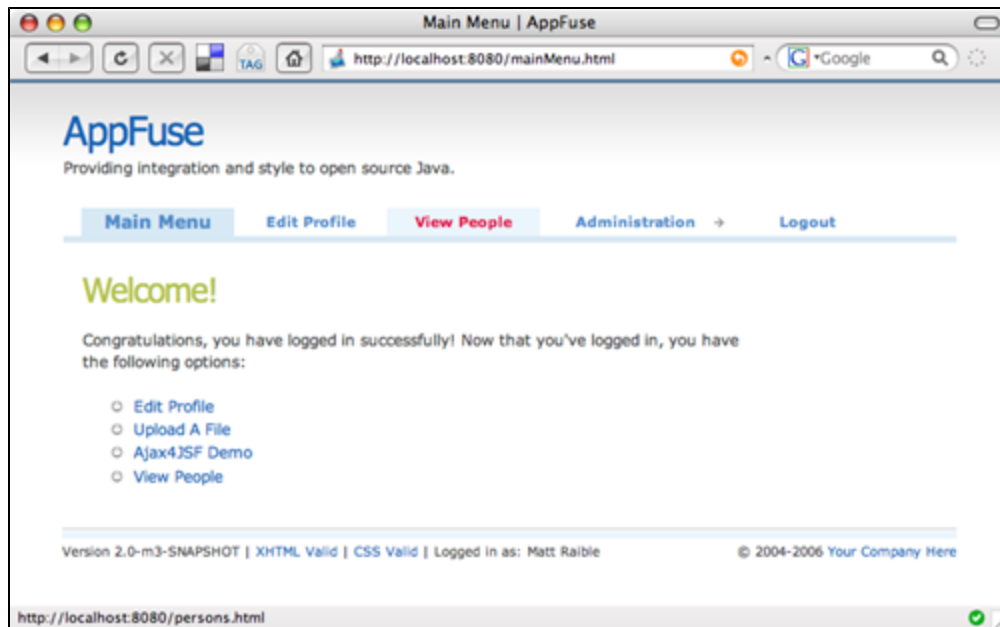
```

<%@ include file="/common/taglibs.jsp"%>

<menu:useMenuDisplay name="Velocity" config="WEB-INF/classes/cssHorizontalMenu.vm" permissions=
"rolesAdapter">
<ul id="primary-nav" class="menuList">
    <li class="pad">&nbsp;</li>
    <c:if test="{empty pageContext.request.remoteUser}">
        <li><a href="{c:url value="/login"/}" class="current">
            <fmt:message key="login.title"/></a></li>
    </c:if>
    <menu:displayMenu name="MainMenu"/>
    <menu:displayMenu name="UserMenu"/>
    <menu:displayMenu name="PeopleMenu"/>
    <menu:displayMenu name="AdminMenu"/>
    <menu:displayMenu name="Logout"/>
</ul>
</menu:useMenuDisplay>

```

Now if you run **mvn jetty:run** and go to <http://localhost:8080/mainMenu>, you should see something like the screenshot below.



Notice that there is a new link in your main screen (from mainMenu.xhtml) and on the top in your menu bar (from menu.jsp).

#### That's it!

You've completed the full lifecycle of developing a set of master-detail pages with AppFuse and JSF - **Congratulations!**

Because it's so much fun to watch tests fly by and success happen, run all your tests again using **mvn install**.

#### Happy Day!

**BUILD SUCCESSFUL**

Total time: 58 seconds

## Using Spring MVC

### About this Tutorial

This tutorial will show you how to create master/detail screens with [Spring MVC](#). The list (master) screen will have the ability to sort columns, as well as page 25 records at a time. The form (detail) screen will use an elegant CSS form layout (courtesy of [Wufoo](#)). You will also configure client and server-side validation to improve your users' experience.



#### IntelliJ IDEA Rocks

We highly recommend using IntelliJ IDEA when developing web applications in Java. Not only is its Java and JUnit support fantastic, but it has excellent CSS and JavaScript support. Using [JRebel with IDEA](#) is likely to provide you with the most pleasant Java development experiences you've ever had.



This tutorial assumes you've created a project with the **appfuse-basic-spring** archetype and have already completed the [Persistence](#) and [Services](#) tutorials. If you're using the **appfuse-modular-spring** archetype, please morph your mind into using the *web* module as the root directory. If you created your project with a different web framework than Spring MVC, you're likely to be confused and nothing will work in this tutorial. 😊

### Table of Contents

1. Introduction to Spring MVC
2. Create a PersonControllerTest
3. Create a PersonController that will fetch people
4. Create persons.jsp that shows search results
5. Create a PersonFormControllerTest and PersonFormController
6. Create personform.jsp to edit a person
7. Configure Validation
8. Create a Canoo WebTest to test browser-like actions
9. Add link to menu







### Source Code

The code for this tutorial is located in the "tutorial-spring" of the [appfuse-demos](#) project on Google Code. Use the following command to check it out from Subversion:

```
svn checkout http://appfuse-demos.googlecode.com/svn/trunk/tutorial-spring
```

## Introduction to Spring MVC

Spring 2.x had two main types of controllers: a [Controller](#) interface and a [SimpleFormController](#) class. The [Controller](#) is best suited for displaying read-only data (such as list screens), while the [SimpleFormController](#) handles forms (such as edit, save, delete). The [Controller](#) interface is quite simple, containing a single `handleRequest(request, response)` method.

As of version 2.5, Spring MVC began supporting annotations and made configuration much simpler.

### Create a `PersonControllerTest`

Testing is an important part of any application, and testing a Spring MVC web application is pretty easy. Not only are Spring's controllers lightweight, they're also easy to test using Spring's Mock library. This library has mock implements for much of the Servlet API and makes it quite simple to test Spring Controllers.

Create a `PersonControllerTest.java` class in `src/test/java/**/webapp/controller`.

```
package org.appfuse.tutorial.webapp.controller;

import org.springframework.ui.ModelMap;
import org.springframework.web.servlet.ModelAndView;
import org.springframework.beans.factory.annotation.Autowired;

import java.util.List;
import java.util.Map;

import org.junit.Test;
import static org.junit.Assert.*;

public class PersonControllerTest extends BaseControllerTestCase {
    @Autowired
    private PersonController controller;

    @Test
    public void testHandleRequest() throws Exception {
        ModelAndView mav = controller.handleRequest();
        ModelMap m = mav.getModelMap();
        assertNotNull(m.get("personList"));
        assertTrue(((List) m.get("personList")).size() > 0);
    }
}
```

This class will not compile until you create the `PersonController` class.

### Create a `PersonController` that will fetch people

Create a `PersonController.java` class in `src/main/java/**/webapp/controller`.

```

package org.appfuse.tutorial.webapp.controller;

import org.appfuse.service.GenericManager;
import org.appfuse.tutorial.model.Person;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.servlet.ModelAndView;

@Controller
@RequestMapping("/persons*")
public class PersonController {
    private GenericManager<Person, Long> personManager;

    @Autowired
    public void setPersonManager(@Qualifier("personManager") GenericManager<Person, Long>
personManager) {
        this.personManager = personManager;
    }

    @RequestMapping(method = RequestMethod.GET)
    public ModelAndView handleRequest()
throws Exception {
        return new ModelAndView().addObject(personManager.getAll());
    }
}

```

AppFuse leverages Spring MVC's [convention over configuration](#) features. This means the following:

- The default view page will be located at `src/main/webapp/WEB-INF/pages/persons.jsp` because of the [RequestToViewNameTranslator](#) (check out the [DefaultRequestToViewNameTranslator](#)).
- The list of people (the model) will be available from "personList" thanks to [ModelMap](#).

Run the `PersonControllerTest` using your IDE or `mvn test -Dtest=PersonControllerTest`.

### Create persons.jsp to show search results

Create a `src/main/webapp/WEB-INF/pages/persons.jsp` page to display the list of people:

```

<%@ include file="/common/taglibs.jsp"%>

<head>
  <title><fmt:message key="personList.title"/></title>
  <meta name="heading" content="<fmt:message key='personList.heading'/>"/>
  <meta name="menu" content="PersonMenu"/>
</head>

<input type="button" style="margin-right: 5px" onclick="location.href='<c:url value="/personform"/>' "
value="<fmt:message key="button.add"/>"/>
<input type="button" onclick="location.href='<c:url value="/mainMenu"/>' " value="<fmt:message key="
button.done"/>"/>

<display:table name="personList" class="table" requestURI="" id="personList" export="true" pagesize="
25">
  <display:column property="id" sortable="true" href="personform" media="html"
    paramId="id" paramProperty="id" titleKey="person.id"/>
  <display:column property="id" media="csv excel xml pdf" titleKey="person.id"/>
  <display:column property="firstName" sortable="true" titleKey="person.firstName"/>
  <display:column property="lastName" sortable="true" titleKey="person.lastName"/>

  <display:setProperty name="paging.banner.item_name"><fmt:message key="personList.person"
/></display:setProperty>
  <display:setProperty name="paging.banner.items_name"><fmt:message key="personList.persons"
/></display:setProperty>

  <display:setProperty name="export.excel.filename"><fmt:message key="personList.title"
/>.xls</display:setProperty>
  <display:setProperty name="export.csv.filename"><fmt:message key="personList.title"
/>.csv</display:setProperty>
  <display:setProperty name="export.pdf.filename"><fmt:message key="personList.title"
/>.pdf</display:setProperty>
</display:table>

<input type="button" style="margin-right: 5px" onclick="location.href='<c:url value="/personform"/>' "
value="<fmt:message key="button.add"/>"/>
<input type="button" onclick="location.href='<c:url value="/mainMenu"/>' " value="<fmt:message key="
button.done"/>"/>

<script type="text/javascript">
  highlightTableRows("personList");
</script>

```

Open `src/main/resources/ApplicationResources.properties` and add 18 keys/values for the various "person" properties:

```

# -- person form --
person.id=Id
person.firstName=First Name
person.lastName=Last Name

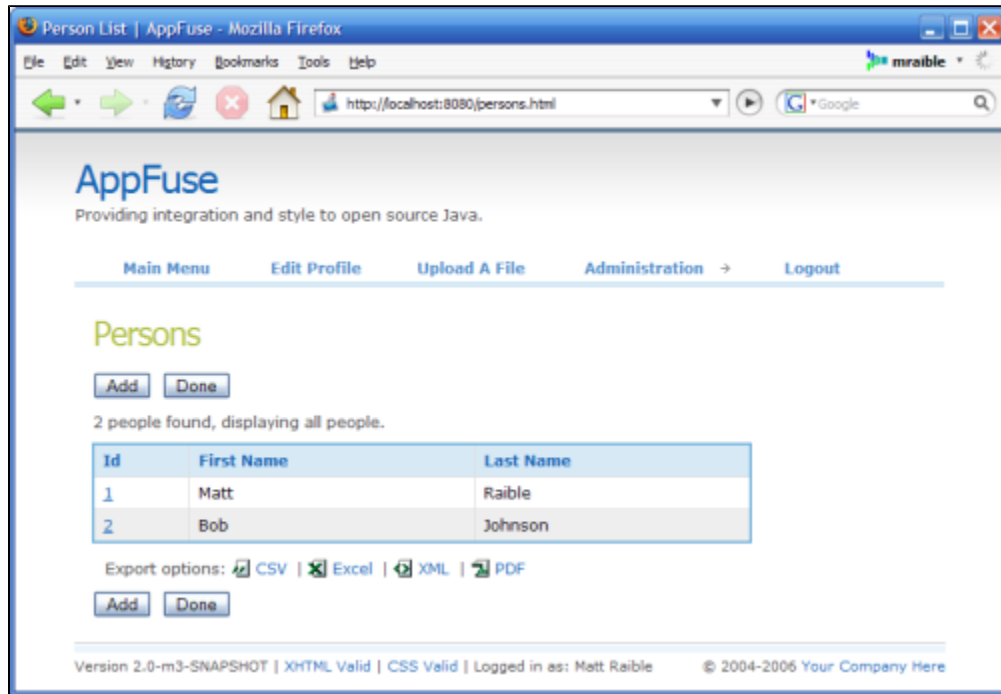
person.added=Person has been added successfully.
person.updated=Person has been updated successfully.
person.deleted=Person has been deleted successfully.

# -- person list page --
personList.title=Person List
personList.heading=Persons

# -- person detail page --
personDetail.title=Person Detail
personDetail.heading=Person Information

```

Run `mvn jetty:run` and open <http://localhost:8080/persons> in your browser. Login with **admin/admin** and you should see a screen similar to the figure below.



Security settings for AppFuse specify that most url-patterns should be protected (except for /signup and /passwordHint). This guarantees that clients must go through a Controller to get to a JSP (or at least the ones in *WEB-INF/pages*).



#### CSS Customization

If you want to customize the CSS for a particular page, you can add `<body id="pageName"/>` to the top of the file. This will be slurped up by SiteMesh and put into the final page. You can then customize your CSS on a page-by-page basis using something like the following:

```
body#pageName element.class { background-color: blue }
```

#### Create a PersonFormControllerTest and PersonFormController

To start creating the detail screen, create a `PersonFormControllerTest.java` class in `src/test/java/**/webapp/controller`.

```

package org.appfuse.tutorial.webapp.controller;

import org.appfuse.tutorial.model.Person;
import org.junit.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.mock.web.MockHttpServletRequest;
import org.springframework.mock.web.MockHttpServletResponse;
import org.springframework.validation.BindingResult;
import org.springframework.validation.DataBinder;

import static org.junit.Assert.*;

public class PersonFormControllerTest extends BaseControllerTestCase {
    @Autowired
    private PersonFormController form;
    private Person person;
    private MockHttpServletRequest request;

    @Test
    public void testEdit() throws Exception {
        log.debug("testing edit...");
        request = newGet("/personform");
        request.addParameter("id", "1");

        person = form.showForm(request);
        assertNotNull(person);
    }

    @Test
    public void testSave() throws Exception {
        request = newGet("/personform");
        request.addParameter("id", "1");

        person = form.showForm(request);
        assertNotNull(person);

        request = newPost("/personform");

        person = form.showForm(request);
        // update required fields

        BindingResult errors = new DataBinder(person).getBindingResult();
        form.onSubmit(person, errors, request, new MockHttpServletResponse());
        assertFalse(errors.hasErrors());
        assertNotNull(request.getSession().getAttribute("successMessages"));
    }

    @Test
    public void testRemove() throws Exception {
        request = newPost("/personform");
        request.addParameter("delete", "");
        person = new Person();
        person.setId(2L);

        BindingResult errors = new DataBinder(person).getBindingResult();
        form.onSubmit(person, errors, request, new MockHttpServletResponse());

        assertNotNull(request.getSession().getAttribute("successMessages"));
    }
}

```

Nothing will compile at this point; you need to create the `PersonFormController` that you're referring to in this test.

In `src/main/java/**/webapp/controller`, create a `PersonFormController.java` class that extends AppFuse's `BaseFormController`. Populate it with the following code:

```

package org.appfuse.tutorial.webapp.controller;

import org.apache.commons.lang.StringUtils;
import org.appfuse.service.GenericManager;
import org.appfuse.tutorial.model.Person;
import org.appfuse.tutorial.webapp.controller.BaseFormController;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.stereotype.Controller;
import org.springframework.validation.BindingResult;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.util.Locale;

@Controller
@RequestMapping("/personform*")
public class PersonFormController extends BaseFormController {
    private GenericManager<Person, Long> personManager = null;

    @Autowired
    public void setPersonManager(@Qualifier("personManager") GenericManager<Person, Long>
personManager) {
        this.personManager = personManager;
    }

    public PersonFormController() {
        setCancelView("redirect:persons");
        setSuccessView("redirect:persons");
    }

    @ModelAttribute
    @RequestMapping(method = {RequestMethod.GET, RequestMethod.POST})
    protected Person showForm(HttpServletRequest request)
    throws Exception {
        String id = request.getParameter("id");

        if (!StringUtils.isBlank(id)) {
            return personManager.get(new Long(id));
        }

        return new Person();
    }

    @RequestMapping(method = RequestMethod.POST)
    public String onSubmit(Person person, BindingResult errors, HttpServletRequest request,
        HttpServletResponse response)
    throws Exception {
        if (request.getParameter("cancel") != null) {
            return getCancelView();
        }

        log.debug("entering 'onSubmit' method...");

        boolean isNew = (person.getId() == null);
        String success = getSuccessView();
        Locale locale = request.getLocale();

        if (request.getParameter("delete") != null) {
            personManager.remove(person.getId());
            saveMessage(request, getText("person.deleted", locale));
        } else {
            personManager.save(person);
            String key = (isNew) ? "person.added" : "person.updated";
            saveMessage(request, getText(key, locale));
        }
    }
}

```

```
    if (!isNew) {  
        success = "redirect:personform?id=" + person.getId();  
    }  
}  
  
return success;
```

```
}  
}
```

You might notice a number of keys in this file - "person.deleted", "person.added" and "person.updated". These are all keys that need to be in your i18n bundle (`ApplicationResources.properties`). You should've added these at the beginning of this tutorial. If you want to customize these messages, to add the a person's name or something, simply add a `{0}` placeholder in the key's message and then use the `getText(key, stringtoreplace, locale)` method.

If you look at your `PersonFormControllerTest`, all the tests depend on having a record with `id=1` in the database (and `testRemove` depends on `id=2`), so let's add those records to our sample data file (`src/test/resources/sample-data.xml`). Adding it at the bottom should work - order is not important since it (currently) does not relate to any other tables. If you already have this table, make sure the 2nd record exists so `testRemove()` doesn't fail.

```
<table name='person'>  
  <column>id</column>  
  <column>first_name</column>  
  <column>last_name</column>  
  <row>  
    <value>1</value>  
    <value>Matt</value>  
    <value>Raible</value>  
  </row>  
  <row>  
    <value>2</value>  
    <value>Bob</value>  
    <value>Johnson</value>  
  </row>  
</table>
```

DbUnit loads this file before you run any tests, so these records will be available to your `PersonFormControllerTest` class. Save all your files and run the tests in `PersonFormControllerTest` using the command **`mvn test -Dtest=PersonFormControllerTest`**.

**BUILD SUCCESSFUL**  
Total time: 13 seconds

### Create `personform.jsp` to edit a person

Create a `src/main/webapp/WEB-INF/pages/personform.jsp` page to display the form:



```

<%@ include file="/common/taglibs.jsp"%>

<head>
  <title><fmt:message key="personDetail.title"/></title>
  <meta name="heading" content="<fmt:message key='personDetail.heading' />"/>
</head>

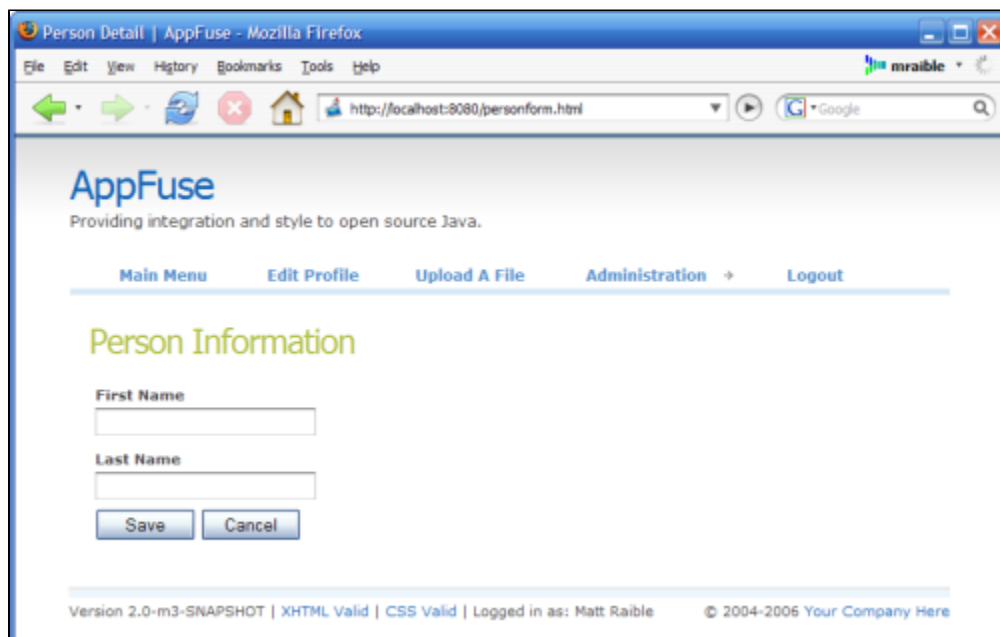
<form:form commandName="person" method="post" action="personform" id="personForm">
<form:errors path="*" cssClass="error" element="div"/>
<form:hidden path="id"/>
<ul>
  <li>
    <appfuse:label styleClass="desc" key="person.firstName"/>
    <form:errors path="firstName" cssClass="fieldError"/>
    <form:input path="firstName" id="firstName" cssClass="text medium" cssErrorClass="text medium
error" maxLength="50"/>
  </li>
  <li>
    <appfuse:label styleClass="desc" key="person.lastName"/>
    <form:errors path="lastName" cssClass="fieldError"/>
    <form:input path="lastName" id="lastName" cssClass="text medium" cssErrorClass="text medium
error" maxLength="50"/>
  </li>

  <li class="buttonBar bottom">
    <input type="submit" class="button" name="save" value="<fmt:message key="button.save"/>"/>
    <c:if test="${not empty person.id}">
      <input type="submit" class="button" name="delete" onclick="bCancel=true;return
confirmDelete('person')"
value="<fmt:message key="button.delete"/>" />
    </c:if>
    <input type="submit" class="button" name="cancel" value="<fmt:message key="button.cancel"/>"/>
    onclick="bCancel=true"/>
  </li>
</ul>
</form:form>

<script type="text/javascript">
  Form.focusFirstElement($('personForm'));
</script>

```

Run `mvn jetty:run`, open your browser to <http://localhost:8080/persons>, and click on the **Add** button.



Fill in the first name and last name fields and click the Save button. This should route you to the list screen, where a success message flashes and the new person displays in the list.



### Displaying success messages

The `src/main/webapp/common/messages.jsp` file in AppFuse renders the success message in this screen. This file is included in `decorators/default.jsp`. It also handles displaying error messages:

```
<%-- Error Messages --%>
<c:if test="${not empty errors}">
    <div class="error" id="errorMessages">
        <c:forEach var="error" items="${errors}">
            "
                alt="<fmt:message key="icon.warning"/>" class="icon" />
            <c:out value="${error}" /><br />
        </c:forEach>
    </div>
    <c:remove var="errors" />
</c:if>

<%-- Success Messages --%>
<c:if test="${not empty successMessages}">
    <div class="message" id="successMessages">
        <c:forEach var="msg" items="${successMessages}">
            "
                alt="<fmt:message key="icon.information"/>" class="icon" />
            <c:out value="${msg}" /><br />
        </c:forEach>
    </div>
    <c:remove var="successMessages" scope="session" />
</c:if>
```

## Configure Validation

Spring MVC supports a number of different options for configuring validation. AppFuse 2.x currently uses Spring Modules' **Commons Validator** support. However, you can change this to use another validation framework if you like. The instructions below show you how to configure Commons Validator with Spring MVC.


Open `src/main/webapp/WEB-INF/validation.xml` and add rules for the person object, so both the first and last names are required:

```
<form name="person">
    <field property="firstName" depends="required">
        <arg key="person.firstName" />
    </field>
    <field property="lastName" depends="required">
        <arg key="person.lastName" />
    </field>
</form>
```


After making these changes and saving all your files, the first and last name fields should be required. To test, go to <http://localhost:8080/personform> and try to add a new person with no first or last name. You should see the following validation errors:

## Person Information

First Name is a required field.  
 Last Name is a required field.

**First Name** \* 

First Name is a required field.

**Last Name** \* 

Last Name is a required field.

To enable client-side validation, you need to make the following changes to `personform.jsp`:

1. Add an `onsubmit()` handler to the form.
2. Add `bCancel=true` to the `onclick()` handlers of the delete and cancel buttons (to cancel validation when they're clicked).
3. Add JSP Tags after the form to render the validation JavaScript functions.

Below is the revised contents of the `<form:form>` tag with these changes. Replace the `<form:form>` in your `personform.jsp` with these changes.

```

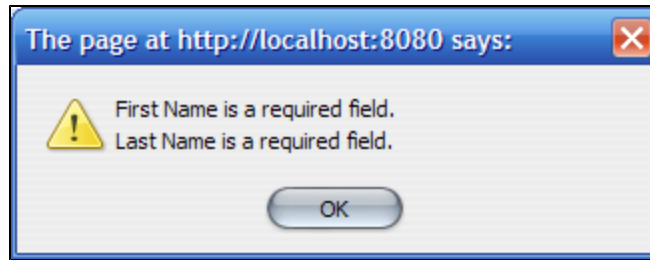
<form:form commandName="person" method="post" action="personform" id="personForm" onsubmit="return
validatePerson(this)">
<form:errors path="*" cssClass="error" element="div"/>
<form:hidden path="id"/>
<ul>
  <li>
    <appfuse:label styleClass="desc" key="person.firstName"/>
    <form:errors path="firstName" cssClass="fieldError"/>
    <form:input path="firstName" id="firstName" cssClass="text medium" cssErrorClass="text medium
error" maxLength="50"/>
  </li>
  <li>
    <appfuse:label styleClass="desc" key="person.lastName"/>
    <form:errors path="lastName" cssClass="fieldError"/>
    <form:input path="lastName" id="lastName" cssClass="text medium" cssErrorClass="text medium
error" maxLength="50"/>
  </li>

  <li class="buttonBar bottom">
    <input type="submit" class="button" name="save" value="<fmt:message key="button.save"/>" />
    <c:if test="${not empty person.id}">
      <input type="submit" class="button" name="delete" onclick="bCancel=true;return
confirmDelete('person')"
value="<fmt:message key="button.delete"/>" />
    </c:if>
    <input type="submit" class="button" name="cancel" value="<fmt:message key="button.cancel"/>"
onclick="bCancel=true"/>
  </li>
</ul>
</form:form>

<v:javascript formName="person" cdata="false" dynamicJavascript="true" staticJavascript="false"/>
<script type="text/javascript" src="<c:url value='/scripts/validator.jsp' />"></script>

```

After saving all your files and running `mvn jetty:run`, client-side validation should kick in when you try to save this form. To test, go to <http://localhost:8080/personform> and try to add a new person with no first or last name. You should get the following JavaScript alert:



## Create a Canoo WebTest to test browser-like actions

The next (optional) step in this tutorial is to create a [Canoo WebTest](#) to test your UI. This step is optional, because you can run the same tests manually through your browser. Regardless, it's a good idea to automate as much of your testing as possible.

You can use the following URLs to test the different actions for adding, editing and saving a person.

- Add - <http://localhost:8080/personform>.
- Edit - Go to <http://localhost:8080/persons> and click on an existing record.
- Delete - Use the edit link above and click on the Delete button.
- Save - Use the edit link above and click the Save button.



### WebTest Recorder

There is a [WebTest Recorder](#) Firefox plugin that allows you to record your tests, rather than manually writing them.

Canoo tests are pretty slick in that they're simply configured in an XML file. To add tests for add, edit, save and delete, open `src/test/resources/web-tests.xml` and add the following XML. You'll notice that this fragment has a target named **PersonTests** that runs all the related tests.

```
<!-- runs person-related tests -->
<target name="PersonTests"
  depends="SearchPersons,EditPerson,SavePerson,AddPerson,DeletePerson"
  description="Call and executes all person test cases (targets)">
  <echo>Successfully ran all Person UI tests!</echo>
</target>

<!-- Verify the persons list screen displays without errors -->
<target name="SearchPersons" description="Tests search for and displaying all persons">
  <webtest name="searchPersons">
    &config;
    <steps>
      &login;
      <invoke description="click View Person link" url="/persons"/>
      <verifytitle description="we should see the personList title"
        text=".*${personList.title}.*" regex="true"/>
    </steps>
  </webtest>
</target>

<!-- Verify the edit person screen displays without errors -->
<target name="EditPerson" description="Tests editing an existing Person's information">
  <webtest name="editPerson">
    &config;
    <steps>
      &login;
      <invoke description="View Person List" url="/persons"/>
      <clicklink label="1" description="Click edit link"/>
      <verifytitle description="we should see the personDetail title"
        text=".*${personDetail.title}.*" regex="true"/>
    </steps>
  </webtest>
</target>

<!-- Edit a person and then save -->
<target name="SavePerson" description="Tests editing and saving a person">
```

```

<webtest name="savePerson">
  &config;
  <steps>
    &login;
    <invoke description="click Edit Person link" url="/personform?id=1"/>
    <verifytitle description="we should see the personDetail title"
      text=".*${personDetail.title}.*" regex="true"/>

    <clickbutton label="${button.save}" description="Click Save"/>
    <verifytitle description="Page re-appears if save successful"
      text=".*${personDetail.title}.*" regex="true"/>
    <verifytext description="verify success message" text="${person.updated}"/>
  </steps>
</webtest>
</target>

<!-- Add a new Person -->
<target name="AddPerson" description="Adds a new Person">
  <webtest name="addPerson">
    &config;
    <steps>
      &login;
      <invoke description="click Add Button" url="/personform"/>
      <verifytitle description="we should see the personDetail title"
        text=".*${personDetail.title}.*" regex="true"/>

      <setinputfield description="set firstName" name="firstName" value="Jack"/>
      <setinputfield description="set lastName" name="lastName" value="Raible"/>

      <clickbutton label="${button.save}" description="Click button 'Save'"/>
      <verifytitle description="Person List appears if save successful"
        text=".*${personList.title}.*" regex="true"/>
      <verifytext description="verify success message" text="${person.added}"/>
    </steps>
  </webtest>
</target>

<!-- Delete existing person -->
<target name="DeletePerson" description="Deletes existing Person">
  <webtest name="deletePerson">
    &config;
    <steps>
      &login;
      <invoke description="click Edit Person link" url="/personform?id=2"/>
      <prepareDialogResponse description="Confirm delete" dialogType="confirm" response="true"/>
      <clickbutton label="${button.delete}" description="Click button 'Delete'"/>
      <verifyNoDialogResponses/>
      <verifytitle description="display Person List" text=".*${personList.title}.*" regex="true"
/>

      <verifytext description="verify success message" text="${person.deleted}"/>
    </steps>
  </webtest>
</target>

```

```
</webtest>
</target>
```

To include the PersonTests when all Canoo tests are run, add it as a dependency to the "run-all-tests" target in *src/test/resources/web-test.xml*.

```
<target name="run-all-tests"
  depends=
    "Login,Logout,PasswordHint,Signup,UserTests,StaticPages,WebServices,DWR,FileUpload,PersonTests"
  description="Call and executes all test cases (targets)"/>
```

After adding this, you should be able to run **mvn verify** and have these tests execute. If this command results in "BUILD SUCCESSFUL" - nice work!

### Add link to menu

The last step is to make the list, add, edit and delete functions visible to the user. The simplest way is to add a new link to the list of links in *src/main/webapp/WEB-INF/pages/mainMenu.jsp*.

```
<li>
  <a href="<c:url value="/persons"/>"><fmt:message key="menu.viewPeople"/></a>
</li>
```

Where `menu.viewPeople` is an entry in *src/main/resources/ApplicationResources.properties*.

```
menu.viewPeople=View People
```

The other (more likely) alternative is that you'll want to add it to the menu. To do this, add the following to *src/main/webapp/WEB-INF/menu-config.xml*:

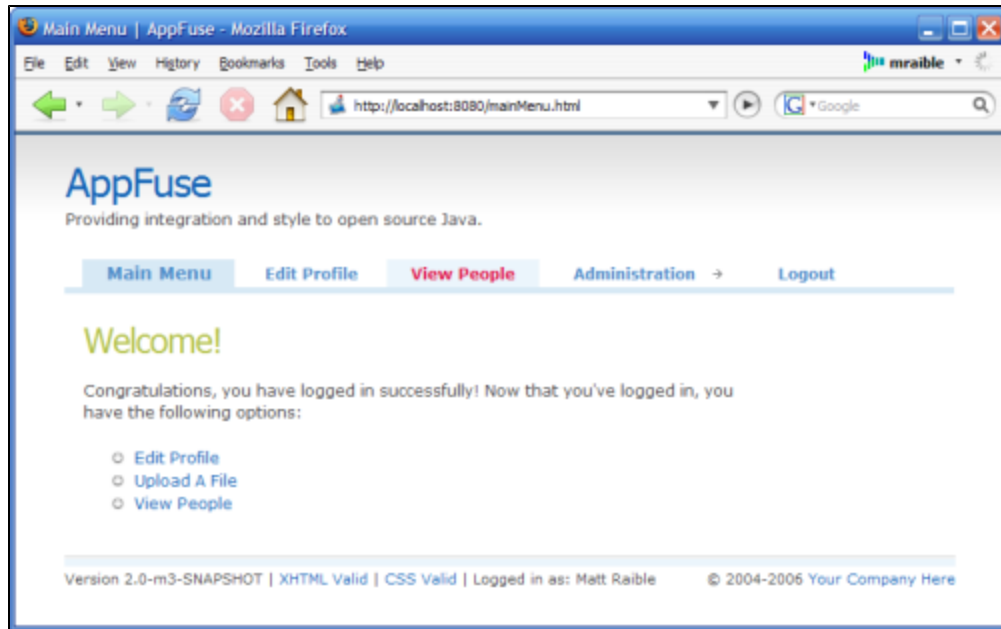
```
<Menu name="PersonMenu" title="personList.title" page="/persons"/>
```

Make sure the above XML is inside the `<Menus>` tag, but not within another `<Menu>`. Then open *src/main/webapp/common/menu.jsp* and add the following code to it:

```
<%@ include file="/common/taglibs.jsp"%>

<menu:useMenuDisplay name="Velocity" config="cssHorizontalMenu.vm" permissions="rolesAdapter">
<ul id="primary-nav" class="menuList">
  <li class="pad">&nbsp;</li>
  <c:if test="{empty pageContext.request.remoteUser}">
    <li><a href="<c:url value="/login"/>" class="current"><fmt:message key="login.title"
  /></a></li>
  </c:if>
  <menu:displayMenu name="MainMenu" />
  <menu:displayMenu name="UserMenu" />
  <menu:displayMenu name="PersonMenu" />
  <menu:displayMenu name="AdminMenu" />
  <menu:displayMenu name="Logout" />
</ul>
</menu:useMenuDisplay>
```

Now if you run **mvn jetty:run** and go to <http://localhost:8080/mainMenu>, you should see something like the screenshot below.



Notice that there is a new link in your main screen (from mainMenu.jsp) and on the top in your menu bar (from menu.jsp).

#### That's it!

You've completed the full lifecycle of developing a set of master-detail pages with AppFuse and Spring MVC - **Congratulations!**

Because it's so much fun to watch tests fly by and success happen, run all your tests again using **mvn install**.

#### Happy Day!

**BUILD SUCCESSFUL**

Total time: 48 seconds

## Using Struts 2

### About this Tutorial

This tutorial will show you how to create master/detail screens with **Struts 2**. The list (master) screen will have the ability to sort columns, as well as page 25 records at a time. The form (detail) screen will use an elegant CSS form layout (courtesy of **Wufoo**). You will also configure client and server-side validation to improve your users' experience.



#### IntelliJ IDEA Rocks

We highly recommend using IntelliJ IDEA when developing web applications in Java. Not only is its Java and JUnit support fantastic, but it has excellent CSS and JavaScript support. [Using JRebel with IDEA](#) is likely to provide you with the most pleasant Java development experiences you've ever had.



This tutorial assumes you've created a project with the **appfuse-basic-struts** archetype and have already completed the [Persistence](#) and [Services](#) tutorials. If you're using the **appfuse-modular-struts** archetype, please morph your mind into using the *web* module as the root directory. If you created your project with a different web framework than Struts, you're likely to be confused and nothing will work in this tutorial. 😊

### Table of Contents

1. Introduction to Struts 2
2. Create a PersonActionTest
3. Create a PersonAction that will fetch people
4. Create personList.jsp to show search results
5. Modify PersonActionTest and PersonAction for edit(), save() and delete() methods
6. Create personForm.jsp to edit a person
7. Configure Validation
8. Create a Canoo WebTest to test browser-like actions
9. Add link to menu

# Struts<sup>2</sup>



### Source Code

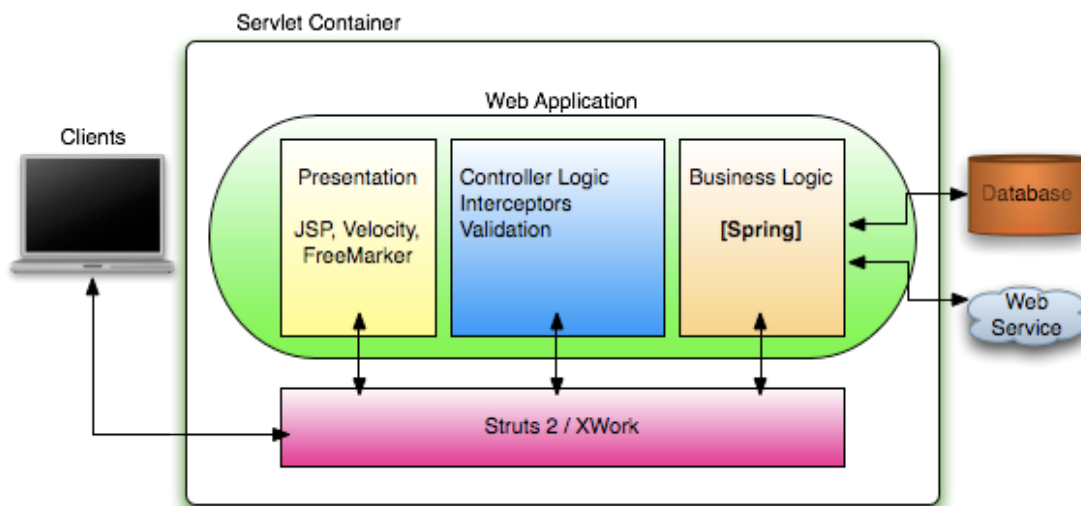
The code for this tutorial is located in the "tutorial-struts2" module of the [appfuse-demos](#) project on Google Code. Use the following command to check it out from Subversion:

```
svn checkout http://appfuse-demos.googlecode.com/svn/trunk/tutorial-struts2
```

## Introduction to Struts 2

Struts 2 (formerly WebWork) is a web framework designed with simplicity in mind. It's built on top of XWork, which is a generic command framework. XWork also has an IoC container, but it isn't as full-featured as Spring and won't be covered in this section. Struts 2 controllers are called *Actions*, mainly because they must implement the [Action](#) interface. The [ActionSupport](#) class implements this interface, and it is most common parent class for Struts 2 actions.

The figure below shows how Struts 2 fits into a web application's architecture.



Struts 2 actions typically contain methods for accessing model properties and methods for returning strings. These strings are matched with "result" names in a *struts.xml* configuration file. Actions typically have a single *execute()* method, but you can easily add multiple methods and control execution using URLs and button names.

Struts 2 uses interceptors to intercept the request and response process. This is much like Servlet Filters, except you can talk directly to the action. Struts 2 uses interceptors in the framework itself. A number of them initialize the Action, prepare it for population, set parameters on it and handle any conversion errors.

### Create a PersonActionTest

Testing is an important part of any application, and testing a Struts application is easier than most. The generic command pattern provided by XWork doesn't depend on the Servlet API at all. This makes it easy to use JUnit to test your Actions.

Create a `PersonActionTest.java` class in `src/test/java/**/webapp/action`.



```

package org.appfuse.tutorial.webapp.action;

import com.opensymphony.xwork2.ActionSupport;
import org.apache.struts2.ServletActionContext;
import org.appfuse.service.GenericManager;
import org.appfuse.tutorial.model.Person;
import org.junit.Before;
import org.junit.Test;
import org.springframework.mock.web.MockHttpServletRequest;

import static org.junit.Assert.*;

public class PersonActionTest extends BaseActionTestCase {
    private PersonAction action;

    @Before
    public void onSetUp() {
        super.onSetUp();

        action = new PersonAction();
        GenericManager personManager = (GenericManager) applicationContext.getBean("personManager");
        action.setPersonManager(personManager);

        // add a test person to the database
        Person person = new Person();

        // enter all required fields
        personManager.save(person);
    }

    @Test
    public void testGetAllPersons() throws Exception {
        assertEquals(action.list(), ActionSupport.SUCCESS);
        assertTrue(action.getPersons().size() >= 1);
    }
}

```

This class won't compile yet; you must first create the `PersonAction` class.

### Create a `PersonAction` that will fetch people

Create a `PersonAction.java` class (that extends AppFuse's `BaseAction`) in `src/main/java/**/webapp/action`:

```

package org.appfuse.tutorial.webapp.action;

import com.opensymphony.xwork2.Preparable;
import org.appfuse.service.GenericManager;
import org.appfuse.tutorial.model.Person;

import java.util.List;

public class PersonAction extends BaseAction {
    private GenericManager<Person, Long> personManager;
    private List persons;

    public void setPersonManager(GenericManager<Person, Long> personManager) {
        this.personManager = personManager;
    }

    public List getPersons() {
        return persons;
    }

    public String list() {
        persons = personManager.getAll();
        return SUCCESS;
    }
}

```

Struts 2 actions are typically both the controller and the model. In this example, the `list()` method acts as the controller, and the `getPersons()` method retrieves data from the model. This simplification of the MVC paradigm makes this web framework very easy to program with.

Run the `PersonActionTest` using your IDE or `mvn test -Dtest=PersonActionTest`.



#### Zero Configuration

Struts' [Zero Configuration](#) feature is turned on by default. If you want to configure your Actions as Spring beans, you can do that by using `class="beanId"` in your Action definition, and then defining the bean in `applicationContext.xml`. Otherwise, they will automatically be wired up by name with Spring dependencies. All you need to do is add a setter to your Action to get a Spring bean injected into it.

### Create `personList.jsp` to show search results

Create a `src/main/webapp/WEB-INF/pages/personList.jsp` page to display the list of people:

```

<%@ include file="/common/taglibs.jsp"%>

<head>
  <title><fmt:message key="personList.title"/></title>
  <meta name="heading" content="<fmt:message key='personList.heading'/>" />
  <meta name="menu" content="PersonMenu" />
</head>

<input type="button" style="margin-right: 5px" class="button" onclick="location.href='<c:url value="
/editPerson"/>' value="<fmt:message key="button.add"/>" />
<input type="button" class="button" onclick="location.href='<c:url value="/mainMenu"/>' value=
"<fmt:message key="button.done"/>" />

<display:table name="persons" class="table" requestURI="" id="personList" export="true" pagesize="25">
  <display:column property="id" sortable="true" href="editPerson" media="html"
    paramId="id" paramProperty="id" titleKey="person.id"/>
  <display:column property="id" media="csv excel xml pdf" titleKey="person.id"/>
  <display:column property="firstName" sortable="true" titleKey="person.firstName"/>
  <display:column property="lastName" sortable="true" titleKey="person.lastName"/>

  <display:setProperty name="paging.banner.item_name"><fmt:message key="personList.person"
/></display:setProperty>
  <display:setProperty name="paging.banner.items_name"><fmt:message key="personList.persons"
/></display:setProperty>

  <display:setProperty name="export.excel.filename"><fmt:message key="personList.title"
/>.xls</display:setProperty>
  <display:setProperty name="export.csv.filename"><fmt:message key="personList.title"
/>.csv</display:setProperty>
  <display:setProperty name="export.pdf.filename"><fmt:message key="personList.title"
/>.pdf</display:setProperty>
</display:table>

<input type="button" style="margin-right: 5px" class="button" onclick="location.href='<c:url value="
/editPerson"/>' value="<fmt:message key="button.add"/>" />
<input type="button" class="button" onclick="location.href='<c:url value="/mainMenu"/>' value=
"<fmt:message key="button.done"/>" />

<script type="text/javascript">
  highlightTableRows("personList");
</script>

```

Open the `struts.xml` file in the `src/main/resources` directory. Define an `<action>` (at the bottom of this file) and set its class attribute to match the fully-qualified class name of the `PersonAction` class.

```

<action name="persons" class="org.appfuse.tutorial.webapp.action.PersonAction" method="list">
  <result>/WEB-INF/pages/personList.jsp</result>
</action>

```

The default result type is "dispatcher" and its name is "success". This configured result type simply forwards you to the `personList.jsp` file when "success" is returned from `PersonAction.list()`. Other result types include `redirect` and `chain`. `Redirect` performs a client-side redirect and `chain` forwards you to another action. For a full list of result types, see Struts 2's [Result Types documentation](#).

The "method" attribute of this action has a `list` attribute, which calls the `list()` method when the "persons" URL is invoked. If you exclude the method attribute, it calls the `execute()` method.

Open `src/main/resources/ApplicationResources.properties` and add 18n keys/values for the various "person" properties:

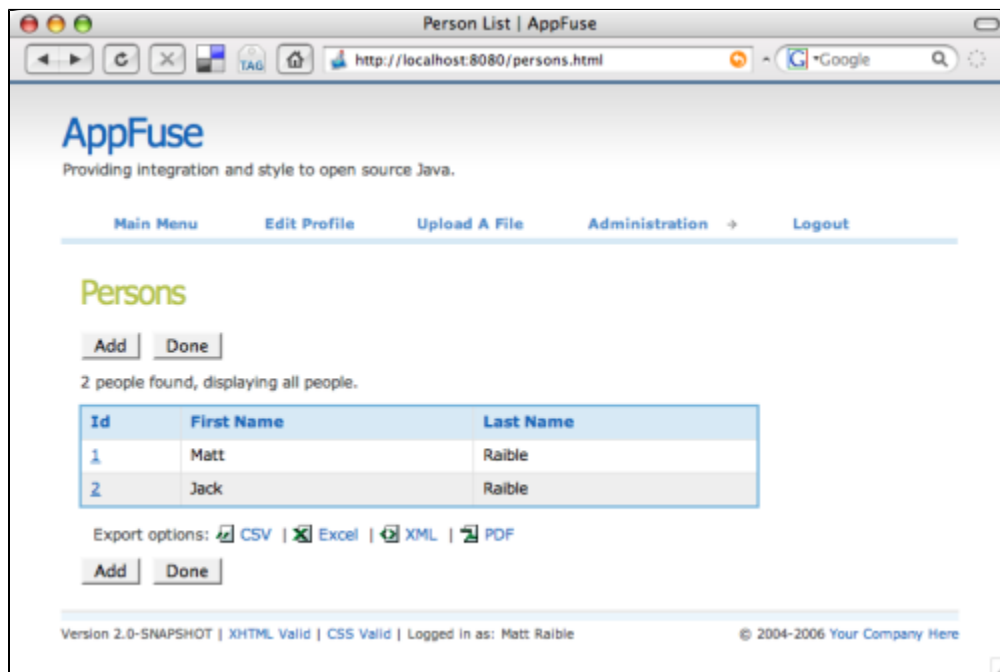
```
# -- person form --
person.id=Id
person.firstName=First Name
person.lastName=Last Name

person.added=Person has been added successfully.
person.updated=Person has been updated successfully.
person.deleted=Person has been deleted successfully.

# -- person list page --
personList.title=Person List
personList.heading=Persons

# -- person detail page --
personDetail.title=Person Detail
personDetail.heading=Person Information
```

Run **mvn jetty:run** and open <http://localhost:8080/persons> in your browser. Login with **admin/admin** and you should see a screen similar to the figure below.



Security settings for AppFuse specify that most url-patterns should be protected (except for /signup and /passwordHint). This guarantees that clients must go through an Action to get to a JSP (or at least the ones in *WEB-INF/pages*).



#### CSS Customization

If you want to customize the CSS for a particular page, you can add `<body id="pageName"/>` to the top of the file. This will be slurped up by SiteMesh and put into the final page. You can then customize your CSS on a page-by-page basis using something like the following:

```
body#pageName element.class { background-color: blue }
```

### Modify PersonActionTest and PersonAction for edit(), save() and delete() method

To create the detail screen, add `edit()`, `save()`, and `delete()` methods to the `PersonAction` class. Before doing this, create tests for these methods.

Open `src/test/java/**/webapp/action/PersonActionTest.java` and add test methods for edit, save, and delete operations:

```

@Test
public void testEdit() throws Exception {
    log.debug("testing edit...");
    action.setId(1L);
    assertNull(action.getPerson());
    assertEquals("success", action.edit());
    assertNotNull(action.getPerson());
    assertFalse(action.hasActionErrors());
}

@Test
public void testSave() throws Exception {
    MockHttpServletRequest request = new MockHttpServletRequest();
    ServletActionContext.setRequest(request);
    action.setId(1L);
    assertEquals("success", action.edit());
    assertNotNull(action.getPerson());

    // update last name and save
    action.getPerson().setLastName("Updated Last Name");
    assertEquals("input", action.save());
    assertEquals("Updated Last Name", action.getPerson().getLastName());
    assertFalse(action.hasActionErrors());
    assertFalse(action.hasFieldErrors());
    assertNotNull(request.getSession().getAttribute("messages"));
}

@Test
public void testRemove() throws Exception {
    MockHttpServletRequest request = new MockHttpServletRequest();
    ServletActionContext.setRequest(request);
    action.setDelete("");
    Person person = new Person();
    person.setId(2L);
    action.setPerson(person);
    assertEquals("success", action.delete());
    assertNotNull(request.getSession().getAttribute("messages"));
}

```

This class will not compile yet because you need to update your `src/main/java/**/action/PersonAction.java` class. The cancel and delete properties capture the click of the Cancel and Delete buttons. The `execute()` method routes the different actions on the form to the appropriate method.

```

private Person person;
private Long id;

public void setId(Long id) {
    this.id = id;
}

public Person getPerson() {
    return person;
}

public void setPerson(Person person) {
    this.person = person;
}

public String delete() {
    personManager.remove(person.getId());
    saveMessage(getText("person.deleted"));

    return SUCCESS;
}

public String edit() {
    if (id != null) {
        person = personManager.get(id);
    } else {
        person = new Person();
    }

    return SUCCESS;
}

public String save() throws Exception {
    if (cancel != null) {
        return CANCEL;
    }

    if (delete != null) {
        return delete();
    }

    boolean isNew = (person.getId() == null);

    person = personManager.save(person);

    String key = (isNew) ? "person.added" : "person.updated";
    saveMessage(getText(key));

    if (!isNew) {
        return INPUT;
    } else {
        return SUCCESS;
    }
}

```

If you look at your `PersonActionTest`, all the tests depend on having a record with `id=1` in the database (and `testRemove` depends on `id=2`), so let's add those records to our sample data file (`src/test/resources/sample-data.xml`). Adding it at the bottom should work - order is not important since it (currently) does not relate to any other tables. If you already have this table, make sure the 2nd record exists so `testRemove()` doesn't fail.

```

<table name='person'>
  <column>id</column>
  <column>first_name</column>
  <column>last_name</column>
  <row>
    <value>1</value>
    <value>Matt</value>
    <value>Raible</value>
  </row>
  <row>
    <value>2</value>
    <value>Bob</value>
    <value>Johnson</value>
  </row>
</table>

```

DbUnit loads this file before you run any tests, so this record will be available to your Action test.

Save all your files and run the tests in `PersonActionTest` using the command `mvn test -Dtest=PersonActionTest`.

**BUILD SUCCESSFUL**  
Total time: 13 seconds

### Create `personForm.jsp` to edit a person's information

Create a `src/main/webapp/WEB-INF/pages/personForm.jsp` page to display the form:

```

<%@ include file="/common/taglibs.jsp"%>

<head>
  <title><fmt:message key="personDetail.title"/></title>
  <meta name="heading" content="<fmt:message key='personDetail.heading' />" />
</head>

<s:form id="personForm" action="savePerson" method="post" validate="true">
  <li style="display: none">
    <s:hidden key="person.id" />
  </li>
  <s:textfield key="person.firstName" required="false" cssClass="text medium" />
  <s:textfield key="person.lastName" required="false" cssClass="text medium" />

  <li class="buttonBar bottom">
    <s:submit cssClass="button" method="save" key="button.save" theme="simple" />
    <c:if test="${not empty person.id}">
      <s:submit cssClass="button" method="delete" key="button.delete"
        onclick="return confirmDelete('Person');" theme="simple" />
    </c:if>
    <s:submit cssClass="button" method="cancel" key="button.cancel" theme="simple" />
  </li>
</s:form>

<script type="text/javascript">
  Form.focusFirstElement("${personForm}");
</script>

```

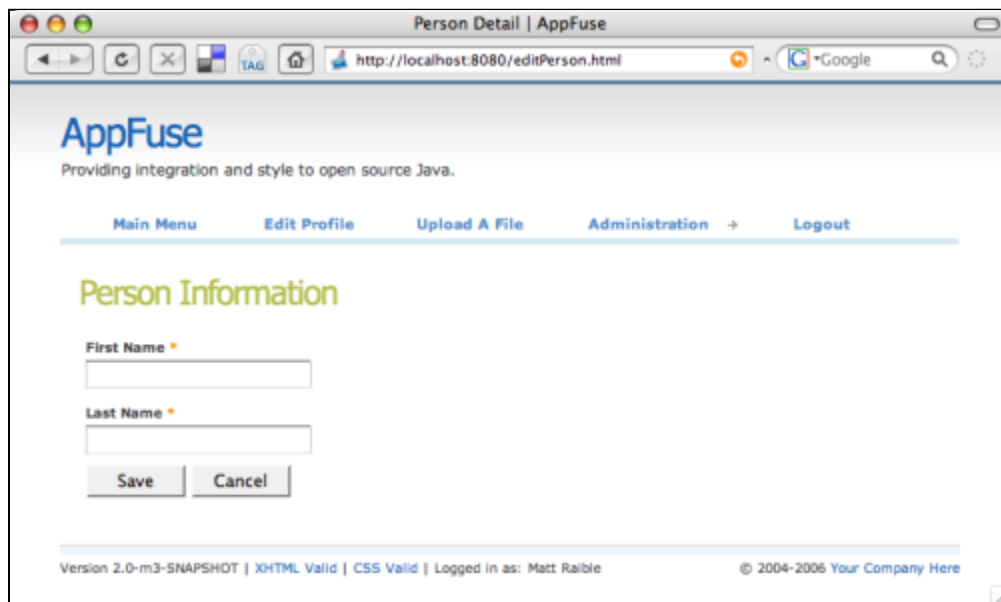
Struts reduces the amount of HTML you have to write for a form. The `<s:form>` tag writes the `<form>` and structure tags for you. The `<s:textfield>` tag writes the whole row, including the `<ul>` and `<li>` tags to hold the input field's label.

Next, update the `src/main/resources/struts.xml` file to include the "editPerson" and "savePerson" actions.

```
<action name="editPerson" class="org.appfuse.tutorial.webapp.action.PersonAction" method="edit">
  <result>/WEB-INF/pages/personForm.jsp</result>
  <result name="error">/WEB-INF/pages/personList.jsp</result>
</action>

<action name="savePerson" class="org.appfuse.tutorial.webapp.action.PersonAction" method="save">
  <result name="input">/WEB-INF/pages/personForm.jsp</result>
  <result name="cancel" type="redirectAction">persons</result>
  <result name="delete" type="redirectAction">persons</result>
  <result name="success" type="redirectAction">persons</result>
</action>
```

Run `mvn jetty:run`, open your browser to <http://localhost:8080/persons>, and click on the Add button.



Fill in the first name and last name fields and click the Save button. This should route you to the list screen, where a success message flashes and the new person displays in the list.





### Displaying success messages

The `src/main/webapp/common/messages.jsp` file in AppFuse renders the success message in this screen. This file is included in `decorators/default.jsp`. It also handles displaying validation errors:

```

<% if (request.getAttribute("struts.valueStack") != null) { %>
<!-- ActionError Messages - usually set in Actions --%>
<s:if test="hasActionErrors()">
  <div class="error" id="errorMessages">
    <s:iterator value="actionErrors">
      "
        alt="<fmt:message key="icon.warning"/>" class="icon" />
      <s:property/><br />
    </s:iterator>
  </div>
</s:if>

<!-- FieldError Messages - usually set by validation rules --%>
<s:if test="hasFieldErrors()">
  <div class="error" id="errorMessages">
    <s:iterator value="fieldErrors">
      <s:iterator value="value">
        "
          alt="<fmt:message key="icon.warning"/>" class="icon" />
        <s:property/><br />
      </s:iterator>
    </s:iterator>
  </div>
</s:if>
<% } %>

<!-- Success Messages --%>
<c:if test="{not empty messages}">
  <div class="message" id="successMessages">
    <c:forEach var="msg" items="{messages}">
      "
        alt="<fmt:message key="icon.information"/>" class="icon" />
      <c:out value="{msg}" /><br />
    </c:forEach>
  </div>
  <c:remove var="messages" scope="session"/>
</c:if>

<!-- Error Messages (on JSPs, not through Struts --%>
<c:if test="{not empty errors}">
  <div class="error" id="errorMessages">
    <c:forEach var="error" items="{errors}">
      "
        alt="<fmt:message key="icon.warning"/>" class="icon" />
      <c:out value="{error}" /><br />
    </c:forEach>
  </div>
  <c:remove var="errors" scope="session"/>
</c:if>

```

## Configure Validation

Struts 2 allows two types of validation: per-action and model-based. Since you likely want the same rules applied for the person object across different actions, this tutorial will use model-based.

Create a new file named `Person-validation.xml` in the `src/main/resources/**/model` directory (you'll need to create this directory). It should contain the following XML:

```

<!DOCTYPE validators PUBLIC "-//OpenSymphony Group//XWork Validator 1.0.2//EN"
"http://www.opensymphony.com/xwork/xwork-validator-1.0.2.dtd">
<validators>
  <field name="person.firstName">
    <field-validator type="requiredstring">
      <message key="errors.required"/>
    </field-validator>
  </field>
  <field name="person.lastName">
    <field-validator type="requiredstring">
      <message key="errors.required"/>
    </field-validator>
  </field>
</validators>

```

The "errors.message" key in errors\*.properties (listed below) will use the field's "name" attribute to do internationalization. You can also give the <message> element a body if you don't need i18n.

```
errors.required=${getText(fieldName)} is a required field.
```

Now you need to configure PersonAction to know about [visitor validation](#). To do this, create a PersonAction-validation.xml file in `src/main/resources/**/webapp/action` (you'll need to create this directory). Fill it with the following XML:

```

<!DOCTYPE validators PUBLIC "-//OpenSymphony Group//XWork Validator 1.0.2//EN"
"http://www.opensymphony.com/xwork/xwork-validator-1.0.2.dtd">
<validators>
  <field name="person">
    <field-validator type="visitor">
      <param name="appendPrefix">false</param>
      <message/>
    </field-validator>
  </field>
</validators>

```



Unfortunately, Struts doesn't have a transparent mechanism for reading from the `Person-validation.xml` file and marking fields as required on the UI. AppFuse's Spring MVC implementation use a `LabelTag` that makes this possible, but it also both use Commons Validator. It is my hope to someday provide this same functionality for Struts. In the meantime, the JSP tags "required" attribute has nothing to with the validation rules you specify. Rather, they simply add an asterisk to the label with no further functionality.



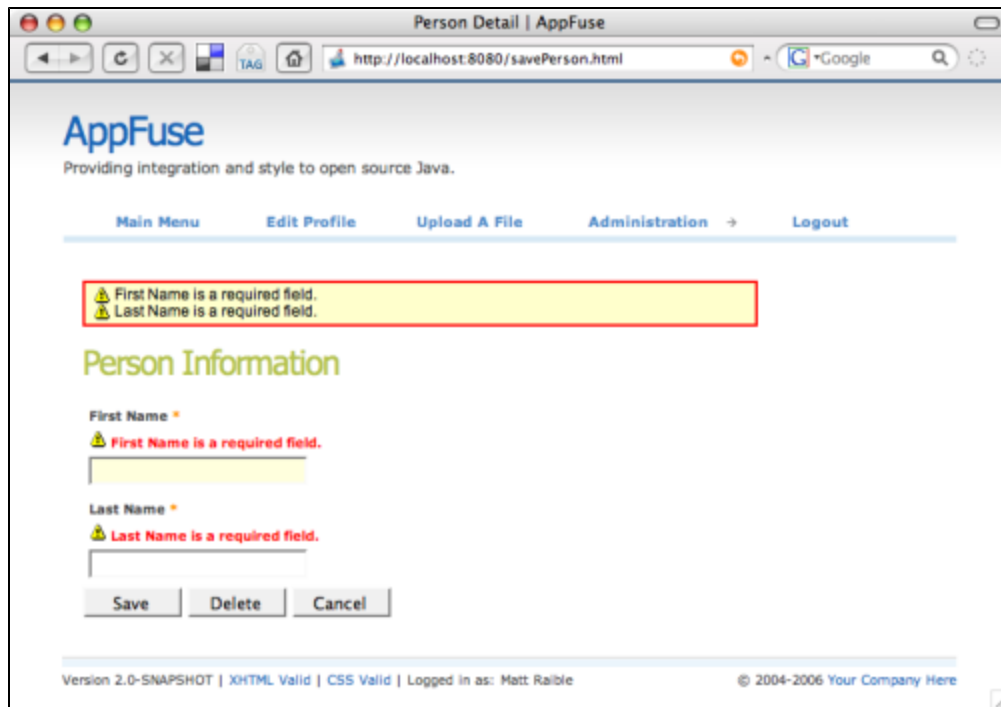
#### Client-side validation

Client-side validation of model-based validation rules [doesn't work](#) with the Struts setup that AppFuse uses. Furthermore, I believe that Struts's client-side validation needs some additional features, namely: [allow cancelling](#) and [showing all errors in one dialog](#). Because of this, only server-side validation works in Struts+AppFuse. If you'd like, you can [read more](#) about my frustrations with client-side validation.

As a workaround, you can use per-action validation. Just copy the `Person-validation.xml` file to the "webapp.action" package and rename it to `PersonAction-validation.xml`.

Struts' validation interceptor is enabled by default, so you don't need to configure anything for validation to work.

After saving all your files and running `mvn jetty:run`, validation should kick in when you try to save this form. To test, go to <http://localhost:8080/editPerson?id=1>. If you erase the values in the firstName and lastName fields and click the Save button, you should see the following:



AppFuse is configured so that methods cancel, execute, delete, edit, list, and start are not validated. This allows you to go back from a form with errors on (like above) by pressing the Cancel button.

### Create a Canoo WebTest to test browser-like actions

The next (optional) step in this tutorial is to create a [Canoo WebTest](#) to test the JSPs. This step is optional, because you can run the same tests manually through your browser. Regardless, it's a good idea to automate as much of your testing as possible.

You can use the following URLs to test the different actions for adding, editing and saving a user.

- Add - <http://localhost:8080/editPerson>.
- Edit - <http://localhost:8080/editPerson?id=1>.
- Delete - Use the edit link above and click on the Delete button.
- Save - Click [edit](#) and then click the Save button.



#### WebTest Recorder

There is a [WebTest Recorder](#) Firefox plugin that allows you to record your tests, rather than manually writing them.

Canoo tests are pretty slick in that they're simply configured in an XML file. To add tests for add, edit, save and delete, open `src/test/resources/web-tests.xml` and add the following XML. You'll notice that this fragment has a target named "PersonTests" that runs all the related tests.

```
<!-- runs person-related tests -->
<target name="PersonTests"
  depends="SearchPeople,EditPerson,SavePerson,AddPerson,DeletePerson"
  description="Call and executes all person test cases (targets)">
  <echo>Successfully ran all Person UI tests!</echo>
</target>

<!-- Verify the people list screen displays without errors -->
<target name="SearchPeople" description="Tests search for and displaying all people">
  <webtest name="searchPeople">
    &config;
    <steps>
      &login;
      <invoke description="click View People link" url="/persons.html"/>
      <verifytitle description="we should see the personList title"
        text=".*${personList.title}.*" regex="true"/>
    </steps>
  </webtest>
</target>
```

```

    </webtest>
</target>

<!-- Verify the edit person screen displays without errors -->
<target name="EditPerson"
  description="Tests editing an existing Person's information">
  <webtest name="editPerson">
    &config;
    <steps>
      &login;
      <invoke description="click Edit Person link" url="/editPerson.html?id=1"/>
      <verifytitle description="we should see the personDetail title"
        text=".*${personDetail.title}.*" regex="true"/>
    </steps>
  </webtest>
</target>

<!-- Edit a person and then save -->
<target name="SavePerson"
  description="Tests editing and saving a user">
  <webtest name="savePerson">
    &config;
    <steps>
      &login;
      <invoke description="click Edit Person link" url="/editPerson.html?id=1"/>
      <verifytitle description="we should see the personDetail title"
        text=".*${personDetail.title}.*" regex="true"/>
      <setinputfield description="set lastName" name="person.lastName" value="Canoo"/>
      <clickbutton label="${button.save}" description="Click Save"/>
      <verifytitle description="Page re-appears if save successful"
        text=".*${personDetail.title}.*" regex="true"/>
      <verifytext description="verify success message" text="${person.updated}"/>
    </steps>
  </webtest>
</target>

<!-- Add a new Person -->
<target name="AddPerson"
  description="Adds a new Person">
  <webtest name="addPerson">
    &config;
    <steps>
      &login;
      <invoke description="click Add Button" url="/editPerson.html"/>
      <verifytitle description="we should see the personDetail title"
        text=".*${personDetail.title}.*" regex="true"/>
      <setinputfield description="set firstName" name="person.firstName" value="Abbie"/>
      <setinputfield description="set lastName" name="person.lastName" value="Raible"/>
      <clickbutton label="${button.save}" description="Click button 'Save'"/>
      <verifytitle description="Person List appears if save successful"
        text=".*${personList.title}.*" regex="true"/>
      <verifytext description="verify success message" text="${person.added}"/>
    </steps>
  </webtest>
</target>

<!-- Delete existing person -->
<target name="DeletePerson"
  description="Deletes existing Person">
  <webtest name="deletePerson">
    &config;
    <steps>
      &login;
      <invoke description="click Edit Person link" url="/editPerson.html?id=1"/>
      <prepareDialogResponse description="Confirm delete" dialogType="confirm" response="true"/>
      <clickbutton label="${button.delete}" description="Click button 'Delete'"/>
      <verifyNoDialogResponses/>
      <verifytitle description="display Person List" text=".*${personList.title}.*" regex="true"
/>

      <verifytext description="verify success message" text="${person.deleted}"/>

```

</steps>

```

    </webtest>
</target>

```

After adding this, you should be able to run **mvn verify** and have these tests execute. If this command results in "BUILD SUCCESSFUL" - nice work!

To include the PersonTests when all Canoo tests are run, add it as a dependency to the "run-all-tests" target in *src/test/resources/web-test.xml*.

```

<target name="run-all-tests"
  depends=
    "Login,Logout,PasswordHint,Signup,UserTests,StaticPages,WebServices,DWR,FileUpload,PersonTests"
  description="Call and executes all test cases (targets)"/>

```

## Add link to menu

The last step is to make the list, add, edit and delete functions visible to the user. The simplest way is to add a new link to the list of links in *src/main/webapp/WEB-INF/pages/mainMenu.jsp*.

```

<li>
  <a href="<c:url value="/persons"/>"><fmt:message key="menu.viewPeople"/></a>
</li>

```

Where `menu.viewPeople` is an entry in *src/main/resources/ApplicationResources.properties*.

```

menu.viewPeople=View People

```

The other (more likely) alternative is that you'll want to add it to the menu. To do this, add the following to *src/main/webapp/WEB-INF/menu-config.xml*:

```

<Menu name="PersonMenu" title="menu.viewPeople" page="/persons"/>

```

Make sure the above XML is inside the `<Menus>` tag, but not within another `<Menu>`. Then open *src/main/webapp/common/menu.jsp* and add the following code to it:

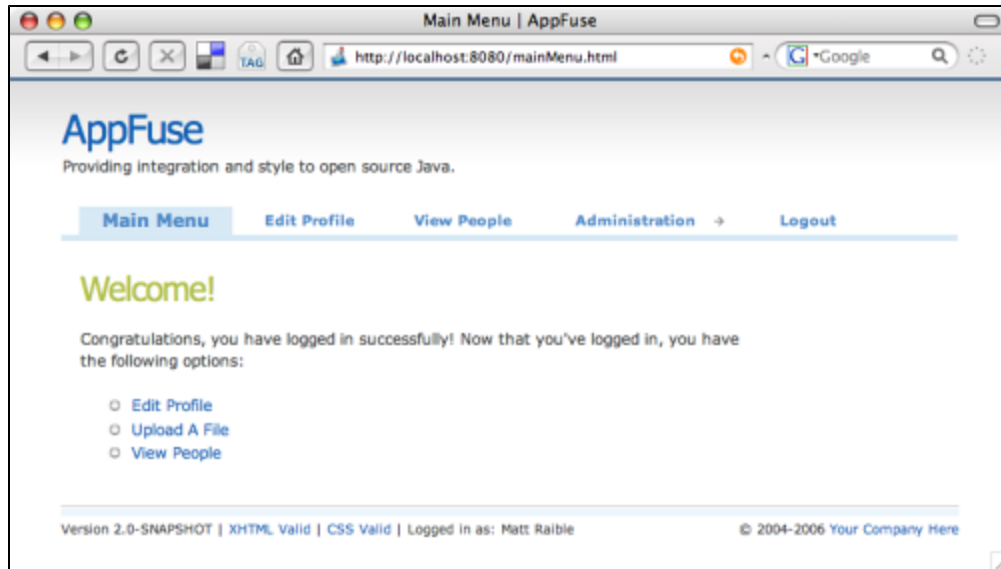
```

<%@ include file="/common/taglibs.jsp"%>

<menu:useMenuDisplay name="Velocity" config="cssHorizontalMenu.vm" permissions="rolesAdapter">
<ul id="primary-nav" class="menuList">
  <li class="pad">&nbsp;</li>
  <c:if test="${empty pageContext.request.remoteUser}">
  <li><a href="<c:url value="/login"/>" class="current">
    <fmt:message key="login.title"/></a></li>
  </c:if>
  <menu:displayMenu name="MainMenu"/>
  <menu:displayMenu name="UserMenu"/>
  <menu:displayMenu name="PersonMenu"/>
  <menu:displayMenu name="AdminMenu"/>
  <menu:displayMenu name="Logout"/>
</ul>
</menu:useMenuDisplay>

```

Now if you run **mvn jetty:run** and go to <http://localhost:8080/mainMenu>, you should see something like the screenshot below.



Notice that there is a new link in your main screen (from mainMenu.jsp) and on the top in your menu bar (from menu.jsp).

#### That's it!

You've completed the full lifecycle of developing a set of master-detail pages with AppFuse and Struts 2 - **Congratulations!**

Because it's so much fun to watch tests fly by and success happen, run all your tests again using **mvn install**.

#### Happy Day!

**BUILD SUCCESSFUL**

Total time: 53 seconds

## Using Tapestry

### About this Tutorial

This tutorial will show you how to create master/detail screens with [Tapestry 5](#). The list (master) screen will have the ability to sort columns, as well as page 25 records at a time. The form (detail) screen will use a nifty CSS form layout (courtesy of [Wufoo](#)). You will also configure client and server-side validation to improve your users' experience.



#### IntelliJ IDEA Rocks

We highly recommend using IntelliJ IDEA when developing web applications in Java. Not only is its Java and JUnit support fantastic, but it has excellent CSS and JavaScript support. [Using JRebel with IDEA](#) is likely to provide you with the most pleasant Java development experiences you've ever had.



This tutorial assumes you've created a project with the **appfuse-basic-tapestry** archetype and have already completed the [Persistence](#) and [Services](#) tutorials. If you're using the **appfuse-modular-tapestry** archetype, please morph your mind into using the *web* module as the root directory. If you created your project with a different web framework than Tapestry, you're likely to be confused and nothing will work in this tutorial. 😊

### Table of Contents

1. [Introduction to Tapestry](#)
2. [Create a PersonListTest](#)
3. [Create a PersonList class that will fetch people](#)
4. [Create PersonList.html to show search results](#)
5. [Create a PersonFormTest and PersonForm for edit\(\), save\(\) and delete\(\) methods](#)
6. [Add an edit listener to PersonList.java](#)
7. [Create PersonForm.html to edit a person](#)
8. [Configure Validation](#)
9. [Create a Canoo WebTest to test browser-like actions](#)
10. [Add link to menu](#)



tapestry



### Source Code

The code for this tutorial is located in the "tutorial-tapestry" module of the [appfuse-demos](#) project on Google Code. Use the following command to check it out from Subversion:

```
svn checkout http://appfuse-demos.googlecode.com/svn/trunk/tutorial-tapestry
```

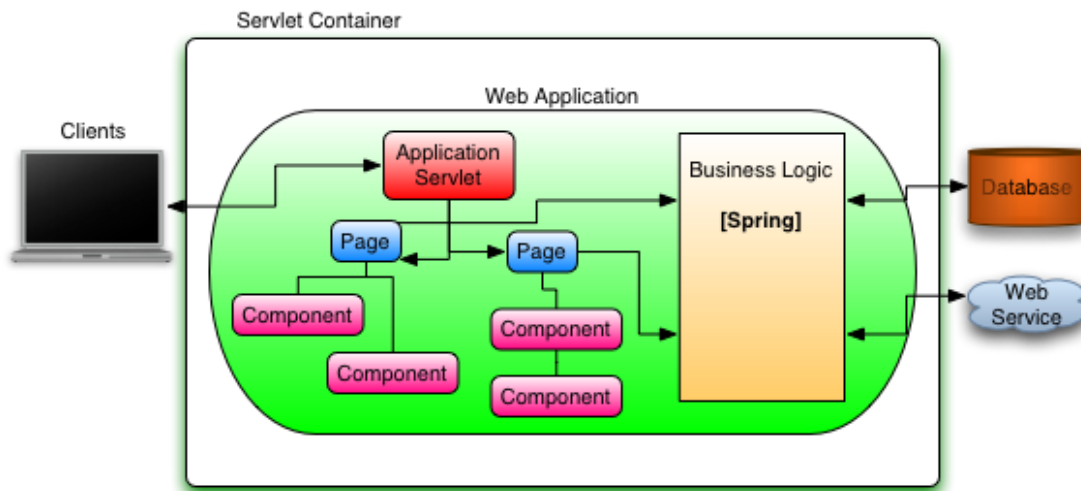
## Introduction to Tapestry

Tapestry is a component-based framework for developing web applications. Unlike many other Java web frameworks, Tapestry uses a component object model similar to traditional GUI frameworks. According to Howard Lewis Ship, the founder of Tapestry:

*A component is an object that fits into an overall framework; the responsibilities of the component are defined by the design and structure of the framework. A component is a component, and not simply an object, when it follows the rules of the framework. These rules can take the form of classes to inherit from, naming conventions (for classes or methods) to follow, or interfaces to implement. Components can be used within the context of the framework. The framework will act as a container for the component, controlling when the component is instantiated and initialized, and dictating when the methods of the component are invoked.*

– Lewis Ship, Howard. *Tapestry in Action*. Greenwich, CT: Manning Publications Co., 2004.

The figure below shows how Tapestry fits into a web application's architecture:



Tapestry's component model allows you to have a very high level of reuse within and between projects. You can package components in JAR files and distribute them among teams and developers.

Tapestry tries to hide the Servlet API from developers. Learning Tapestry is often characterized as an "unlearning" process. GUI programmers typically have an easier time adjusting to the way things work in Tapestry. Tapestry operates in terms of objects, methods and properties, rather than URLs and query parameters. All of the URL building, page dispatching and method invocation happens transparently.

Other benefits of Tapestry include line-precise error reporting and easy-to-use HTML templates. While other frameworks use external templating systems, Tapestry has its own templating system. Tapestry templates are often HTML files, but they can also be WML or XML. You can hook into these templates by using Tapestry-specific attributes on existing HTML elements.

## Create a PersonListTest

This tutorial shows you how to create a Tapestry application using test-first development. You will use JUnit and a `BasePageTestCase` that instantiates page classes for you.

Create a `PersonListTest.java` class in `src/test/java/**/webapp/pages`:



```

package org.appfuse.tutorial.webapp.pages;

import org.apache.tapestry5.dom.Element;
import org.apache.tapestry5.dom.Node;
import org.junit.Test;

import java.util.List;
import java.util.ResourceBundle;

import static org.junit.Assert.*;

public class PersonListTest extends BasePageTestCase {
    @Test
    public void testList() {
        doc = tester.renderPage("personList");
        assertNotNull(doc.getElementById("personList"));
        assertTrue(doc.getElementById("personList").find("tbody").getChildren().size() >= 2);
    }
}

```

This class will not compile until you create the `PersonList` class.

### Create a `PersonList` that will fetch people

Create a `PersonList.java` file in `src/main/java/**/webapp/pages`:

```

package org.appfuse.tutorial.webapp.pages;

import org.apache.tapestry5.annotations.Component;
import org.apache.tapestry5.annotations.InjectPage;
import org.apache.tapestry5.annotations.Property;
import org.apache.tapestry5.annotations.Service;
import org.apache.tapestry5.corelib.components.EventLink;
import org.apache.tapestry5.ioc.annotations.Inject;
import org.appfuse.service.GenericManager;
import org.appfuse.tutorial.model.Person;
import org.slf4j.Logger;

import java.util.List;

public class PersonList extends BasePage {

    @Inject
    @Service("personManager")
    private GenericManager<Person, Long> personManager;

    @Property
    private Person person;

    @Component(parameters = {"event=add"})
    private EventLink addTop, addBottom;

    @Component(parameters = {"event=done"})
    private EventLink doneTop, doneBottom;

    public List<Person> getPersons() {
        return personManager.getAll();
    }

    Object onDone() {
        return MainMenu.class;
    }
}

```

Since Tapestry's PageTester class requires your template exists before tests will pass, please continue to the next step before running your test.

### Create PersonsList.tml to show search results

Create a `_src/main/webapp/PersonList.tml` page to display the list of people.

```
<t:layout title="message:personList.title"
  heading="message:personList.heading" menu="literal:PersonMenu"
  xmlns:t="http://tapestry.apache.org/schema/tapestry_5_0_0.xsd">

  <t:messagebanner t:id="message" type="type"/>
  <p>
    <t:eventlink t:id="addTop">
      <input type="button" class="button" value="${message:button.add}"/>
    </t:eventlink>
    <t:eventlink t:id="doneTop">
      <input type="button" class="button" value="${message:button.done}"/>
    </t:eventlink>
  </p>

  <t:grid source="persons" row="person" id="personList" class="table"/>

  <p>
    <t:eventlink t:id="addBottom">
      <input type="button" class="button" value="${message:button.add}"/>
    </t:eventlink>
    <t:eventlink t:id="doneBottom">
      <input type="button" class="button" value="${message:button.done}"/>
    </t:eventlink>
  </p>

  <script type="text/javascript">
    highlightTableRows("personList");
  </script>
</t:layout>
```



Now if you run `mvn test -Dtest=PersonListTest`, your test should pass.

**Nice!**

**BUILD SUCCESSFUL**

Total time: 17 seconds

Open `src/main/resources/ApplicationResources.properties` and add 18 keys/values for the various "person" properties:

```
# -- person form --
person.id=Id
person.firstName=First Name
person.lastName=Last Name

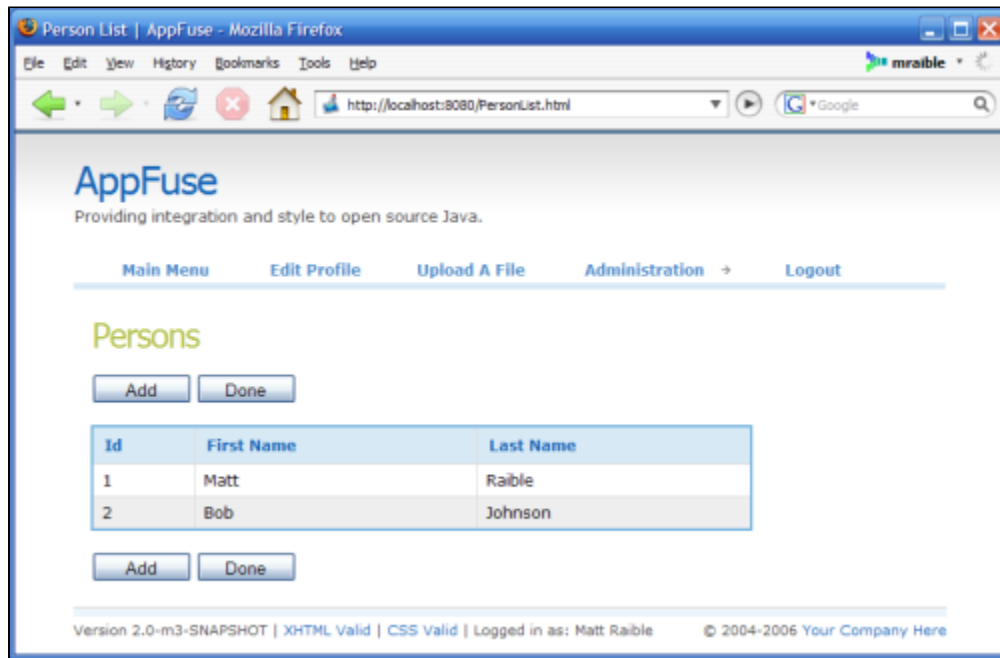
person.added=Person has been added successfully.
person.updated=Person has been updated successfully.
person.deleted=Person has been deleted successfully.

# -- person list page --
personList.title=Person List
personList.heading=Persons

# -- person detail page --
personDetail.title=Person Detail
personDetail.heading=Person Information
```

Run `mvn jetty:run` and open <http://localhost:8080/personlist> in your browser. Login with `admin/admin` and you should see a screen similar to the

figure below.



Security settings for AppFuse specify that most url-patterns should be protected (except for /signup and /passwordhint). This guarantees that clients must go through Tapestry's TapestrySpringFilter to get to view pages.



#### CSS Customization

If you want to customize the CSS for a particular page, you can add `<body id="pageName"/>` to the top of the file. This will be slurped up by SiteMesh and put into the final page. You can then customize your CSS on a page-by-page basis using something like the following:

```
body#pageName element.class { background-color: blue }
```

### Create a PersonFormTest and PersonForm for edit(), save() and delete() methods

To start creating the detail screen, create a `PersonFormTest.java` class in `src/test/java/**/webapp/pages:`

```
package org.appfuse.tutorial.webapp.pages;

import org.apache.tapestry5.dom.Element;
import org.apache.tapestry5.dom.Node;
import static org.junit.Assert.*;
import org.junit.Test;

import java.util.List;
import java.util.ResourceBundle;

public class PersonFormTest extends BasePageTestCase {

    @Test
    public void testCancel() throws Exception {
        doc = tester.renderPage("personList");
        Element table = doc.getElementById("personList");
        List<Node> rows = table.find("tbody").getChildren();
        String id = ((Element) rows.get(0)).find("td/a").getChildMarkup().trim();

        Element editLink = table.getElementById("person-" + id);
        doc = tester.clickLink(editLink);
    }
}
```

```

        Element cancelButton = doc.getElementById("cancel");
        doc = tester.clickSubmit(cancelButton, fieldValues);

        ResourceBundle rb = ResourceBundle.getBundle(MESSAGES);

        assertTrue(doc.toString().contains("<title>" +
            rb.getString("personList.title")));
    }

    @Test
    public void testSave() throws Exception {
        doc = tester.renderPage("personForm");

        Element form = doc.getElementById("personForm");
        assertNotNull(form);

        // enter all required fields
        fieldValues.put("firstName", "Jack");
        fieldValues.put("lastName", "Raible");

        doc = tester.submitForm(form, fieldValues);

        Element errors = doc.getElementById("errorMessages");

        if (errors != null) {
            System.out.println(errors);
        }

        assertNull(doc.getElementById("errorMessages"));

        Element successMessages = doc.getElementById("successMessages");
        assertNotNull(successMessages);
        assertTrue(successMessages.toString().contains("added successfully"));
        Element table = doc.getElementById("personList");
        assertNotNull(table);
    }

    @Test
    public void testRemove() throws Exception {
        doc = tester.renderPage("personList");
        Element table = doc.getElementById("personList");
        List<Node> rows = table.find("tbody").getChildren();
        String id = ((Element) rows.get(1)).find("td/a").getChildMarkup().trim();

        Element editLink = table.getElementById("person-" + id);
        doc = tester.clickLink(editLink);

        Element deleteButton = doc.getElementById("delete");
        doc = tester.clickSubmit(deleteButton, fieldValues);
        assertTrue(doc.toString().contains("deleted successfully"));
    }

```

```
}  
}
```

Nothing will compile at this point; you need to create the `PersonForm` that you're referring to in this test.

In `src/main/java/**/webapp/pages`, create a `PersonForm.java` class that extends AppFuse's `BasePage`. Populate it with the following code:

```
package org.appfuse.tutorial.webapp.pages;  
  
import org.apache.tapestry5.annotations.Component;  
import org.apache.tapestry5.annotations.InjectPage;  
import org.apache.tapestry5.annotations.Persist;  
import org.apache.tapestry5.annotations.Service;  
import org.apache.tapestry5.corelib.components.Form;  
import org.apache.tapestry5.ioc.annotations.Inject;  
import org.appfuse.service.GenericManager;  
import org.appfuse.tutorial.model.Person;  
import org.slf4j.Logger;  
  
public class PersonForm extends BasePage {  
    @Inject  
    private Logger log;  
  
    @Inject  
    @Service("personManager")  
    private GenericManager<Person, Long> personManager;  
  
    @Persist  
    private Person person;  
  
    public Person getPerson() {  
        return person;  
    }  
  
    /**  
     * Allows setting person object from another class (i.e. PersonList)  
     *  
     * @param person an initialized instance  
     */  
    public void setPerson(Person person) {  
        this.person = person;  
    }  
  
    @InjectPage  
    private PersonList personList;  
  
    @Component(id = "personForm")  
    private Form form;  
  
    private boolean cancel;  
    private boolean delete;  
  
    void onValidateForm() {  
        if (!delete && !cancel) {  
            // manually validate required fields or annotate the Person object  
            /*if (foo.getProperty() == null || user.getProperty().trim().equals("")) {  
                form.recordError("Property is a required field.");  
            }*/  
        }  
    }  
  
    void onActivate(Long id) {  
        if (id != null) {  
            person = personManager.get(id);  
        }  
    }  
}
```

```
Object onSuccess() {
    if (delete) return onDelete();
    if (cancel) return onCancel();

    log.debug("Saving person...");

    boolean isNew = (getPerson().getId() == null);

    personManager.save(person);

    String key = (isNew) ? "person.added" : "person.updated";

    if (isNew) {
        personList.addInfo(key, true);
        return personList;
    } else {
        addInfo(key, true);
        return this;
    }
}

void onSelectedFromDelete() {
    log.debug("Deleting person...");
    delete = true;
}

void onSelectedFromCancel() {
    log.debug("Cancelling form...");
    cancel = true;
}

Object onDelete() {
    personManager.remove(person.getId());
    personList.addInfo("person.deleted", true);
    return personList;
}

Object onCancel() {
    return personList;
}
```

```
}
}
```

You might notice a number of keys in this file - "person.deleted", "person.added" and "person.updated". These are all keys that need to be in your i18n bundle (`ApplicationResources.properties`). You should've added these at the beginning of this tutorial.

If you look at your `PersonFormTest`, all the tests depend on having a record with `id=1` in the database (and `testRemove` depends on `id=2`), so let's add those records to our sample data file (`src/test/resources/sample-data.xml`). Adding it at the bottom should work - order is not important since it (currently) does not relate to any other tables. If you already have this table, make sure the 2nd record exists so `testRemove()` doesn't fail.

```
<table name='person'>
  <column>id</column>
  <column>first_name</column>
  <column>last_name</column>
  <row>
    <value>1</value>
    <value>Matt</value>
    <value>Raible</value>
  </row>
  <row>
    <value>2</value>
    <value>Bob</value>
    <value>Johnson</value>
  </row>
</table>
```

DbUnit loads this file before you run any tests, so these records will be available to your `PersonFormTest` class. Since Tapestry's `PageTester` class requires your template exists before tests will pass, please continue to the next step before running your test.

Save all your files and run the tests in `PersonFormTest` using the command **`mvn test -Dtest=PersonFormTest`**.

**BUILD SUCCESSFUL**  
Total time: 16 seconds

### Add an edit listener to `PersonList.java`

To allow users to click on the list screen to get to the edit screen, you need to add `onAdd()` and `onActionFromEdit()` methods to `PersonList.java`. Open `PersonListTest.java` and add the following `testEdit()` method:

```
@Test
public void testEdit() {
    doc = tester.renderPage("personList");

    Element table = doc.getElementById("personList");
    List<Node> rows = table.find("tbody").getChildren();
    String id = ((Element) rows.get(0)).find("td/a").getChildMarkup().trim();
    Element editLink = table.getElementById("person-" + id);
    doc = tester.clickLink(editLink);

    ResourceBundle rb = ResourceBundle.getBundle(MESSAGES);

    assertTrue(doc.toString().contains("<title>" +
        rb.getString("personDetail.title")));
}
```

Add the aforementioned methods to `PersonList.java`:

```

@Inject
private Logger log;

@InjectPage
private PersonForm form;

Object onAdd() {
    form.setPerson(new Person());
    return form;
}

Object onActionFromEdit(Long id) {
    log.debug("fetching person with id: {}", id);
    return form;
}

```

Then add `<t:parameter>` and `<t:pagelink>` elements to the grid component in `PersonList.tml`

```

<t:grid source="persons" row="person" id="personList" class="table">
    <t:parameter name="idCell">
        <t:pagelink page="personform" context="person.id" id="person-${person.id}">
            ${person.id}
        </t:pagelink>
    </t:parameter>
</t:grid>

```

Now you need to create the view template so you can edit a person's information.

### Create PersonForm.html to edit a person

Create a `src/main/webapp/PersonForm.tml` page to display the form:

```

<t:layout title="message:personDetail.title"
    heading="message:personDetail.heading" menu="literal:PersonMenu"
    xmlns:t="http://tapestry.apache.org/schema/tapestry_5_0_0.xsd">

    <t:messagebanner t:id="message" type="type"/>

    <form t:id="personForm">
        <t:errors/>

        <div class="t-beaneditor">
            <t:beaneditor t:id="person" object="person" exclude="id"/>

            <div class="t-beaneditor-row" style="text-align: center">
                <input t:type="submit" t:id="save" id="save" value="message:button.save"/>
                <input t:type="submit" t:id="delete" id="delete" value="message:button.delete"
                    onclick="return confirmDelete('Person')"/>
                <input t:type="submit" t:id="cancel" id="cancel" value="message:button.cancel"/>
            </div>
        </div>
    </form>

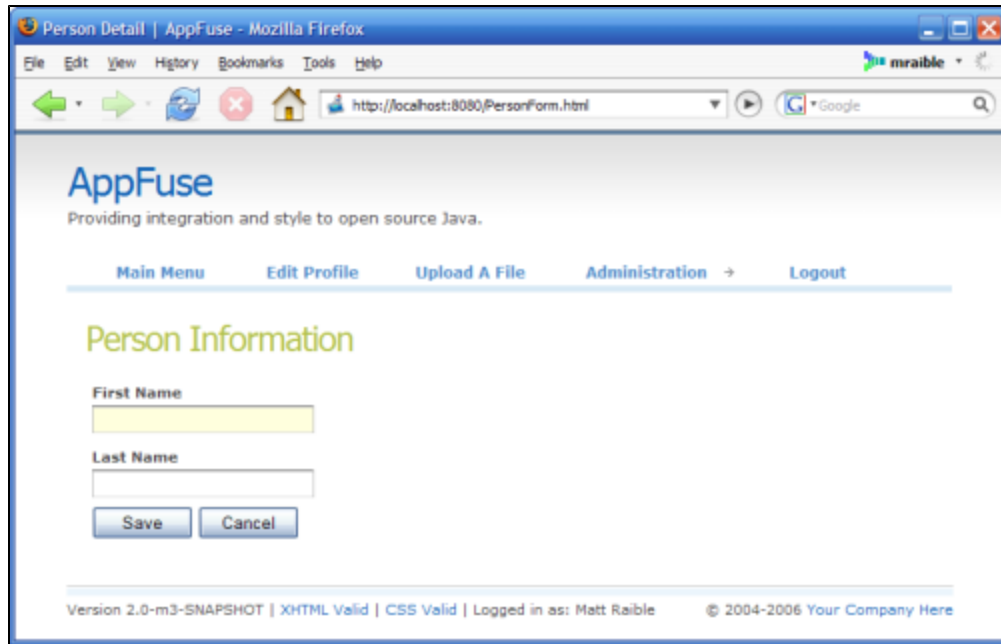
    <script type="text/javascript">
        Form.focusFirstElement($("#personForm"));
    </script>

</t:layout>

```

Run `mvn jetty:run`, open your browser to <http://localhost:8080/personlist>, and click on the **Add** button.





Fill in the first name and last name fields and click the Save button. This should route you to the list screen, where a success message flashes and the new person displays in the list.



#### Displaying success messages

AppFuse renders success and error messages using the following code that refers to the MessageBanner component in `src/main/java/**/webapp/components/MessageBanner.tml`.

```
<t:messagebanner t:id="message" type="type"/>
```

### Configure Validation

To enable server-side validation, you need to add the `PersonForm.java` object so you can specify validation information with annotations. Also, you'll want to add a `beforeRender()` method to make sure there's a `Person` object that can be populated.

```
@Component(id = "firstName", parameters = {"value=person.firstName", "validate=required"})
private TextField firstNameField;

@Component(id = "lastName", parameters = {"value=person.lastName", "validate=required"})
private TextField lastNameField;

void beginRender() {
    if (person == null) {
        person = new Person();
    }
}
```

Then replace the form in `PersonForm.tml` with the following:

```

<form t:id="personForm">
  <t:errors/>

  <ul>
    <li>
      <t:label class="desc" for="firstName">${message:person.firstName}</t:label>
      <input class="text medium" type="text" t:id="firstName"/>
    </li>
    <li>
      <t:label class="desc" for="lastName">${message:person.lastName}</t:label>
      <input class="text medium" type="text" t:id="lastName"/>
    </li>
    <li class="buttonBar bottom">
      <input t:type="submit" t:id="save" id="save" value="message:button.save"/>
      <input t:type="submit" t:id="delete" id="delete" value="message:button.delete"
        onclick="return confirmDelete('Person')"/>
      <input t:type="submit" t:id="cancel" id="cancel" value="message:button.cancel"/>
    </li>
  </ul>
</form>

```

After saving all your files and running **mvn jetty:run**, validation should kick in when you try to save this form. To test, go to <http://localhost:8080/personform> and try to add a new user with no first or last name. You should see the following validation errors:

With Tapestry, client-side validation is enabled by default. You can turn it off by adding `clientValidation="false"` to the `<form>` tag in `PersonForm.tml`.

```

<form t:id="personForm" clientValidation="true">

```

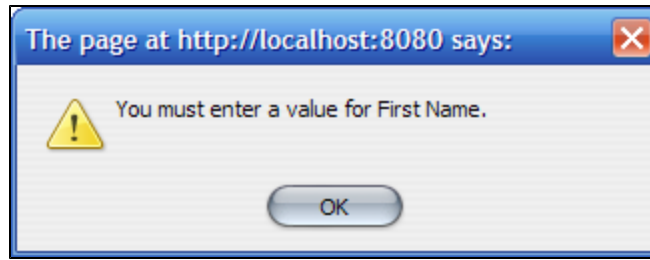
You'll also need to add `form.onsubmit = null` to the `onclick()` handlers of the Delete and Cancel buttons so client-side validation is disabled when they're clicked:

```

<li class="buttonBar bottom">
  <input t:type="submit" t:id="save" id="save" value="message:button.save"/>
  <input t:type="submit" t:id="delete" id="delete" value="message:button.delete"
    onclick="form.onsubmit = null; return confirmDelete('Person')"/>
  <input t:type="submit" t:id="cancel" id="cancel" value="message:button.cancel"
    onclick="form.onsubmit = null"/>
</li>

```

After saving all your files and running **mvn jetty:run**, client-side validation should kick in when you try to save this form. To test, go to <http://localhost:8080/personform> and try to add a new user with no first or last name. You should get the following JavaScript alert:



## Create a Canoo WebTest to test browser-like actions

The next (optional) step in this tutorial is to create a [Canoo WebTest](#) to test your UI. This step is optional, because you can run the same tests manually through your browser. Regardless, it's a good idea to automate as much of your testing as possible.

You can use the following URLs to test the different actions for adding, editing and saving a user.

- Add - <http://localhost:8080/personform>.
- Edit - Go to <http://localhost:8080/personlist> and click on an existing record.
- Delete - Use the edit link above and click on the Delete button.
- Save - Use the edit link above and click the Save button.



### WebTest Recorder

There is a [WebTest Recorder](#) Firefox plugin that allows you to record your tests, rather than manually writing them.

Canoo tests are pretty slick in that they're simply configured in an XML file. To add tests for add, edit, save and delete, open `src/test/resources/web-tests.xml` and add the following XML. You'll notice that this fragment has a target named **PersonTests** that runs all the related tests.

```
<!-- runs person-related tests -->
<target name="PersonTests" depends="SearchPersons,EditPerson,SavePerson,AddPerson,DeletePerson"
  description="Call and executes all person test cases (targets)">
  <echo>Successfully ran all Person UI tests!</echo>
</target>

<!-- Verify the persons list screen displays without errors -->
<target name="SearchPersons" description="Tests search for and displaying all persons">
  <webtest name="searchPersons">
    &config;
    <steps>
      &login;
      <invoke description="click View Person link" url="/personlist"/>
      <verifytitle description="we should see the personList title"
        text=".*${personList.title}.*" regex="true"/>
    </steps>
  </webtest>
</target>

<!-- Verify the edit person screen displays without errors -->
<target name="EditPerson" description="Tests editing an existing Person's information">
  <webtest name="editPerson">
    &config;
    <steps>
      &login;
      <invoke description="View Person List" url="/personlist"/>
      <clicklink description="click on first record in list" label="1"/>
      <verifytitle description="we should see the personDetail title"
        text=".*${personDetail.title}.*" regex="true"/>
    </steps>
  </webtest>
</target>

<!-- Edit a person and then save -->
<target name="SavePerson" description="Tests editing and saving a person">
  <webtest name="savePerson">
```

```

    &config;
    <steps>
        &login;
        <invoke description="View Person List" url="/personlist"/>
        <clicklink description="click on first record in list" label="1"/>
        <verifytitle description="we should see the personDetail title"
            text=".*${personDetail.title}.*" regex="true"/>

        <clickbutton label="${button.save}" description="Click Save"/>
        <verifytitle description="Page re-appears if save successful"
            text=".*${personDetail.title}.*" regex="true"/>
        <verifytext description="verify success message" text="${person.updated}"/>
    </steps>
</webtest>
</target>

<!-- Add a new Person -->
<target name="AddPerson" description="Adds a new Person">
    <webtest name="addPerson">
        &config;
        <steps>
            &login;
            <invoke description="Click Add button" url="/personform"/>
            <verifytitle description="we should see the personDetail title"
                text=".*${personDetail.title}.*" regex="true"/>

            <!-- enter required fields -->
            <setinputfield description="set firstName" name="firstName" value="Abbie"/>
            <setinputfield description="set lastName" name="lastName" value="Raible"/>

            <clickbutton label="${button.save}" description="Click button 'Save'"/>
            <verifytitle description="Person List appears if save successful"
                text=".*${personList.title}.*" regex="true"/>
            <verifytext description="verify success message" text="${person.added}"/>
        </steps>
    </webtest>
</target>

<!-- Delete existing person -->
<target name="DeletePerson" description="Deletes existing Person">
    <webtest name="deletePerson">
        &config;
        <steps>
            &login;
            <invoke description="View Person List" url="/personlist"/>
            <clicklink description="click on first record in list" label="2"/>
            <prepareDialogResponse description="Confirm delete" dialogType="confirm" response="true"/>
            <clickbutton label="${button.delete}" description="Click button 'Delete'"/>
            <verifyNoDialogResponses/>
            <verifytitle description="display Person List" text=".*${personList.title}.*" regex="true"
/>

            <verifytext description="verify success message" text="${person.deleted}"/>
        </steps>
    </webtest>
</target>

```

```

    </webtest>
</target>

```

To include the PersonTests when all Canoo tests are run, add it as a dependency to the "run-all-tests" target in *src/test/resources/web-test.xml*.

```

<target name="run-all-tests"
  depends=
    "Login,Logout,PasswordHint,Signup,UserTests,StaticPages,WebServices,DWR,FileUpload,PersonTests"
  description="Call and executes all test cases (targets)"/>

```

After adding this, you should be able to run **mvn verify** and have these tests execute. If this command results in "BUILD SUCCESSFUL" - nice work!

### Add link to menu

The last step is to make the list, add, edit and delete functions visible to the user. The simplest way is to add a new link to the list of links in *src/main/webapp/MainMenu.tml*.

```

<li>
  <a t:type="pagelink" page="PersonList">${message:menu.viewPeople}</a>
</li>

```

Where menu.viewPeople is an entry in *src/main/resources/ApplicationResources.properties* (which is auto-copied to *src/main/webapp/WEB-INF/app.properties*).

```

menu.viewPeople=View People

```

The other (more likely) alternative is that you'll want to add it to the menu. To do this, add the following to *src/main/webapp/WEB-INF/menu-config.xml*:

```

<Menu name="PersonMenu" title="menu.viewPeople" page="/personlist"/>

```

Make sure the above XML is inside the <Menus> tag, but not within another <Menu>. Then create *src/main/webapp/common/menu.jsp* and add the following code to it:

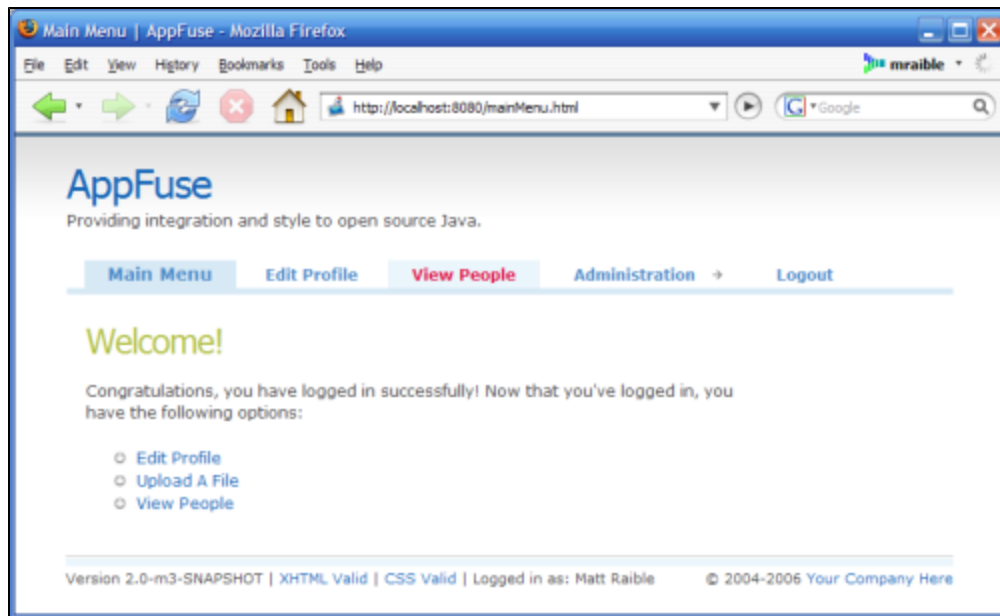
```

<%@ include file="/common/taglibs.jsp" %>

<menu:useMenuDisplayer name="Velocity" config="cssHorizontalMenu.vm" permissions="rolesAdapter">
  <ul id="primary-nav" class="menuList">
    <li class="pad">&nbsp;</li>
    <c:if test="${empty pageContext.request.remoteUser}">
      <li><a href="<c:url value="/login"/>" class="current"><fmt:message key="login.title"
    /></a></li>
    </c:if>
    <menu:displayMenu name="MainMenu"/>
    <menu:displayMenu name="UserMenu"/>
    <menu:displayMenu name="PersonMenu"/>
    <menu:displayMenu name="AdminMenu"/>
    <menu:displayMenu name="Logout"/>
  </ul>
</menu:useMenuDisplayer>

```

Now if you run **mvn jetty:run** and go to <http://localhost:8080/mainmenu>, you should see something like the screenshot below.



Notice that there is a new link in your main screen (from mainMenu.html) and on the top in your menu bar (from menu.jsp).

### **That's it!**

You've completed the full lifecycle of developing a set of master-detail pages with AppFuse and Tapestry 5 - **Congratulations!**

Because it's so much fun to watch tests fly by and success happen, run all your tests again using **mvn install**.

### **Happy Day!**

**BUILD SUCCESSFUL**

Total time: 1 minute 33 seconds