Java Developers Guide

Table of Contents

Tools 1

Git 1

Eclipse 1

Creating workspaces and projects 2

Installing Eclipse 2

Installing Eclipse add-ons 2

Importing existing preferences 4

Manually configuring preferences 4

WildFly 6

Installing WildFly 8.1 on Mac OS X 6

Installing WildFly 8.1 on CentOS 6

Setting up a PostgreSQL data source on WildFly 8.1 8

Enabling WildFly 8.1 on all network interfaces 9

Apache 10

Installing Apache on CentOS 10

Setting up Apache as a web server fronting JBoss 7 on CentOS 10

Enabling SSL support for Apache on CentOS 11

PostgreSQL 12

Using PostgreSQL with Mac OS X 12

Using PostgreSQL with Ubuntu Linux 13

Development Artifacts 14

JEE Web Application 14

Creating a new JEE web application 14

JEE Web Service 22

Creating a new JEE web service 22

# Tools

A number of development tools are employed to produce the various Java development artifacts. These tools include the Intellij and Eclipse Java IDEs, the Wildfly JEE application server and the PostgreSQL relational database management system. Designs, documentation and source code artifacts are managed using the Git version control system.

## Git

“Git is a distributed revision control system with an emphasis on speed. Git was initially designed and developed by Linus Torvalds for Linux kernel development. Every Git working directory is a full-fledged repository with complete history and full revision tracking capabilities, not dependent on network access or a central server.” -- Wikipedia.

## Eclipse

“Eclipse is a multi-language software development environment comprising an IDE and a plug-in system to extend it. It is written primarily in Java and can be used to develop applications in Java and, by means of the various plug-ins, in other languages as well, including C, C++, COBOL, Python, Perl, PHP, and others. The IDE is often called Eclipse ADT for Ada, Eclipse CDT for C, Eclipse JDT for Java and Eclipse PDT for PHP.   
  
Eclipse employs plug-ins in order to provide all of its functionality on top of (and including) the runtime system, in contrast to some other applications where functionality is typically hard coded. The runtime system of Eclipse is based on Equinox, an OSGi standard compliant implementation.

This plug-in mechanism is a lightweight software componentry framework. In addition to allowing Eclipse to be extended using other programming languages such as C and Python, the plug-in framework allows Eclipse to work with typesetting languages like LaTeX,[3] networking applications such as telnet, and database management systems. The plug-in architecture supports writing any desired extension to the environment, such as for configuration management. Java and CVS support is provided in the Eclipse SDK, with Subversion support provided by third-party plug-ins.

The key to the seamless integration (but not of seamless interoperability) of tools with Eclipse is the plug-in. With the exception of a small run-time kernel, everything in Eclipse is a plug-in. This means that every plug-in developed integrates with Eclipse in exactly the same way as other plug-ins; in this respect, all features are created equal. Eclipse provides plug-ins for a wide variety of features, some of which are through third parties using both free and commercial models. Examples of plug-ins include UML plug-in for Sequence and other UML diagrams, plug-in for Database explorer, and many others.

The Eclipse SDK includes the Eclipse Java Development Tools, offering an IDE with a built-in incremental Java compiler and a full model of the Java source files. This allows for advanced refactoring techniques and code analysis. The IDE also makes use of a workspace, in this case a set of metadata over a flat filespace allowing external file modifications as long as the corresponding workspace "resource" is refreshed afterwards. The Visual Editor project allows interfaces to be created interactively, thus allowing Eclipse to be used as a RAD tool.

Eclipse's widgets are implemented by a widget toolkit for Java called SWT, unlike most Java applications, which use the Java standard Abstract Window Toolkit (AWT) or Swing. Eclipse's user interface also uses an intermediate GUI layer called JFace, which simplifies the construction of applications based on SWT.” -- Wikipedia

### Creating workspaces and projects

Related Eclipse projects are normally grouped under the same workspace e.g. all the Eclipse projects for a particular product will be placed under the workspace associated with that product. The workspaces and projects for Eclipse should be created in specific locations to ensure consistency across development workstations.

All workspaces should be created under the folder **C:\Workspaces** on Windows or on Unix and OS X under the folder **~/Workspaces**.

Eclipse projects are divided into two groups. The first group consists of those projects that are added to the Git version control repository. The second group consists of projects that are created for experimentation or testing purposes only. These projects are not added to Git.



Eclipse workspaces contain hard-coded paths that are relevant only to a particular development workstation. For this reason, they should not be added to the Git repository. Instead, a new workspace should always be created on each development machine and the Eclipse projects stored in the Git repository should be imported into the workspace.

### Installing Eclipse

Download and install the Eclipse IDE for Java EE Developers (Luna) from **https://eclipse.org**.

### Installing Eclipse add-ons

The following sections describe how to install the Eclipse IDE add-ons to support Java and JEE development.

#### Install the Eclipse Graphical Modeling Framework Add-on

Complete the following steps to install the Eclipse Graphical Modeling Framework Add-on into the Eclipse IDE:

1. Start the Eclipse IDE.
2. Select **Help > Install New Software…** from the menu.
3. Click **Add…** to add a new update site.
4. Select the **Luna –** [**http://download.eclipse.org/releases/luna**](http://download.eclipse.org/releases/luna)update site from the **Work with** drop-down list.
5. Expand the **Modeling** node in the tree and select the following items:  
     
   **Graphical Modeling Framework (GMF) Runtime**
6. Click **Next**.
7. Click **Next**.
8. Accept the license agreement and click **Finish** to start the installation.
9. Restart the Eclipse IDE when requested.

#### Install the JBoss Core Tools Eclipse Add-on

Complete the following steps to install the JBoss Core Tools Eclipse Add-on into the Eclipse IDE:

1. Start the Eclipse IDE.
2. Select **Help > Install New Software…** from the menu.
3. Click **Add…** to add a new update site.
4. Enter the following details for the new update site (Repository) and click **OK**:  
     
   **Name:** JBoss Core Tools  
   **Location:** http://download.jboss.org/jbosstools/updates/stable/luna/
5. Expand the **\* Abridged JBoss Tools** node in the tree and select the following items:  
     
   **JBossAS Tools**
6. Click **Next**.
7. Click **Next**.
8. Accept the license agreement and click **Finish** to start the installation.
9. Restart the Eclipse IDE when requested.

#### Install the Eclipse BPMN2 Modeler – Diagram Editor Eclipse Add-on

Complete the following steps to install the Eclipse BPMN2 Modeler – Diagram Editor Eclipse Add-on into the Eclipse IDE:

1. Start the Eclipse IDE.
2. Select **Help > Install New Software…** from the menu.
3. Click **Add…** to add a new update site.
4. Enter the following details for the new update site (Repository) and click **OK**:  
     
   **Name:** Eclipse BPMN2 Modeler  
   **Location:** http://download.eclipse.org/bpmn2-modeler/updates/luna/1.1.0/
5. Expand the **Eclipse BPMN2 Modeler** node in the tree and select the following items:  
     
   **BPMN2 Modeler – Diagram Editor**
6. Click **Next.**
7. Click **Next**.
8. Accept the license agreement and click **Finish** to start the installation.
9. Restart the Eclipse IDE when requested.

#### Optional: Install the Jindent Eclipse Add-on

Complete the following steps to install the Jindent Eclipse Add-on into the Eclipse IDE:

1. Start the Eclipse IDE.
2. Select **Help > Install New Software…** from the menu.
3. Click **Add…** to add a new update site.
4. Enter the following details for the new update site (Repository) and click **OK**:  
     
   **Name:** Jindent  
   **Location:** http://downloads.jindent.com/plugins/eclipse/3.x/
5. Expand the **Newforms – Software Development** node in the tree and select the following items:  
     
   **Jindent – Source Code Formatter**
6. Click **Next**.
7. Click **Next.**
8. Accept the license agreement and click **Finish** to start the installation.
9. Restart the Eclipse IDE when requested.

#### Optional: Install the Android Development Tools (ADT) Eclipse Add-on

Complete the following steps to install the Android Development Tools (ADT) Eclipse Add-on into the Eclipse IDE:

1. Start the Eclipse IDE.
2. Select **Help > Install New Software…** from the menu.
3. Click **Add…** to add a new update site.
4. Enter the following details for the new update site (Repository) and click **OK**:  
     
   **Name:** Android Development Tools (ADT)  
   **Location:** https://dl-ssl.google.com/android/eclipse/
5. Expand the **Developers Tools** node in the tree and select the following items:  
     
   **Android DDMS  
   Android Development Tools  
   Android Hierarchy Viewer  
   Android Traceview**
6. Click **Next**.
7. Accept the license agreement and click **Finish** to start the installation.
8. Restart the Eclipse IDE when requested.

### Importing existing preferences

Eclipse can be configured by importing an existing set of preferences using the **File > Import… > General > Preferences** option. These preferences must have been previously exported using the **File > Export… > General > Preferences** option.



Preferences are normally specific to a particular developer’s workstation since they include hard-coded paths. They are also configured for a particular workspace. This means that once you have manually configured the preferences, for a workspace on your developer workstation, you should export them as described above. The preferences can then be imported to configure a new workspace. You will normally **NOT** be able to use the preferences exported on another machine.

### Manually configuring preferences

Complete the following steps to configure Eclipse for Java development:

1. Switch to the **Java EE** perspective.
2. Select the **Window > Preferences…** or **Eclipse > Preferences…** menu option.
3. Select the **General > Editors > Text Editors** option.
   1. Set the **Displayed tab width** to **2**.
   2. Select **Insert spaces for tabs**.
   3. Select the **Show print margin** option and set the **Print margin column** to **100**.
4. Select the **Java > Code Style > Formatter** option and import the Java profile from Git:  
     
   mmp-java/resources/Eclipse Java Profile.xml
5. Select the **Java > Compiler > Errors/Warnings** option.
   1. Expand the **Deprecated and restricted API** section and set the **Deprecated API** option to **Ignore**.
   2. Expand the **Unnecessary code** section and set the **Unused import** to **Ignore**.
   3. Expand the **Generic types** section and set the **Unchecked generic type operation** to **Ignore**.
6. Select the **Validation** option.
   1. Unselect the **Build** checkbox next to the **XML Validator** item in the list.
   2. Click on the **…** button for the **XML Validator** item in the list.
   3. Select the **Exclude Group** from the list.
   4. Click **Add Rule…**
   5. Choose the **Folder or file name** option and click **Next >**
   6. Enter **beans.xml** as the file or folder name and click **Finish**.
7. Select the **Web > CSS Files > Editor** option. If this option is not available then switch to the J2EE view and enable any required capabilities.
   1. Set the **Line width** to **100**.
   2. Select the **Indent using spaces** option and set the **Indentation size** to 2.
   3. Set the **Capitilization style** for **Identifier** to **Lowercase**.
   4. Deselect the **Insert line break between properties** option.
8. Select the **Web > HTML Files > Editor** option.
   1. Set the **Line width** to **100**.
   2. Select the **Indent using spaces** option and set the **Indentation size** to 2.
9. Select the **Web > HTML Files > Validation** option.
   1. Expand the **Elements** section and set the **Unknown tag name** option to **Ignore**.
   2. Expand the **Elements** section and set the **Invalid tag location** option to **Ignore**.
   3. Expand the **Attributes** section and set the **Undefined attribute name** option to **Ignore**.
10. Select the **Web Services > Server and Runtime** option.
    1. Select **JBoss AS v7.1** as the **Web service runtime**.
    2. Select **Apache Axis2** as the **Web services runtime**.
11. Select the **XML > XML Files > Editor** option.
    1. Set the **Line width** to **100**.
    2. Select the **Indent using spaces** option and set the **Indentation size** to 2.
12. Select the **XML > XML Files > Validation** option.
    1. Set the **Validating files > No grammar is specified** option to **Ignore**.
13. Click **OK**.
14. If desired, save your preferences using the **File > Export… > General > Preferences** option. These preferences can be imported when creating a new workspace using the **File > Import… > General > Preferences** option.

## WildFly

WildFly is an application server that implements the Java Platform, Enterprise Edition (Java EE).

WildFly is written in Java and as such is cross-platform: usable on any operating system that supports Java.

WildFly was developed by JBoss, now a division of Red Hat. Licensed under the terms of the GNU Lesser General Public License, WildFly is free and open source software.

### Installing WildFly 8 on Mac OS X

Complete the following steps to install and configure WildFly 8 on Mac OS X:

1. Open a terminal and switch to the **root** user by executing the following command:  
     
   sudo su -
2. Execute the following commands as the **root** user to download the **wildfly-8.2.0.Final.tar.gz** file from **www.wildfly.org** and extract under **/opt**:  
     
   curl -o /tmp/wildfly-8.2.0.Final.tar.gz \  
    http://download.jboss.org/wildfly/8.2.0.Final/wildfly-8.2.0.Final.tar.gz  
   tar zxf /tmp/wildfly-8.2.0.Final.tar.gz -C /opt
3. Execute the following commands as the **root** user to grant all users write access to the WildFly deployment and to create the version independent symbolic link:  
     
   chmod -R a+rw /opt/wildfly-8.2.0.Final  
   ln -s /opt/wildfly-8.2.0.Final /opt/wildfly
4. Install the PostgreSQL JDBC driver module by completing the following steps:
   1. Clone the **mmp-java** project from the Bitbucket repository using the following URL:  
        
      https://marcusportmann@bitbucket.org/marcusportmann/mmp-java.git
   2. Extract the module **mmp-java/resources/wildfly-postgresql-module.tar.gz** under **/opt/wildfly/modules/system/layers/base**.

### Installing WildFly 8 on CentOS

Complete the following steps to install and configure WildFly 8 on CentOS:

1. Open a terminal and switch to the **root** user.
2. Execute the following commands as the **root** user to download the **wildfly-8.2.0.Final.tar.gz** file from **www.wildfly.org** and extract under **/opt**:  
     
   curl -o /tmp/wildfly-8.2.0.Final.tar.gz \  
    http://download.jboss.org/wildfly/8.2.0.Final/wildfly-8.2.0.Final.tar.gz  
   tar zxf /tmp/wildfly-8.2.0.Final.tar.gz -C /opt
3. Execute the following commands as the **root** user to grant all users write access to the WildFly deployment and to create the version independent symbolic link:  
     
   chmod -R a+rw /opt/wildfly-8.2.0.Final  
   ln -s /opt/wildfly-8.2.0.Final /opt/wildfly
4. Install the PostgreSQL JDBC driver module by completing the following steps:
   1. Clone the **mmp-java** project from the Bitbucket repository using the following URL:  
        
      https://marcusportmann@bitbucket.org/marcusportmann/mmp-java.git
   2. Extract the module **mmp-java/resources/wildfly-postgresql-module.tar.gz** under **/opt/wildfly/modules/system/layers/base**.
5. Execute the following commands as the **root** user to create the WildFly user:  
     
   mkdir /home/wildfly  
   groupadd --gid 201 wildfly  
   useradd --uid 201 --gid 201 -s /bin/bash -d /home/wildfly wildfly  
   chown -R wildfly /home/wildfly  
   chgrp -R wildfly /home/wildfly
6. Create the new file **/etc/profile.d/wildfly.sh** with the following contents:  
     
   export JBOSS\_HOME=/opt/wildfly  
   export WILDFLY\_HOME=/opt/wildfly  
   export PATH=$WILDFLY\_HOME/bin:$PATH  
   export MODULEPATH=
7. Create the new file **/etc/init.d/wildfly** with the following contents:  
     
   #!/bin/sh  
   ### BEGIN INIT INFO  
   # Provides: wildfly  
   # Required-Start: $local\_fs $remote\_fs $network $syslog  
   # Required-Stop: $local\_fs $remote\_fs $network $syslog  
   # Default-Start: 2 3 4 5  
   # Default-Stop: 0 1 6  
   # Short-Description: Start/Stop WildFly AS 8  
   ### END INIT INFO  
   #  
   # Source some script files in order to set and export environmental variables as well as add the appropriate executables to $PATH  
   [ -r /etc/profile.d/java.sh ] && . /etc/profile.d/java.sh  
   [ -r /etc/profile.d/wildfly.sh ] && . /etc/profile.d/wildfly.sh  
     
   start(){  
   echo "Starting WildFly 8"  
   su -l wildfly -c "sh ${WILDFLY\_HOME}/bin/standalone.sh -b 0.0.0.0 >/dev/null 2>/dev/null &"  
   }  
     
   stop(){  
   echo "Stopping WildFly 8"  
   su -l wildfly -c "sh ${WILDFLY\_HOME}/bin/jboss-cli.sh --connect --command=:shutdown"  
   }  
     
   restart(){  
   stop  
   # give stuff some time to stop before we restart  
   sleep 60  
   # protect against any services that can't stop before we restart  
   su -l wildfly -c "killall java"  
   start  
   }  
     
   case "$1" in  
   start)  
   start  
   ;;  
   stop)  
   stop  
   ;;  
   restart)  
   restart  
   ;;  
   \*)  
   echo "Usage: /etc/init.d/wildfly {start|stop|restart}"  
   exit 1  
   ;;  
   esac  
     
   exit 0
8. Execute the following command to set the permissions for the WildFly files:  
     
   chown -R wildfly.wildfly /opt/wildfly-8.2.0.Final   
     
   chmod a+x /etc/init.d/wildfly
9. Optional: Execute the following commands to permit access to JBoss through the CentOS firewall:  
     
   iptables -I INPUT -m state --state NEW -m tcp -p tcp --dport 8080 -j ACCEPT  
   service iptables save  
   service iptables restart  
     
   ip6tables -I INPUT -m state --state NEW -m tcp -p tcp --dport 8080 -j ACCEPT  
   service ip6tables save  
   service ip6tables restart
10. Optional: Execute the following commands to grant AJP access to a specific host (This is required if using another server as an HTTP server):  
      
    NOTE: Replace XXX.XXX.XXX.XXX with the IP address of the HTTP server.  
      
    iptables -I INPUT --source XXX.XXX.XXX.XXX -m state --state NEW -m tcp -p tcp --dport 8009 -j ACCEPT  
    service iptables save  
    service iptables restart  
      
    ip6tables -I INPUT --source XXX.XXX.XXX.XXX -m state --state NEW -m tcp -p tcp --dport 8009 -j ACCEPT  
    service ip6tables save  
    service ip6tables restart
11. OPTIONAL: Complete the following steps to setup the WildFly admin user:
    1. Start the WildFly application server.
    2. Execute the ***<WILDFLY\_HOME>*/bin/add-user.sh** script.
    3. Select option **(a)** to add a **Management User**.
    4. Enter the name of the new admin user e.g. admin.
    5. Enter the password for the new admin user e.g. admin.
    6. Complete the steps to add the new admin user.

### Setting up a PostgreSQL data source on WildFly 8

Complete the following steps to setup a PostgreSQL Data Source on WildFly 8:

1. Clone the **mmp-java** project from the Bitbucket repository using the following URL:  
     
   https://marcusportmann@bitbucket.org/marcusportmann/mmp-java.git
2. Create the following folder structure under the WildFly distribution folder:  
     
   ***<WILDFLY\_HOME>*/modules/system/layers/base/org/postgresql/main**
3. Copy the following JAR from the **mmp-java mmp-java /resources** folder to the ***<WILDFY\_HOME>*/modules/system/layers/base/org/postgresql/main** folder:  
     
   **postgresql-9.3-1101.jdbc41.jar**
4. Create a new file named **module.xml** under the **main** folder with the following contents:  
     
   <?xml version="1.0" encoding="UTF-8"?>  
   <module xmlns="urn:jboss:module:1.0" name="org.postgresql">  
    <resources>  
    <resource-root path="postgresql-9.3-1101.jdbc41.jar"/>  
    </resources>  
    <dependencies><module name="javax.api"/></dependencies>  
   </module>
5. Add an entry for the data source to the ***<WILDFLY\_HOME>*/standalone/configuration/standalone.xml** file under the **<datasources>** node:  
     
   <datasources>  
    ...  
    <datasource jndi-name="java:jboss/datasources/SampleDS" pool-name="SampleDS" enabled="true" jta="true"   
    use-java-context="true" use-ccm="true">  
    <connection-url>jdbc:postgresql://localhost:5432/sampledb</connection-url>  
    <driver-class>org.postgresql.Driver</driver-class>  
    <driver>postgresql-jdbc4</driver>  
    <pool>  
    <min-pool-size>2</min-pool-size>  
    <max-pool-size>5</max-pool-size>  
    <prefill>true</prefill>  
    <use-strict-min>false</use-strict-min>  
    <flush-strategy>FailingConnectionOnly</flush-strategy>  
    </pool>  
    <security>  
    <user-name>sampledb</user-name>  
    <password>Password1</password>  
    </security>  
    <validation>  
    <check-valid-connection-sql>SELECT 1</check-valid-connection-sql>  
    <validate-on-match>false</validate-on-match>  
    <background-validation>false</background-validation>  
    <use-fast-fail>false</use-fast-fail>  
    </validation>  
    </datasource>  
      
   </datasources>
6. Add an entry for the data source to the ***<WILDFLY\_HOME>*/standalone/configuration/standalone.xml** file under the **<datasources>** node:  
     
   <datasources>  
    ...  
    <drivers>  
    <driver name="postgresql-jdbc4" module="org.postgresql"/>  
    ...  
    </drivers>  
   </datasources>

### Enabling WildFly 8 on all network interfaces

Complete the following steps to setup a WildFly 8 server instance to listen on all network interfaces:

1. Add an entry for the **global** interface to the ***<WILDFLY\_HOME>*/standalone/configuration/standalone.xml** file under the **<interfaces>** node:  
     
   <interfaces>  
    ...  
    <interface name="global">  
    <!-- Use the wildcard address -->  
    <any-address/>  
    </interface>  
    ...  
   </interfaces>
2. If required modify the existing **management** interface in the ***<WILDFLY\_HOME>*/standalone/configuration/standalone.xml** file under the **<interfaces>** node:  
     
   <interfaces>  
    ...  
    <interface name="management">  
    <!-- Use the wildcard address -->  
    <any-address/>  
    </interface>  
    ...  
   </interfaces>
3. Edit the **<socket-binding>** node to use the **global** interface as the default interface:  
     
   <socket-binding-group name="standard-sockets" default-interface="**global**">

## Apache

The Apache HTTP Server Project is an effort to develop and maintain an open-source HTTP server for modern operating systems including UNIX and Windows NT. The goal of the Apache project is to provide a secure, efficient and extensible server that provides HTTP services in sync with the current HTTP standards.

### Installing Apache on CentOS

Complete the following steps to install Apache on CentOS:

1. Open a terminal and switch to the **root** user.
2. Execute the following commands to download and install Apache from the **yum repositories**.  
     
   yum install httpd mod\_ssl  
     
   chkconfig httpd on  
     
   service httpd start

### Setting up Apache as a web server fronting JBoss 7 on CentOS

Complete the following steps to setup Apache as a web server fronting JBoss 7 on CentOS:

1. Open a terminal and switch to the **root** user.
2. Edit the ***<JBOSS\_HOME>*/standalone/configuration/standalone.xml** file and put the following **<system-properties>** element inside the **<server>** element after the **<extensions>** element.  
     
   <system-properties>  
    <property name="jvmRoute" value="centos"/>  
   </system-properties>
3. Edit the ***<JBOSS\_HOME>*/standalone/configuration/standalone.xml** file and put the following **<connector>** element inside the **<subsystem>** element after the http **<connector>** element:  
     
   <subsystem xmlns="urn:jboss:domain:web:1.0" default-virtual-server="default-host">  
    <connector name="http" protocol="HTTP/1.1" socket-binding="http" scheme="http"/>  
    <connector name="ajp" protocol="AJP/1.3" socket-binding="ajp" scheme="http"/>  
    …
4. Edit the ***<JBOSS\_HOME>*/standalone/configuration/standalone.xml** file and put the following **<socket-binding>** element inside the **<socket-binding-group>** element after the http **<socket-binding>** element:  
     
   <socket-binding-group name="standard-sockets" default-interface="public">  
    <socket-binding name="http" port="8080"/>  
    <socket-binding name="ajp" port="8009"/>  
    …
5. Create a file named **jboss.conf** under the directory **/etc/httpd/conf.d** with the following contents, replacing the application path in red as appropriate:  
     
   <Proxy balancer://**sample**>  
    BalancerMember ajp://127.0.0.1:8009/**sample** route=centos  
    ProxySet stickysession=JSESSIONID  
   </Proxy>  
     
   ProxyPass /**sample** balancer://**sample**  
   ProxyPassReverse /**sample** balancer://**sample**
6. Execute the following commands to permit access to Apache through the CentOS firewall:  
     
   iptables -I INPUT -m state --state NEW -m tcp -p tcp --dport 80 -j ACCEPT  
   iptables -I INPUT -m state --state NEW -m tcp -p tcp --dport 443 -j ACCEPT  
   service iptables save  
   service iptables restart  
     
   ip6tables -I INPUT -m state --state NEW -m tcp -p tcp --dport 80 -j ACCEPT  
   ip6tables -I INPUT -m state --state NEW -m tcp -p tcp --dport 443 -j ACCEPT  
   service ip6tables save  
   service ip6tables restart
7. Execute the following command to restart Apache:  
     
   service httpd restart

### Enabling SSL support for Apache on CentOS

Complete the following steps to enable SSL support for Apache on CentOS:

1. TODO: COMPLETE THIS SECTION.

## PostgreSQL

PostgreSQL is an object-relational database management system (ORDBMS) released under a BSD-style license. -- Wikipedia.

### Using PostgreSQL with Mac OS X

The following sections explain how to use the PostgreSQL object-relational database management system (ORDBMS) on Mac OS X.

#### Installing PostgreSQL usinc MacPorts

Complete the following steps to install PostgreSQL on Mac OS X using the MacPorts software repository:

1. TODO: COMPLETE THIS SECTION.

#### Creating a PostgreSQL Database

Complete the following steps to create and initialize a PostgreSQL Database on Mac OS X:

1. Create the database.  
     
   sudo su postgres -c 'createdb sampledb'
2. Execute the SQL statements to initialise the database.  
     
   sudo su postgres -c 'psql -d sampledb -f SamplePostgres.sql'

#### Deleting a PostgreSQL Database

Complete the following steps to delete a PostgreSQL Database on Mac OS X:

1. To delete the database.  
     
   sudo su postgres -c 'dropdb sampledb'

#### Restarting the PostgreSQL Database Server

Complete the following steps to restart the PostgreSQL Database Server on Mac OS X:

1. To restart the database.  
     
   sudo su postgres -c 'pg\_ctl -D /opt/local/var/db/postgresql93/defaultdb -l /opt/local/var/db/postgresql93/postgres.log stop'  
     
   sudo su postgres -c 'pg\_ctl -D /opt/local/var/db/postgresql93/defaultdb -l /opt/local/var/db/postgresql93/postgres.log start'  
     
   sudo su postgres -c 'tail –n500 /opt/local/var/db/postgresql93/postgres.log'

#### Configuring the PostgreSQL Database Server

Complete the following steps to configure the PostgreSQL Database Server on Mac OS X:

1. Open the PostgreSQL configuration file using the following command:  
     
   sudo vi /opt/local/var/db/postgresql93/defaultdb/postgresql.conf
2. Change the **max\_prepared\_transactions** value to 10.
3. Change the **max\_connections** value to 10.
4. Close and save the file.

### Using PostgreSQL with Ubuntu Linux

The following sections explain how to use the PostgreSQL object-relational database management system (ORDBMS) on Ubuntu Linux.

#### Installing PostgreSQL

Complete the following steps to install PostgreSQL on Ubuntu Linux:

1. TODO: Complete this section.

#### Creating a PostgreSQL Database

Complete the following steps to create and initialize a PostgreSQL Database on Ubuntu Linux:

1. Create the database.  
     
   sudo su postgres -c 'createdb sampledb'
2. Execute the SQL statements to initialise the database.  
     
   sudo su postgres -c 'psql -d sampledb -f SamplePostgres.sql'

#### Deleting a PostgreSQL Database

Complete the following steps to delete a PostgreSQL Database on Ubuntu Linux:

1. To delete the database.  
     
   sudo su postgres -c 'dropdb sampledb'

# Development Artifacts

Java developers produce a number of different Java development artifacts. This chapter identifies these artifacts and documents the development processes used to produce these artifacts.

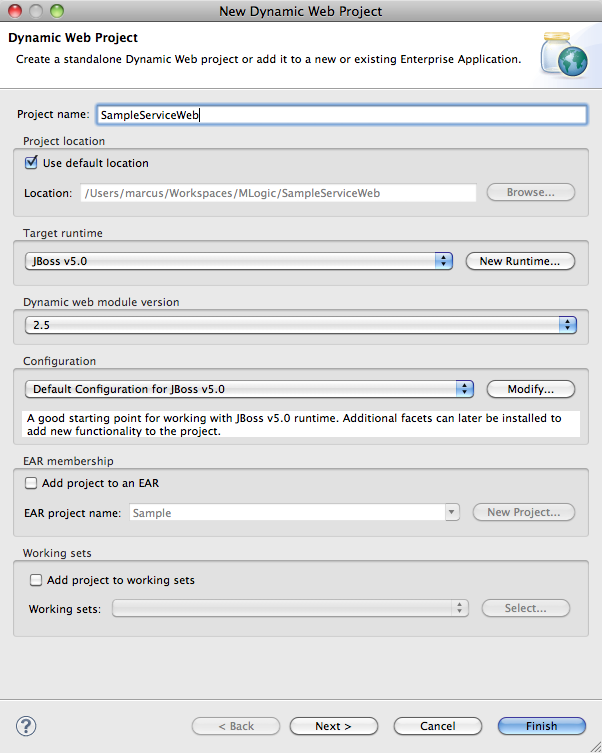
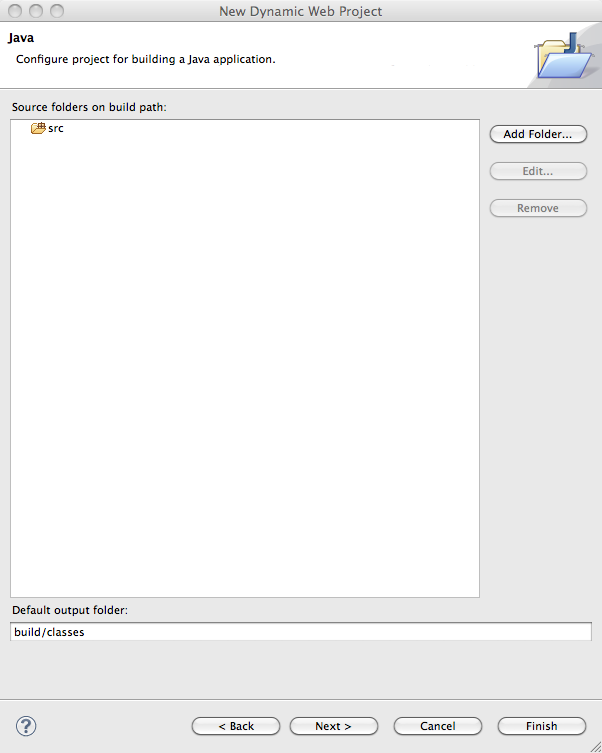
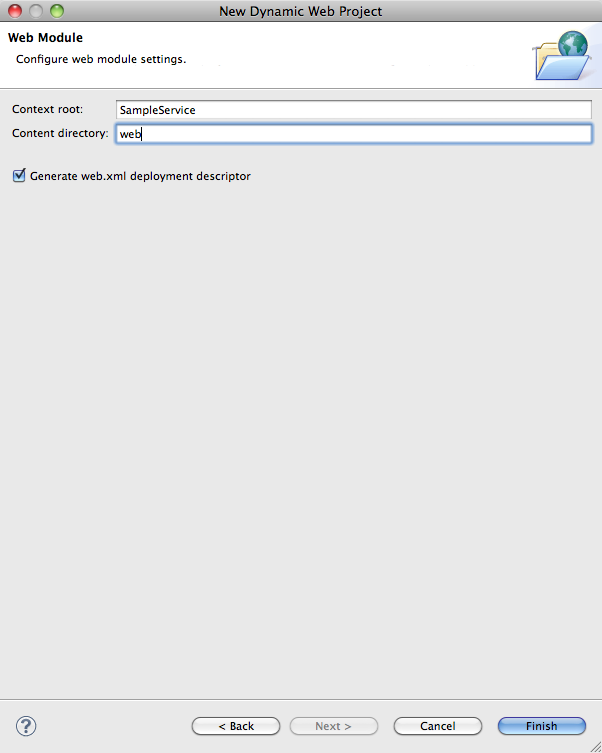
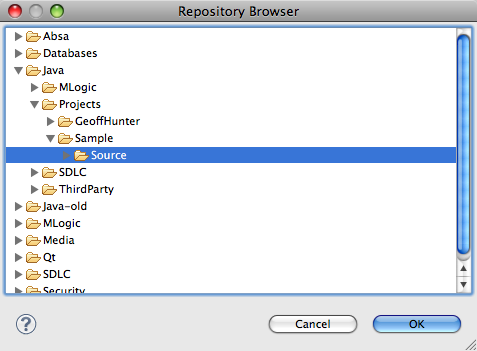
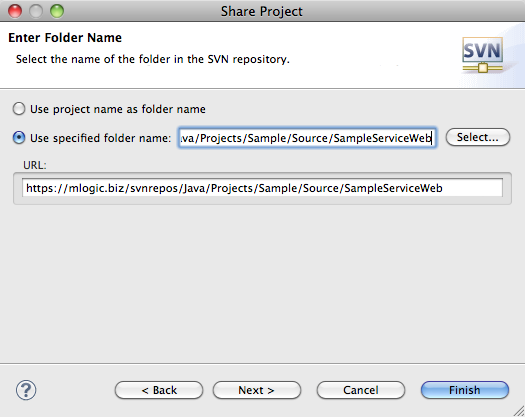
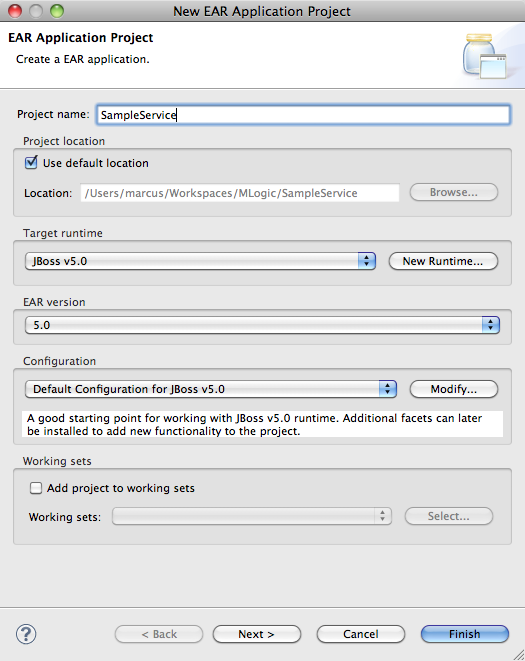
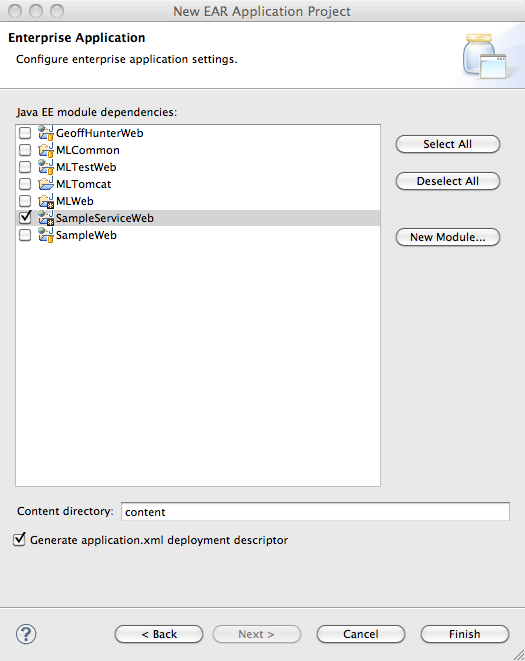
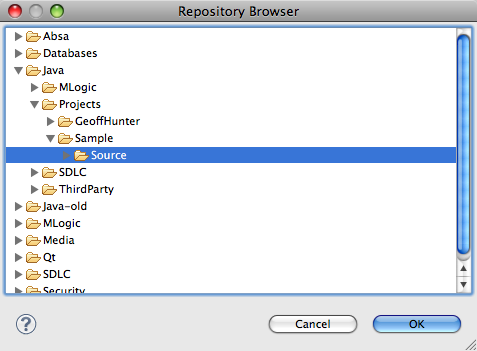
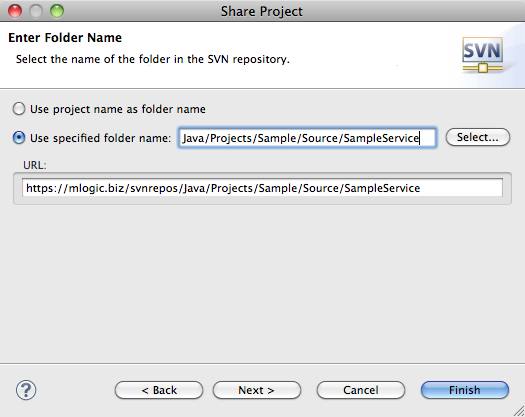
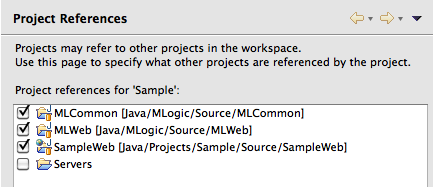
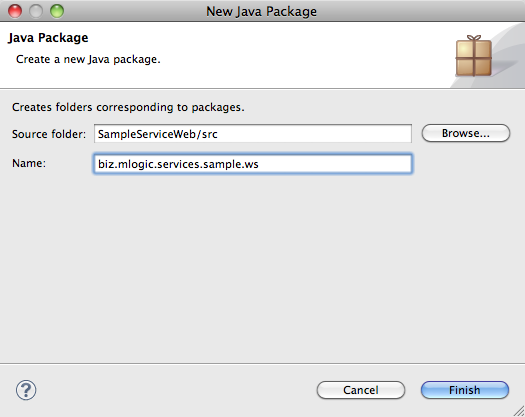
## JEE Web Application

### Creating a new JEE web application

1. Create a new directory for the local Git respository for the JEE Web Application and initialize it.
2. Create a new directory named **src** under the new Git repository that will store all the project directories for the new JEE web application.
3. Create a new file named **pom.xml** under the **src** directory with the following contents, changing the entries in **red** as appropriate:  
     
   <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">  
     
    <modelVersion>4.0.0</modelVersion>  
     
    <groupId>**guru.mmp**</groupId>  
    <artifactId>**mmp-sample**</artifactId>  
    <version>1.0.0</version>  
    <name>**mmp-sample**</name>  
     
    <packaging>pom</packaging>  
     
    <properties>  
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  
    </properties>  
     
    <modules>  
    <module>ear</module>  
    <module>web</module>  
    </modules>  
     
   </project>
4. Create a new directory named **web** under the **src** directory.
5. Create a new file named **pom.xml** under the **web** directory with the following contents, changing the entries in **red** as appropriate:  
     
   <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">  
     
    <modelVersion>4.0.0</modelVersion>  
     
    <groupId>**guru.mmp**</groupId>  
    <artifactId>**mmp-sample-web**</artifactId>  
    <version>1.0.0</version>  
    <name>**mmp-sample-web**</name>  
     
    <parent>  
    <groupId>**guru.mmp**</groupId>  
    <artifactId>**mmp-sample**</artifactId>  
    <version>1.0.0</version>  
    </parent>  
     
    <packaging>war</packaging>  
     
    <properties>  
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  
    </properties>  
     
    <dependencies>  
    <dependency>  
    <groupId>guru.mmp</groupId>  
    <artifactId>mmp-application-wicket</artifactId>  
    <version>1.0.0</version>  
    <type>jar</type>  
    </dependency>  
    <dependency>  
    <groupId>javax</groupId>  
    <artifactId>javaee-api</artifactId>  
    <version>7.0</version>  
    <scope>provided</scope>  
    </dependency>  
    </dependencies>  
     
    <build>  
    <plugins>  
    <plugin>  
    <groupId>org.apache.maven.plugins</groupId>  
    <artifactId>maven-compiler-plugin</artifactId>  
    <version>3.1</version>  
    <configuration>  
    <source>1.8</source>  
    <target>1.8</target>  
    </configuration>  
    </plugin>  
    </plugins>  
    </build>  
     
   </project>
6. Create a new directory named **ear** under the **src** directory.
7. Create a new file named **pom.xml** under the **ear** directory with the following contents, changing the entries in **red** as appropriate:  
     
   <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">  
     
    <modelVersion>4.0.0</modelVersion>  
     
    <groupId>**guru.mmp**</groupId>  
    <artifactId>**mmp-sample-ear**</artifactId>  
    <version>1.0.0</version>  
    <name>**mmp-sample-ear**</name>  
     
    <parent>  
    <groupId>**guru.mmp**</groupId>  
    <artifactId>**mmp-sample**</artifactId>  
    <version>1.0.0</version>  
    </parent>  
     
    <packaging>ear</packaging>  
     
    <dependencies>  
    <dependency>  
    <groupId>**guru.mmp**</groupId>  
    <artifactId>**mmp-sample-web**</artifactId>  
    <version>1.0.0</version>  
    <type>jar</type>  
    </dependency>  
    </dependencies>  
     
    <build>  
    <plugins>  
    <plugin>  
    <groupId>org.apache.maven.plugins</groupId>  
    <artifactId>maven-compiler-plugin</artifactId>  
    <version>3.1</version>  
    <configuration>  
    <source>1.8</source>  
    <target>1.8</target>  
    </configuration>  
    </plugin>  
    <plugin>  
    <groupId>org.apache.maven.plugins</groupId>  
    <artifactId>maven-ear-plugin</artifactId>  
    <version>2.9</version>  
    <configuration>  
    <version>7</version>  
    <defaultLibBundleDir>lib</defaultLibBundleDir>  
    <modules>  
    <webModule>  
    <groupId>**guru.mmp**</groupId>  
    <artifactId>**mmp-sample-web**</artifactId>  
    <context-root>**/sample**</context-root>  
    </webModule>  
    </modules>  
    </configuration>  
    </plugin>  
    </plugins>  
    </build>  
     
   </project>
8. Create the Java, HTML and CSS files for the **“Base” Web Page** pages for the web application.
   1. Create the following directory under the **src** directory for the JEE Web Application:  
        
      **web/src/main/java/<CompanyPrefix>/<ApplicationName>/web/page**  
        
      e.g. web/src/main/java/guru/mmp/sample/web/page
   2. Create the following directory under the **src** directory for the JEE Web Application:  
        
      **web/src/main/resources/<CompanyPrefix>/<ApplicationName>/web/page**  
        
      e.g. web/src/main/resources/guru/mmp/sample/web/page
   3. Create the following directory under the **src** directory for the JEE Web Application:  
        
      **web/src/main/resources/<CompanyPrefix>/<ApplicationName>/web/resource/css**  
        
      e.g. web/src/main/resources/guru/mmp/sample/web/resource/css
   4. Create the logo for the new application as a file under the **css** folder with the name **headerLogo.png**.
   5. Create a file named **application.css** under the **css** package with the following contents:  
        
      #headerLogo  
      {  
       position: absolute;  
       z-index: 6;  
       width: 134px;  
       height: 16px;  
       top: 18px;  
       right: 21px;  
       background: url(headerLogo.png) no-repeat left top;  
      }
   6. Create a file named **HomePage.java** under the **web/src/main/java/<CompanyPrefix>/<ApplicationName>/web/page** directory with the following contents:  
        
      e.g. web/src/main/java/guru/mmp/sample/web/page/HomePage.java  
        
      package guru.mmp.sample.web.page;  
        
      import guru.mmp.application.web.page.AnonymousOnlyWebPage;  
      import guru.mmp.application.web.template.page.TemplateWebPage;  
        
      /\*\*  
       \* The <code>HomePage</code> class implements the "Home"  
       \* page for the web application.  
       \*/  
      @AnonymousOnlyWebPage  
      public class HomePage extends TemplateWebPage  
      {  
       /\*\*  
       \* Constructs a new <code>HomePage</code>.  
       \*/  
       public HomePage()   
       {  
       super("Sample | Home", "Home");  
       }  
      }
   7. Create a file named **HomePage.html** under the **web/src/main/resources/<CompanyPrefix>/<ApplicationName>/web/page** directory with the following contents:  
        
      e.g. web/src/main/resources/guru/mmp/sample/web/page/HomePage.html  
        
      <!DOCTYPE HTML PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"   
       "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">  
        
      <html xml:lang="en" xmlns="http://www.w3.org/1999/xhtml">  
       <head>  
       <link href="../../../../../../../../../MLADF/Source/MLApplication/src/biz/mlogic/application/web/template/resource/theme/mlogic/template.css" rel="StyleSheet" type="text/css" />  
       <link href="../resource/css/application.css" rel="StyleSheet" type="text/css" />  
       </head>  
       <body>  
       <div id="childContentHolderSpacer"></div>  
       <div id="childContentHolder">  
       <wicket:extend>  
       <h1>Home</h1>  
       </wicket:extend>  
       </div>  
       </body>  
      </html>
   8. Create a file named Secure**HomePage.java** under the **web/src/main/java/<CompanyPrefix>/<ApplicationName>/web/page** directory with the following contents:  
        
      e.g. web/src/main/java/guru/mmp/sample/web/page/SecureHomePage.java  
        
      package guru.mmp.sample.web.page;  
        
      import guru.mmp.application.web.page.WebPageSecurity;  
      import guru.mmp.application.web.template.TemplateSecurity;  
      import guru.mmp.application.web.template.page.TemplateWebPage;  
        
      /\*\*  
       \* The <code>SecureHomePage</code> class implements the secure "Home"  
       \* page for the web application.  
       \*/  
      @WebPageSecurity(TemplateSecurity.FUNCTION\_CODE\_SECURE\_HOME)  
      public class SecureHomePage extends TemplateWebPage  
      {  
       /\*\*  
       \* Constructs a new <code>SecureHomePage</code>.  
       \*/  
       public SecureHomePage()   
       {  
       super("Sample | Home", "Home");  
       }  
      }
   9. Create a file named **SecureHomePage.html** under the **web/src/main/resources/<CompanyPrefix>/<ApplicationName>/web/page** directory with the following contents:  
        
      e.g. web/src/main/resources/guru/mmp/sample/web/page/SecureHomePage.html  
        
      <!DOCTYPE HTML PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"   
       "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">  
        
      <html xml:lang="en" xmlns="http://www.w3.org/1999/xhtml">  
       <head>  
       <link href="../../../../../../../../../MLADF/Source/MLApplication/src/biz/mlogic/application/web/template/resource/theme/mlogic/template.css" rel="StyleSheet" type="text/css" />  
       <link href="../resource/css/application.css" rel="StyleSheet" type="text/css" />  
       </head>  
       <body>  
       <div id="childContentHolderSpacer"></div>  
       <div id="childContentHolder">  
       <wicket:extend>  
       <h1>Home</h1>  
       </wicket:extend>  
       </div>  
       </body>  
      </html>
9. Create the “Security” class and Wicket Application class for the web application:
   1. Create a file named **<ApplicationName>Security.java** under the **web/src/main/java/<CompanyPrefix>/<ApplicationName>/web** directory with the following contents:  
        
      e.g. web/src/main/java/guru/mmp/sample/web/SampleSecurity.java  
        
      package guru.mmp.sample.web;  
        
      /\*\*  
       \* The <code>SampleSecurity</code> class provides access to the security-related information  
       \* and functionality for the web application. This includes a list of the authorized  
       \* function codes.  
       \***/**public class SampleSecurity  
      {  
       /\*\* The Sample.SampleFunction function code \*/   
       public static final String FUNCTION\_CODE\_SAMPLE\_FUNCTION = "Sample.SampleFunction";  
      }
   2. Create a file named **<ApplicationName>Application.java** under the **web/src/main/java/<CompanyPrefix>/<ApplicationName>/web** directory with the following contents:  
        
      e.g. web/src/main/java/guru/mmp/sample/web/SampleApplication.java  
        
      package guru.mmp.sample.web;  
        
      import guru.mmp.application.web.page.WebPage;  
      import guru.mmp.application.web.template.TemplateWebApplication;  
      import guru.mmp.application.web.template.navigation.NavigationGroup;  
      import guru.mmp.application.web.template.navigation.NavigationLink;  
      import guru.mmp.application.web.template.page.GroupAdministrationPage;  
      import guru.mmp.application.web.template.page.LoginPage;  
      import guru.mmp.application.web.template.page.OrganisationAdministrationPage;  
      import guru.mmp.application.web.template.page.UserAdministrationPage;  
      import guru.mmp.sample.web.page.HomePage;  
      import guru.mmp.sample.web.page.SecureHomePage;  
      import org.apache.wicket.Page;  
        
      /\*\*  
      \* The <code>SampleApplication</code> provides the implementation of the Wicket Web   
      \* Application class for the web application.  
      \*/  
      public class SampleApplication extends TemplateWebApplication  
      {  
       /\*\*  
       \* Constructs a new <code>SampleApplication</code>.  
       \*/  
       public SampleApplication()   
       {  
       super("Sample");  
       }  
        
       /\*\*  
       \* Returns the home page for the application.  
       \*  
       \* @return the home page for the application  
       \*/  
       public Class<? extends Page> getHomePage()  
       {  
       return HomePage.class;  
       }  
        
       /\*\*  
       \* Returns the page that users will be redirected to once they have logged into the application.  
       \*  
       \* @return the page that users will be redirected to once they have logged into the application.  
       \*/  
       public Class<? extends WebPage> getSecureHomePage()  
       {  
       return SecureHomePage.class;  
       }  
        
       /\*\*  
       \* Abstract method that must be implement by all application-specific <code>WebApplication</code>  
       \* subclasses to setup the navigation hierarchy for the application.  
       \*   
       \* @param root the root of the navigation hierarchy  
       \*/  
       protected void initNavigation(NavigationGroup root)  
       {  
       NavigationGroup sampleGroup = new NavigationGroup("Sample");  
        
       sampleGroup.addItem(new NavigationLink("Home", HomePage.class));  
       sampleGroup.addItem(new NavigationLink("Home", SecureHomePage.class));  
       root.addItem(sampleGroup);  
        
       NavigationGroup administrationGroup = new NavigationGroup("Administration");  
        
       administrationGroup.addItem(new NavigationLink("Organisations", OrganisationAdministrationPage.class));  
       administrationGroup.addItem(new NavigationLink("Groups", GroupAdministrationPage.class));  
       administrationGroup.addItem(new NavigationLink("Users", UserAdministrationPage.class));  
       root.addItem(administrationGroup);  
       }  
      }
10. Enable CDI for the web application:
    1. Create a file named **beans.xml** under the **web/src/main/webapp/WEB-INF** directory with the following contents:  
         
       e.g. web/src/main/webapp/WEB-INF/beans.xml   
         
       <?xml version="1.0" encoding="UTF-8"?>  
       <beans xmlns="http://java.sun.com/xml/ns/javaee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/beans\_1\_0.xsd">  
         
       </beans>
11. Create a file named **web.xml** under the **web/src/main/webapp/WEB-INF** directory with the following contents, changing the entries in **red** as appropriate:  
      
    e.g. web/src/main/webapp/WEB-INF/web.xml   
      
    <?xml version="1.0" encoding="UTF-8"?>  
    <web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"   
     xmlns="http://java.sun.com/xml/ns/javaee"   
     xmlns:web="http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd"   
     xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app\_3\_0.xsd"   
     id="WebApp\_ID" version="3.0">  
      
     <!-- ========================================================================================= -->  
     <!-- Application Configuration -->  
     <!-- ========================================================================================= -->  
     <context-param>  
     <description>Wicket Deployment</description>  
     <param-name>configuration</param-name>  
     <param-value>development</param-value>  
     </context-param>  
     <resource-ref>  
     <description>Application Data Source Reference</description>  
     <res-ref-name>java:app/jdbc/ApplicationDataSource</res-ref-name>  
     <res-type>javax.sql.DataSource</res-type>  
     <res-auth>Container</res-auth>   
     <res-sharing-scope>Shareable</res-sharing-scope>  
     <lookup-name>**java:jboss/datasources/SampleDS**</lookup-name>  
     </resource-ref>  
     <env-entry>  
     <description>Application Database Schema</description>  
     <env-entry-name>java:app/env/ApplicationDatabaseSchema</env-entry-name>  
     <env-entry-type>java.lang.String</env-entry-type>  
     <env-entry-value></env-entry-value>  
     </env-entry>  
     <env-entry>  
     <description>Application Class</description>  
     <env-entry-name>java:app/env/ApplicationClass</env-entry-name>  
     <env-entry-type>java.lang.String</env-entry-type>  
     <env-entry-value>**guru.mmp.sample.web.SampleApplication**</env-entry-value>  
     </env-entry>  
     <env-entry>  
     <description>Application Registry Path Prefix</description>  
     <env-entry-name>java:app/env/RegistryPathPrefix</env-entry-name>  
     <env-entry-type>java.lang.String</env-entry-type>  
     <env-entry-value>**/Sample**</env-entry-value>  
     </env-entry>  
     <listener>  
     <description>Application Listener</description>  
     <listener-class>guru.mmp.application.web.WebApplicationListener</listener-class>  
     </listener>  
     <filter>  
     <description>Application Filter</description>  
     <filter-name>ApplicationFilter</filter-name>  
     <filter-class>guru.mmp.application.web.WebApplicationFilter</filter-class>  
     </filter>  
     <filter-mapping>  
     <filter-name>ApplicationFilter</filter-name>  
     <url-pattern>/\*</url-pattern>  
     </filter-mapping>  
      
     <!-- ========================================================================================= -->  
     <!-- Dump Namespace Servlet -->  
     <!-- ========================================================================================= -->  
     <servlet>  
     <servlet-name>DumpNamespaceServlet</servlet-name>  
     <servlet-class>guru.mmp.application.web.servlet.DumpNamespaceServlet</servlet-class>  
     </servlet>  
     <servlet-mapping>  
     <servlet-name>DumpNamespaceServlet</servlet-name>  
     <url-pattern>/servlet/DumpNamespaceServlet</url-pattern>  
     </servlet-mapping>  
      
     <!-- ========================================================================================= -->  
     <!-- Messaging Servlet -->  
     <!-- ========================================================================================= -->  
     <servlet>  
     <description>Messaging Servlet</description>  
     <servlet-name>MessagingServlet</servlet-name>  
     <servlet-class>guru.mmp.application.web.servlet.MessagingServlet</servlet-class>  
     </servlet>  
     <servlet-mapping>  
     <servlet-name>MessagingServlet</servlet-name>  
     <url-pattern>/servlet/MessagingServlet</url-pattern>  
     </servlet-mapping>  
      
     <!-- ========================================================================================= -->  
     <!-- Unit Test Servlet -->  
     <!-- ========================================================================================= -->  
     <servlet>  
     <servlet-name>UnitTestServlet</servlet-name>  
     <servlet-class>guru.mmp.application.web.servlet.UnitTestServlet</servlet-class>  
     <init-param>  
     <description>The comma delimited list of JUnit test classes</description>  
     <param-name>UnitTestClasses</param-name>  
     <param-value>  
     guru.mmp.application.test.RegistryTests,  
     guru.mmp.application.test.SecurityServiceTests  
     </param-value>  
     </init-param>  
     </servlet>  
     <servlet-mapping>  
     <servlet-name>UnitTestServlet</servlet-name>  
     <url-pattern>/servlet/UnitTestServlet</url-pattern>  
     </servlet-mapping>  
      
     <display-name>**mmp-sample-web**</display-name>  
      
     <welcome-file-list>  
     <welcome-file>index.html</welcome-file>  
     <welcome-file>index.htm</welcome-file>  
     <welcome-file>index.jsp</welcome-file>  
     <welcome-file>default.html</welcome-file>  
     <welcome-file>default.htm</welcome-file>  
     <welcome-file>default.jsp</welcome-file>  
     </welcome-file-list>  
      
    </web-app>
12. Setup the database script for the web application:
    1. Create a new directory named **database** under the **src** directory.
    2. Depending on the type of database being used, e.g. PostgreSQL, etc, copy the appropriate SQL file from the **mmp-java/src/mmp-application/database** folder to the folder created in the previous step using the naming convention **<ApplicationName><DatabaseType>.sql** e.g. SamplePostgres.sql.
    3. Modify the SQL file containing the DDL for the web application’s database as appropriate. Ensure that the **CREATE USER** and **DROP USER** statements reference the correct database users.
13. Setup the security configuration for the application.
    1. Create a new Java keystore (JKS) containing the following:  
         
        The public key for the application (certificate)  
        The private key for the application  
        The public keys for any certificate authorities
    2. Copy the Java keystore under the **src/META-INF** folder for the dynamic web project e.g. **src/META-INF/Sample.jks**.
    3. Create a new file under the **src/META-INF** folder for the dynamic web project, using the naming convention **<EARName>.ApplicationSecurity** e.g. Sample.ApplicationSecurity, with the following contents:  
         
       <?xml version="1.0" encoding="UTF-8"?>  
       <!DOCTYPE applicationSecurity SYSTEM " ApplicationSecurity.dtd">  
         
       <applicationSecurity >  
        <keyStoreName>***ApplicationName***.jks</keyStoreName>  
        <keyStoreAlias>***Application Certificate Alias***</keyStoreAlias>  
        <keyStorePassword>***Application Keystore Password***</keyStorePassword>  
       </applicationSecurity>  
         
       e.g.  
         
       <?xml version="1.0" encoding="UTF-8"?>  
       <!DOCTYPE applicationSecurity SYSTEM " ApplicationSecurity.dtd">  
         
       <applicationSecurity>  
        <keyStoreName>Sample.jks</keyStoreName>  
        <keyStoreAlias>Sample</keyStoreAlias>  
        <keyStorePassword>Password1</keyStorePassword>  
       </applicationSecurity>
14. Setup JPA for the web application:
    1. Create the following directory under the **src** directory for the JEE Web Application:  
         
       **web/src/main/resources/META-INF**   
         
       e.g. web/src/main/resources/META-INF
    2. Create a file named **persistence.xml** under the **web/src/main/java/META-INF** directory with the following contents, changing the entries in **red** as appropriate:  
         
       e.g. web/src/main/java/META-INF/persistence.xml  
         
       <?xml version="1.0" encoding="UTF-8"?>  
       <persistence xmlns="http://java.sun.com/xml/ns/persistence" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://java.sun.com/xml/ns/persistence http://java.sun.com/xml/ns/persistence/persistence\_2\_0.xsd" version="2.0">  
        <persistence-unit name="Application" transaction-type="JTA">  
        <jta-data-source>java:app/jdbc/ApplicationDataSource</jta-data-source>  
         
        <!-- Common Entities -->  
        <class>guru.mmp.application.persistence.VersionedId</class>  
        <class>guru.mmp.application.persistence.Location</class>  
         
        <!-- MLApplicationForms -->  
        <!-- UNCOMMENT WHEN INCLUDING THE MLApplicationForms library -->  
        <!--  
        <class>guru.mmp.application.forms.ArchivedForm</class>  
        <class>guru.mmp.application.forms.Document</class>  
        <class>guru.mmp.application.forms.DocumentDefinition</class>  
        <class>guru.mmp.application.forms.DocumentType</class>  
        <class>guru.mmp.application.forms.Form</class>  
        <class>guru.mmp.application.forms.FormAttribute</class>  
        <class>guru.mmp.application.forms.FormAttributeDefinition</class>  
        <class>guru.mmp.application.forms.FormAuditLogEntry</class>  
        <class>guru.mmp.application.forms.FormAuditLogEntrySummary</class>  
        <class>guru.mmp.application.forms.FormDefinition</class>  
        <class>guru.mmp.application.forms.FormDefinitionSummary</class>  
        <class>guru.mmp.application.forms.FormDocumentAssociation</class>  
        <class>guru.mmp.application.forms.FormProcessAssociation</class>  
        <class>guru.mmp.application.forms.FormSummary</class>  
        <class>guru.mmp.application.forms.FormTransform</class>  
        <class>guru.mmp.application.forms.FormTransformSummary</class>  
        <class>guru.mmp.application.forms.ProcessDefinition</class>  
        <class>guru.mmp.application.forms.ReportDefinition</class>  
        -->  
         
        <properties>  
        <property name="hibernate.dialect" value="org.hibernate.dialect.PostgreSQLDialect" />  
        <property name="hibernate.show\_sql" value="no" />  
        <property name="hibernate.ejb.use\_class\_enhancer" value="true"/>  
        </properties>  
        </persistence-unit>  
       </persistence>

## JEE Web Service

### Creating a new JEE web service

1. Create a new directory that will store all the project directories for the new JEE web service, e.g. SampleService, under the Subversion source control system.
2. Create a new directory called **Source** under the project directory created in step 1.
3. Start Eclipse using an appropriately configured workspace. If this is a new workspace then the standard instructions in the section on **Configuring Eclipse for Java development** should be followed before any projects are created.
4. Complete the following steps to create the **Dynamic Web Project** for the new JEE web application and add it to the Subversion source control system:
   1. Switch to the **Java EE** view.
   2. Select **File > New > Dynamic Web Project** from the menu.
   3. Enter the **name** of the Dynamic Web Project using the naming convention **<ServiceName>Web** e.g. SampleServiceWeb.
   4. Select the **Use default** option for the **Project contents**.
   5. Select **JBoss v5.0** as the **Target runtime**.
   6. Select **2.5** as the **Dynamic web module version**.
   7. Select **Default Configuration or JBoss v5.0** for the **Configuration**.
   8. Click **Next**.  
        
      
   9. Leave the default options for the Java configuration and click **Next**.  
        
      
   10. Change the **Context root** to the name of the service e.g. *SampleService*.
   11. Change the name of the **Content directory** to **web** and click **Finish**.  
         
       
   12. **Right-click** on the new dynamic web project in the **Project Explorer** and select **Team > Share Project**…
   13. Select **SVN** for the **repository type** and click **Next**.
   14. Select one of the **existing repository locations** that has been registered with the Eclipse workspace and click **Next**.  
         
       
   15. Select the option to **Use specified folder name** and click **Select…** to browse the Subversion repository and select the **Source** directory created in step 2. Click **OK** once you have selected the directory.  
         
       
   16. Click **Finish**.  
         
       
5. Complete the following steps to create the **Enterprise Application Project** for the new JEE web application and add it to the Subversion source control system:
   1. Switch to the **Java EE** view.
   2. Select **File > New > Enterprise Application Project** from the menu.
   3. Enter the **name** of the Enterprise Application Project using the naming convention **<ServiceName>** e.g. SampleService.
   4. Select the **Use default** option for the **Project contents**.
   5. Select **JBoss v5.0** as the **Target runtime**.
   6. Select **5.0** as the **EAR version**.
   7. Select **Default Configuration or JBoss v5.0** for the **Configuration**.
   8. Click **Next**.  
        
      
   9. Select the Java EE module for the **Dynamic Web Project** created previously.
   10. Enter **content** as the name of the **Content directory**.
   11. Select the option to **Generate application.xml deployment descriptor**.
   12. Click **Finish**.  
         
       
   13. **Right-click** on the new enterprise application project in the **Project Explorer** and select **Team > Share Project**…
   14. Select **SVN** for the **repository type** and click **Next**.
   15. Select one of the **existing repository locations** that have been registered with the Eclipse workspace and click **Next**.  
         
       
   16. Select the option to **Use specified folder name** and click **Select…** to browse the Subversion repository and select the **Source** directory created in step 2. Click **OK** once you have selected the directory.  
         
       
   17. Click **Finish**.  
         
       
6. Add the following M-Logic infrastructure projects to the workspace:  
     
   **MLCommon  
   MLService**  
   **MLWeb**
7. Associate the M-Logic infrastructure projects with the new JEE web application by completing the following steps:
   1. Right-click on the **Enterprise Application Project** for the new JEE web application and select **Properties**.
   2. Select **Deployment Assembly** from the left-hand menu.
   3. Click **Add…**  then select **Project** from the list and click **Next >**.
   4. Select the **MLCommon, MLService** and **MLWeb** projects from the list of project and click **Finish**.  
        
      **NOTE:** This will automatically select the dependencies for the MLCommon, MLService and MLWeb projects.
   5. Select **Project References** from the left-hand menu.
   6. Add the **MLCommon** and **MLWeb** projects as **Project References**.  
        
      
   7. Click **OK**.
   8. Right-click on the **Dynamic Web Project** for the new JEE web application and select **Properties**.
   9. Select **Project References** from the left-hand menu.
   10. Add the **MLCommon**, **MLService** and **MLWeb** projects as **Project References**.
   11. Click **OK**.
8. Create the Java class that will implement the JAX-WS service.
   1. Create a package under the **Java Resources: src** node for the **Dynamic Web Project** using the following naming convention:  
        
      **<CompanyPrefix>.services.<ServiceName>.ws**  
        
      e.g. guru.mmp.services.sample.ws.  
        
      
   2. Create a new Java class using the naming convention **<ServiceName>** e.g. SampleService with contents similar to the following under the package created in the previous step:  
        
      package guru.mmp.services.sample.ws;  
        
      import javax.jws.HandlerChain;  
      import javax.jws.WebMethod;  
      import javax.jws.WebService;  
        
      /\*\*  
       \* The <code>SampleService</code> class implements the Sample Service.  
       \*/  
      @HandlerChain(file = "/WebServiceSecurity.xml")  
      @WebService(  
       name = "ISampleService",  
       portName = "SampleService",  
       serviceName = "SampleService",   
       targetNamespace = "http://ws.sample.services.mlogic.biz")   
      public class SampleService  
      {  
       /\*\*  
       \* Returns the service version.  
       \*  
       \* @return the service version  
       \*/  
       @WebMethod  
       public String getVersion()  
       {  
       return "1.0.0";  
       }  
      }
9. Setup the security configuration for the web service.
   1. Create a new Java keystore (JKS) containing the following:  
        
       The public key for the web service (certificate)  
       The private key for the web service  
       The public keys for any certificate authorities
   2. Copy the Java keystore under the **src/META-INF** folder for the dynamic web project e.g. **src/META-INF/SampleService.jks**.
   3. Create a new file under the **src/META-INF** folder for the dynamic web project, using the naming convention **<ServiceName>.ServiceSecurity** e.g. SampleService.ServiceSecurity, with the following contents:  
        
      <?xml version="1.0" encoding="UTF-8"?>  
      <!DOCTYPE ServiceSecurity:Config SYSTEM "ServiceSecurity.dtd">  
      <ServiceSecurity:Config>  
       <KeyStoreName>***ServiceName***.jks</KeyStoreName>  
       <KeyStoreAlias>***Service Certificate Alias***</KeyStoreAlias>  
       <KeyStorePassword>***Service Keystore Password***</KeyStorePassword>  
      </ServiceSecurity:Config>  
        
      e.g.  
        
      <?xml version="1.0" encoding="UTF-8"?>  
      <!DOCTYPE ServiceSecurity:Config SYSTEM "ServiceSecurity.dtd">  
      <ServiceSecurity:Config>  
       <KeyStoreName>SampleService.jks</KeyStoreName>  
       <KeyStoreAlias>Sample Service</KeyStoreAlias>  
       <KeyStorePassword>Password1</KeyStorePassword>  
      </ServiceSecurity:Config>
10. Update the **web.xml** configuration file for the dynamic web project.
    1. Add the following **listener** entry to the **web.xml** configuration file for the dynamic web project:  
         
       <listener>  
        <description>M-Logic Secure Web Service Listener</description>  
        <listener-class>guru.mmp.common.service.ws.security.SecureWebServiceListener</listener-class>  
       </listener>
    2. Add the **context parameter**,which references the web service security configuration file for the web service, to the **web.xml** configuration file for the dynamic web project:  
         
       <context-param>  
        <description>M-Logic Service Security Context</description>  
        <param-name>guru.mmp.common.security.context.ServiceSecurityContext</param-name>  
        <param-value>SampleService</param-value>  
       </context-param>  
         
       **NOTE:** The parameter value should be the name of the web service security configuration file without the **.ServiceSecurity** extension.
    3. Add the following **servlet** entries to the **web.xml** configuration file for the dynamic web project changing the URL pattern as appropriate:  
         
       <servlet>  
        <servlet-name>SecureWebServiceServlet</servlet-name>  
        <servlet-class>guru.mmp.common.service.ws.security.SecureWebServiceServlet</servlet-class>  
        <load-on-startup>1</load-on-startup>  
       </servlet>  
       <servlet-mapping>  
        <servlet-name>SecureWebServiceServlet</servlet-name>  
        <url-pattern>/services/SampleService</url-pattern>  
       </servlet-mapping>
11. Create the Metro JAX-WS web services configuration file **/web/WEB-INF/sun-jaxws.xml** with the following contents:  
      
    <?xml version="1.0" encoding="UTF-8"?>  
    <endpoints xmlns="http://java.sun.com/xml/ns/jax-ws/ri/runtime" version="2.0">  
     <endpoint url-pattern="/services/**<ServiceName>**" name="**<ServiceName>**"  
     implementation="**<ServiceClassName>**"/>  
    </endpoints>  
      
    e.g.  
      
    <?xml version="1.0" encoding="UTF-8"?>  
    <endpoints xmlns="http://java.sun.com/xml/ns/jax-ws/ri/runtime" version="2.0">  
     <endpoint url-pattern="/services/SampleService" name="SampleService"  
     implementation="guru.mmp.services.sample.ws.SampleService"/>  
    </endpoints>