L318: AEM coding errors

Table of Contents

Lab Overview

Scope

- Deploy your application via the command line
- Import your application into the Eclipse tooling
- Optimize code to make best of the AEM and Sling functionality

Lab machine tip/tricks

Eclipse-shortcuts

- Organize imports: CMD+SHIFT+O
- Formatting: CMD+SHIFT+F
- Toggle comments: CMD+SHIFT+C
- Auto-complete: CTRL+SPACE
- Finding resources: CTRL + SHIFT + R
- Finding Java-classes: CTRL + SHIFT + T

Lesson 0 – Getting the code

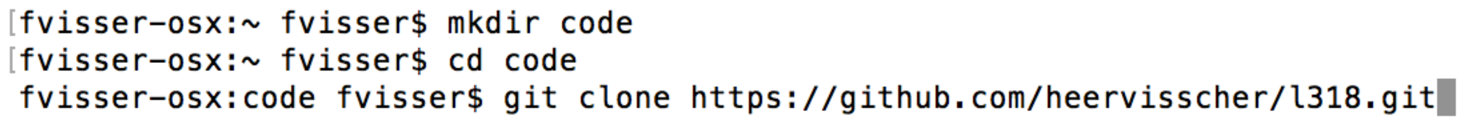
Timing: 10 min

Objective

Get the code from the public github repository

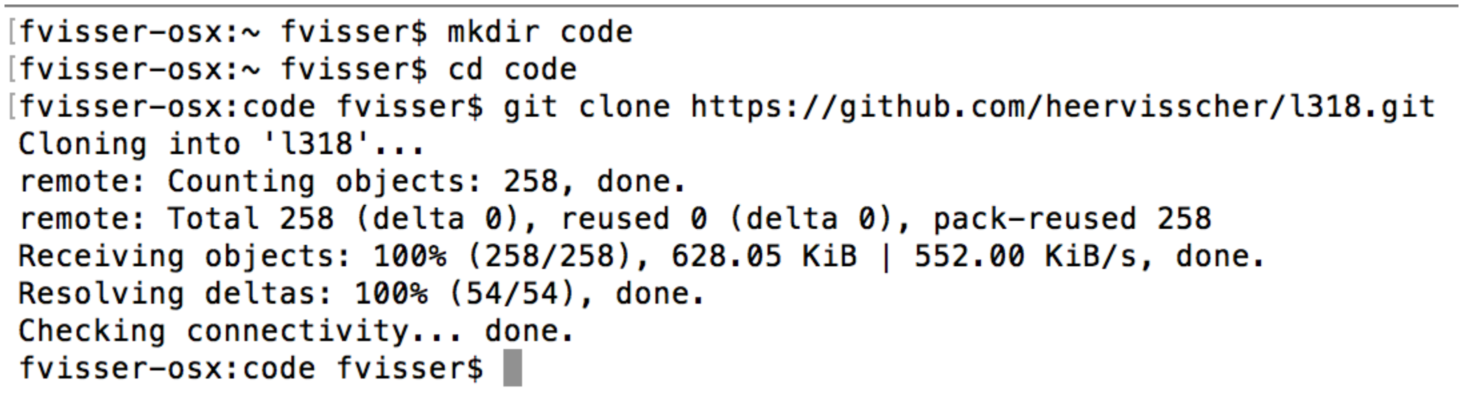
Steps

1. Open the terminal window and create a new directory: mkdir code
2. Go to this directory via: cd code
3. Now you can clone the git repository: git clone https://github.com/heervisscher/l318.git



```
[fvisser-osx:~ fvisser$ mkdir code
[fvisser-osx:~ fvisser$ cd code
 fvisser-osx:code fvisser$ git clone https://github.com/heervisscher/l318.git
```

You will see now that the code has been cloned into your environment



```
[fvisser-osx:~ fvisser$ mkdir code
[fvisser-osx:~ fvisser$ cd code
[fvisser-osx:code fvisser$ git clone https://github.com/heervisscher/l318.git
 Cloning into 'l318'...
 remote: Counting objects: 258, done.
 remote: Total 258 (delta 0), reused 0 (delta 0), pack-reused 258
 Receiving objects: 100% (258/258), 628.05 KiB | 552.00 KiB/s, done.
 Resolving deltas: 100% (54/54), done.
 Checking connectivity... done.
 fvisser-osx:code fvisser$
```

Lesson 1 – Setting up development environment

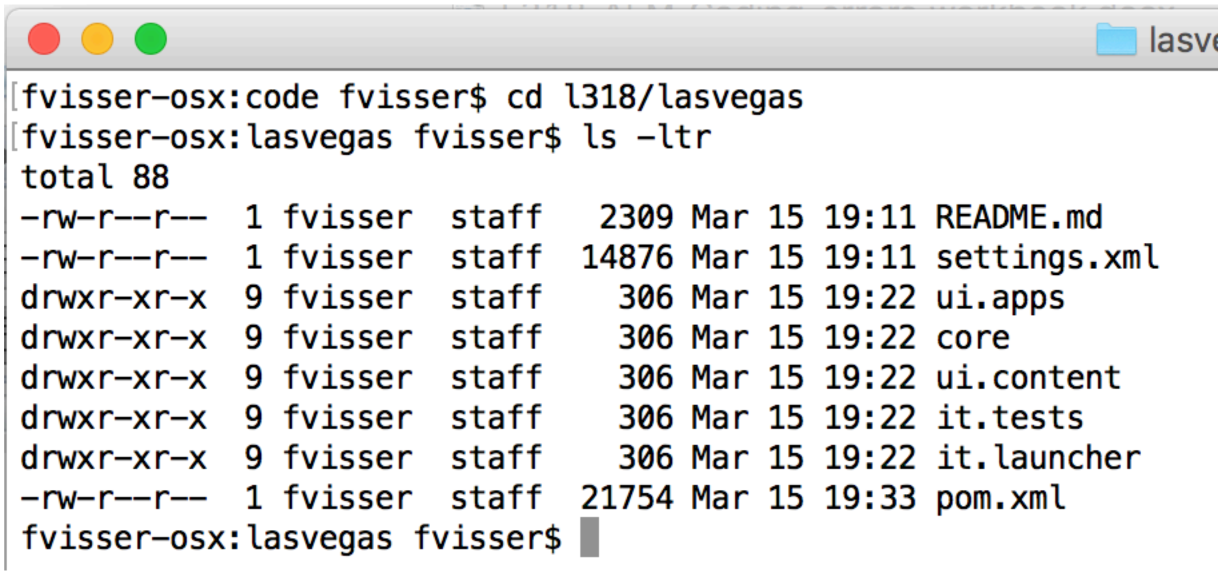
Timing: 15 min

Objective

Deploy the code from the command-line, and import the project into your Eclipse environment.

Steps

Open the terminal window, and go to the "l318/lasvegas" directory: cd l318/lasvegas

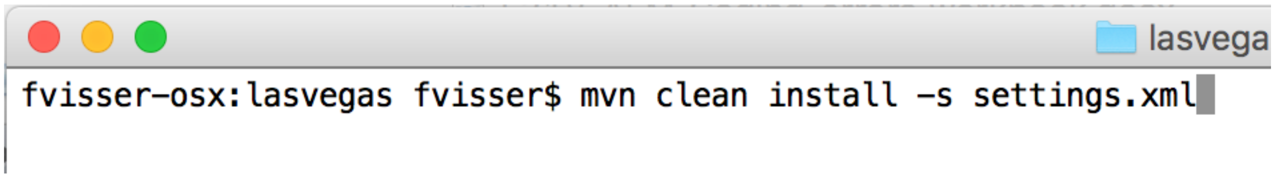


```
[fvisser-osx:code fvisser$ cd l318/lasvegas
[fvisser-osx:lasvegas fvisser$ ls -ltr
total 88
-rw-r--r--  1 fvisser  staff   2309 Mar 15 19:11 README.md
-rw-r--r--  1 fvisser  staff  14876 Mar 15 19:11 settings.xml
drwxr-xr-x  9 fvisser  staff    306 Mar 15 19:22 ui.apps
drwxr-xr-x  9 fvisser  staff    306 Mar 15 19:22 core
drwxr-xr-x  9 fvisser  staff    306 Mar 15 19:22 ui.content
drwxr-xr-x  9 fvisser  staff    306 Mar 15 19:22 it.tests
drwxr-xr-x  9 fvisser  staff    306 Mar 15 19:22 it.launcher
-rw-r--r--  1 fvisser  staff  21754 Mar 15 19:33 pom.xml
fvisser-osx:lasvegas fvisser$
```

Building the project

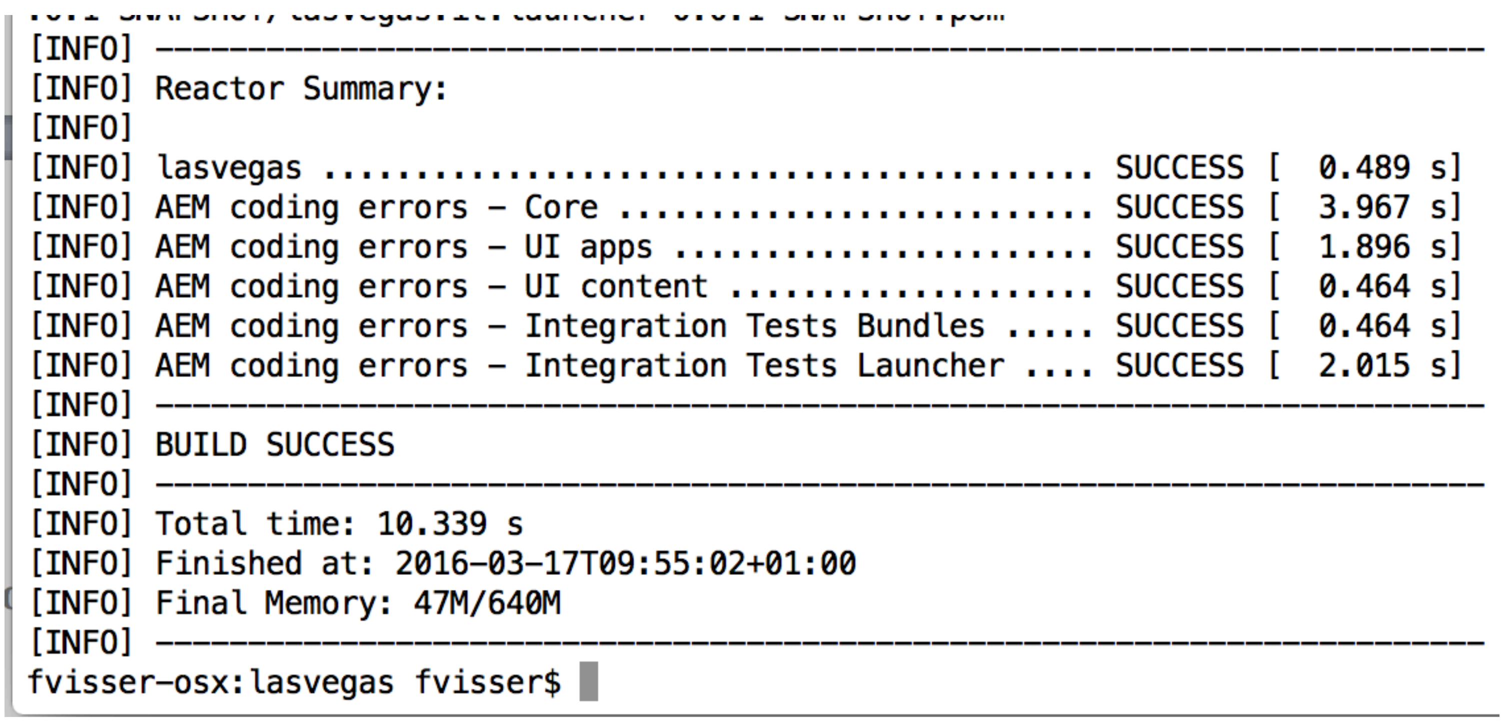
We first going to build the project via the command line, this goes via this command:

mvn clean install –s settings.xml (make sure you are in the directory with the pom.xml and README.md)



```
fvisser-osx:lasvegas fvisser$ mvn clean install -s settings.xml
```

When the project builds well you will have the following output.



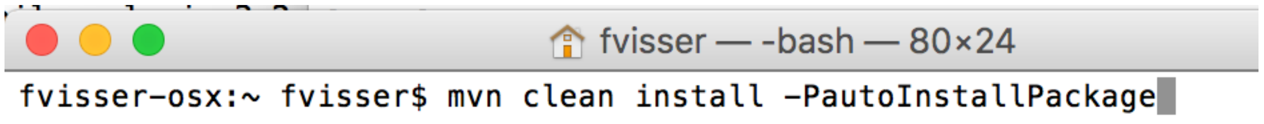
```
[INFO] ------------------------------------------------------------------------
[INFO] Reactor Summary:
[INFO]
[INFO] lasvegas ........................................... SUCCESS [  0.489 s]
[INFO] AEM coding errors - Core ........................... SUCCESS [  3.967 s]
[INFO] AEM coding errors - UI apps ........................ SUCCESS [  1.896 s]
[INFO] AEM coding errors - UI content ..................... SUCCESS [  0.464 s]
[INFO] AEM coding errors - Integration Tests Bundles ..... SUCCESS [  0.464 s]
[INFO] AEM coding errors - Integration Tests Launcher .... SUCCESS [  2.015 s]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 10.339 s
[INFO] Finished at: 2016-03-17T09:55:02+01:00
[INFO] Final Memory: 47M/640M
[INFO] ------------------------------------------------------------------------
fvisser-osx:lasvegas fvisser$
```

Deploying to your local AEM-instance

Now that the project is building correctly, we will deploy this onto the local AEM-instance, running on port 4510

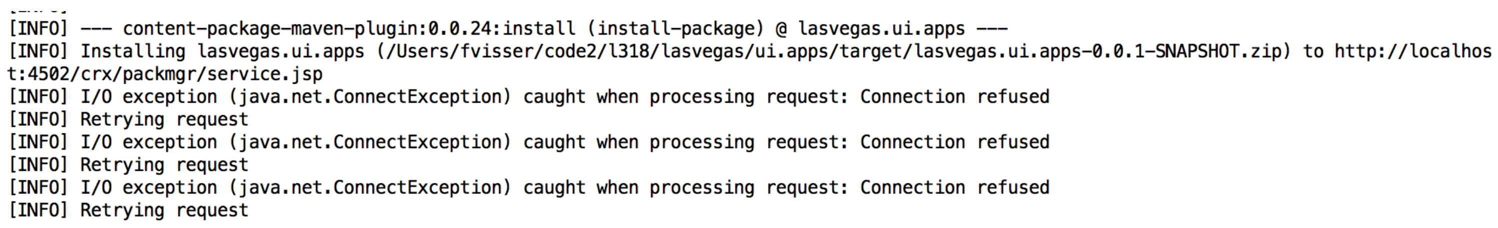
Normally this will go with the command (also check this in the README.md):

mvn clean install –PautoInstallPackage (this will be deployed to port 4502)



```
fvisser-osx:~ fvisser$ mvn clean install -PautoInstallPackage
```

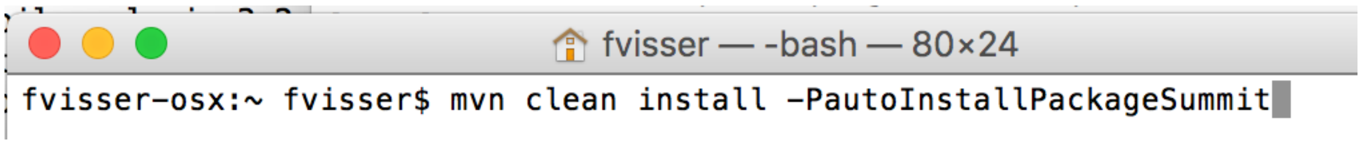
After this deployment you will see these errors:



```
[INFO] --- content-package-maven-plugin:0.0.24:install (install-package) @ lasvegas.ui.apps ---
[INFO] Installing lasvegas.ui.apps (/Users/fvisser/code2/l318/lasvegas/ui.apps/target/lasvegas.ui.apps-0.0.1-SNAPSHOT.zip) to http://localhos
t:4502/crx/packmgr/service.jsp
[INFO] I/O exception (java.net.ConnectException) caught when processing request: Connection refused
[INFO] Retrying request
[INFO] I/O exception (java.net.ConnectException) caught when processing request: Connection refused
[INFO] Retrying request
[INFO] I/O exception (java.net.ConnectException) caught when processing request: Connection refused
[INFO] Retrying request
```

To deploy to port 4510 there is the following profile available autoInstallPackageSummit:

mvn clean install –PautoInstallPackageSummit


fvisser — -bash — 80×24

```
fvisser-osx:~ fvisser$ mvn clean install -PautoInstallPackageSummit
```

CHECKPOINT 1, how can we do this better**...**

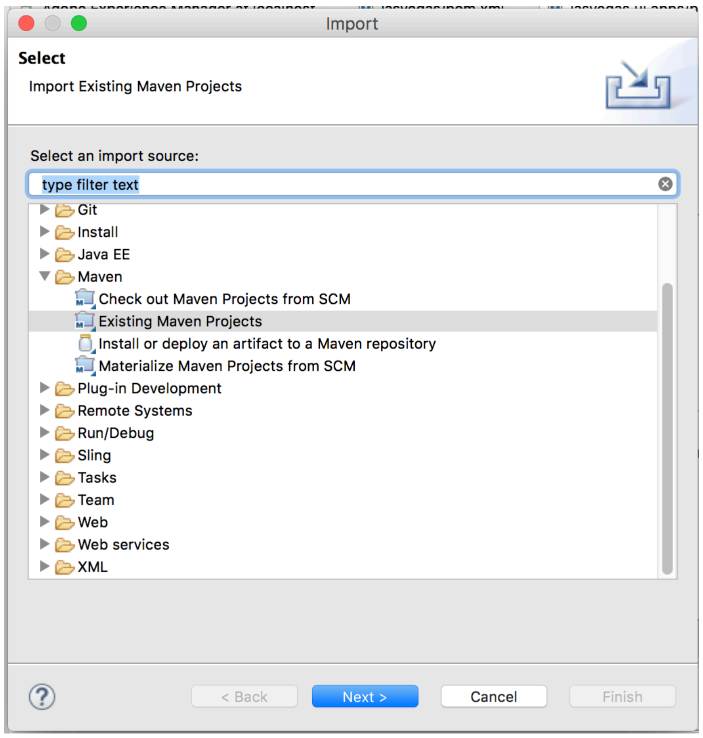
NOTE: In the pom.xml of the ui.apps project a new profile is created, using a new property called aem.summit.port (defined in the parent pom.xml).

Importing project into Eclipse

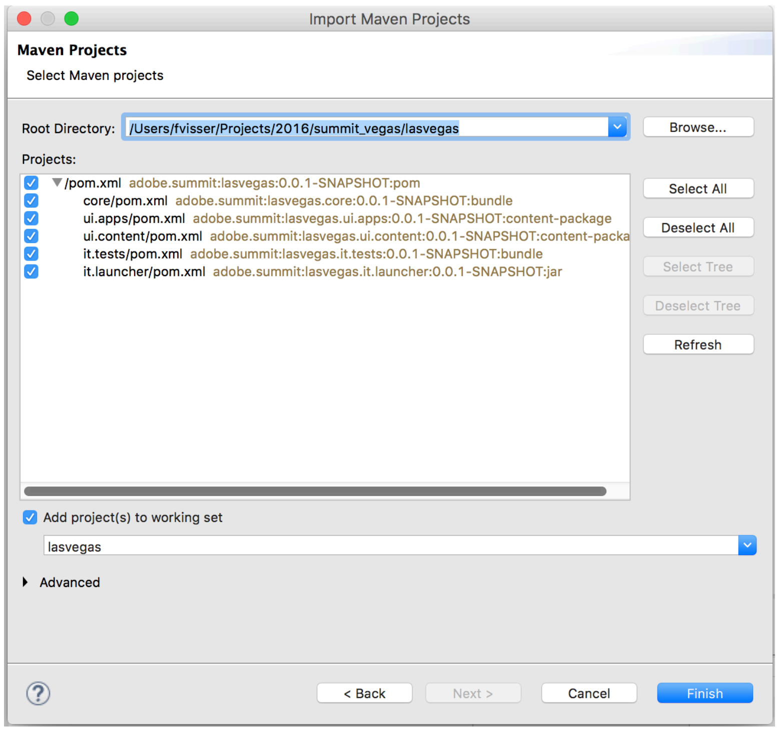
Open Eclipse via the following icon

If a popup comes to choose a workspace, you can click “Ok” on that popup.

To import a project choose the option File -> Import… → Maven → Existing maven projects



Then select the directory where the code is located, then you should see the following screen to import the project.


Import Maven Projects

Maven Projects

Select Maven projects

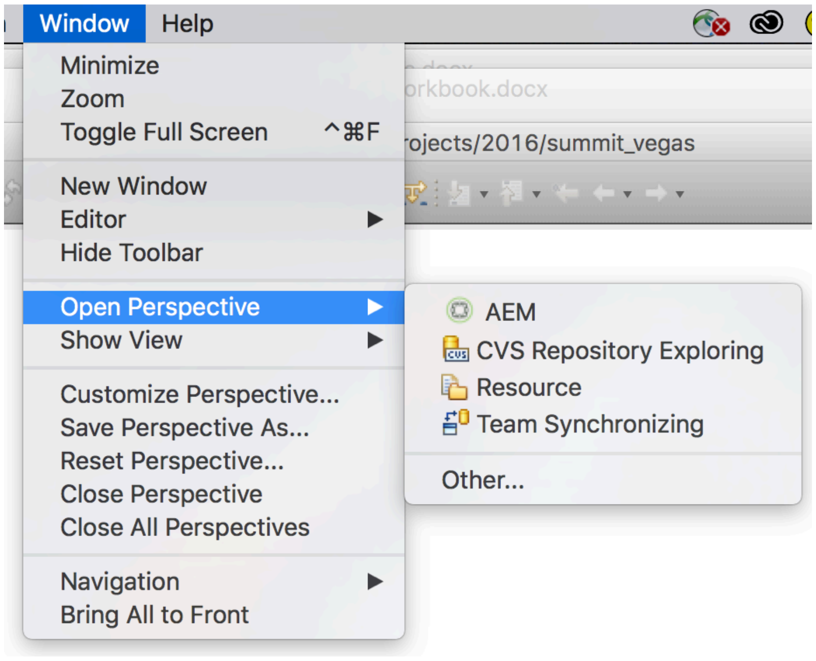
Root Directory: /Users/fvisser/Projects/2016/summit_vegas/lasvegas

Projects:

```
☑ ▼/pom.xml  adobe.summit:lasvegas:0.0.1-SNAPSHOT:pom
☑     core/pom.xml  adobe.summit:lasvegas.core:0.0.1-SNAPSHOT:bundle
☑     ui.apps/pom.xml  adobe.summit:lasvegas.ui.apps:0.0.1-SNAPSHOT:content-package
☑     ui.content/pom.xml  adobe.summit:lasvegas.ui.content:0.0.1-SNAPSHOT:content-packa
☑     it.tests/pom.xml  adobe.summit:lasvegas.it.tests:0.0.1-SNAPSHOT:bundle
☑     it.launcher/pom.xml  adobe.summit:lasvegas.it.launcher:0.0.1-SNAPSHOT:jar
```

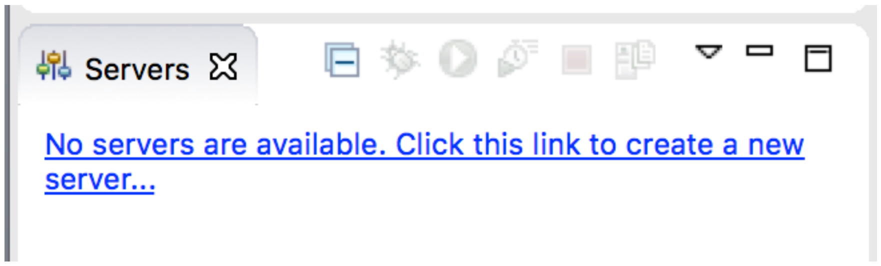
☑ Add project(s) to working set: lasvegas

Switch to the AEM-perspective (Window → Open Perspective → AEM)



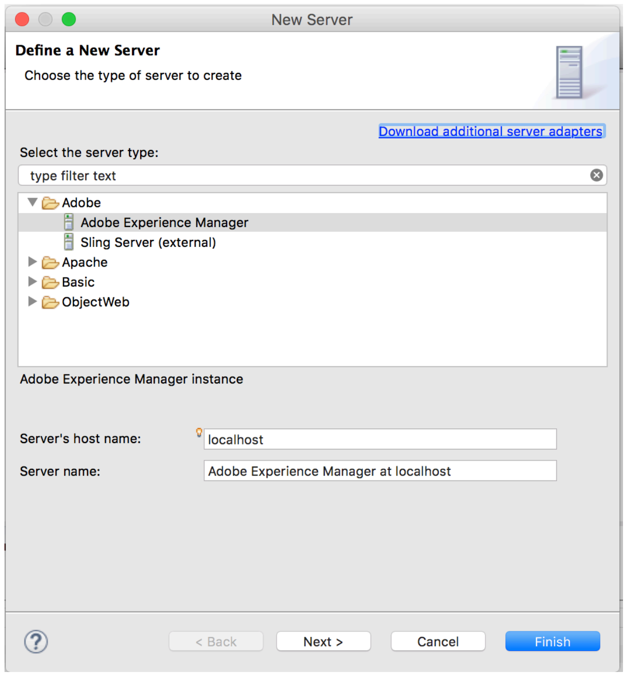
This will activate the AEM-tooling into your Eclipse environment.

We will now create a server connection:



No servers are available. Click this link to create a new server...

Click on this link to create a new server, the first you can have the default values



Define a New Server

Choose the type of server to create

Select the server type:

type filter text

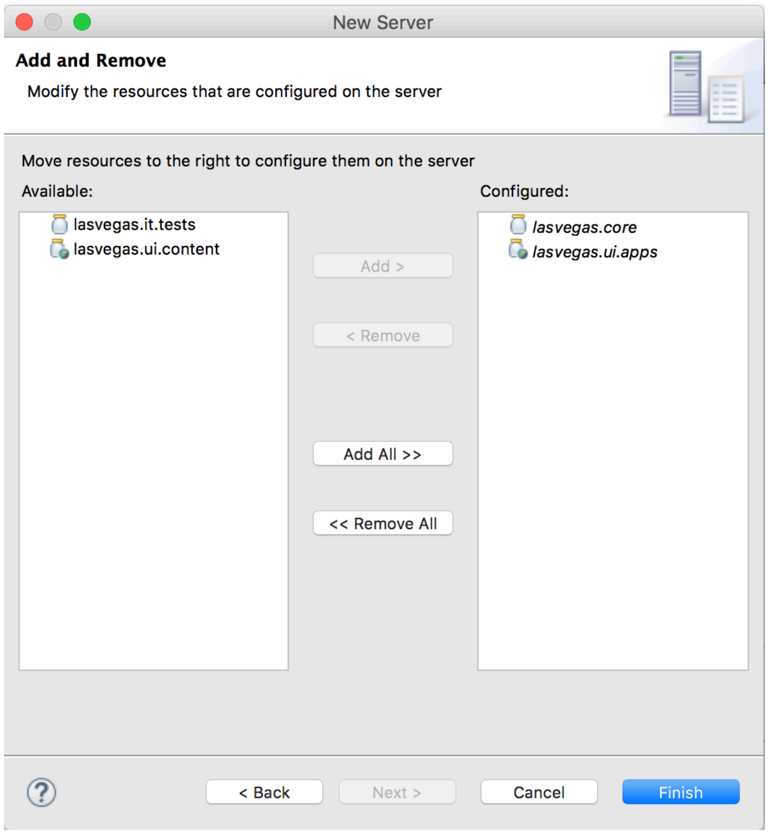
- Adobe
  - Adobe Experience Manager
  - Sling Server (external)
- Apache
- Basic
- ObjectWeb

Adobe Experience Manager instance

Server's host name: localhost

Server name: Adobe Experience Manager at localhost

In the next step, select core and apps as the modules.



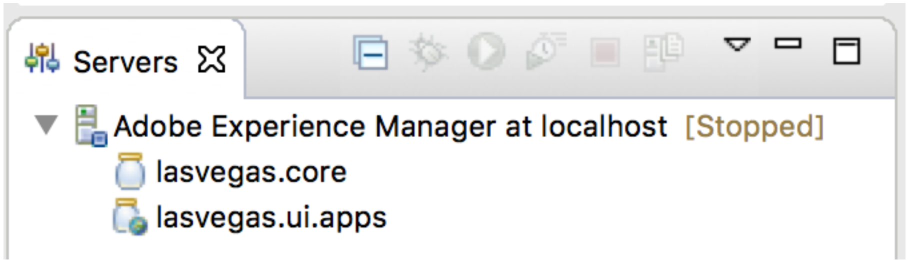
Add and Remove

Modify the resources that are configured on the server

Move resources to the right to configure them on the server

Available:	Configured:
lasvegas.it.tests	*lasvegas.core*
lasvegas.ui.content	*lasvegas.ui.apps*

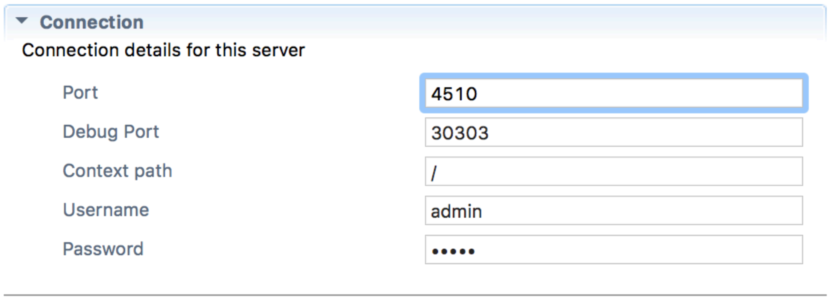
Now the server has been configured with the two modules



```
Adobe Experience Manager at localhost  [Stopped]
    lasvegas.core
    lasvegas.ui.apps
```

By default the port is configured at 8080, so we need to change this to 4510.

Double-click “Adobe Experience Manager at localhost”, then we can change the port to 4510.



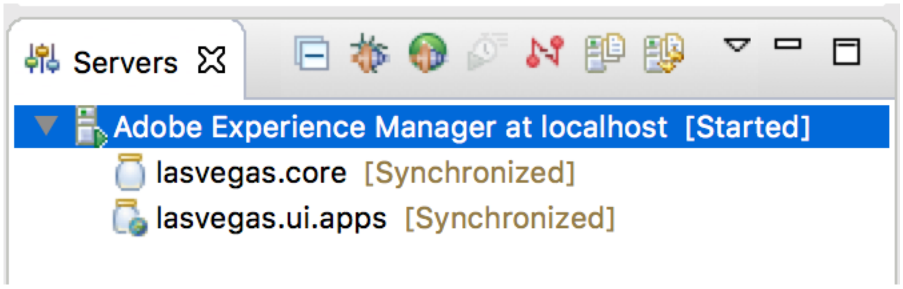
Connection

Connection details for this server

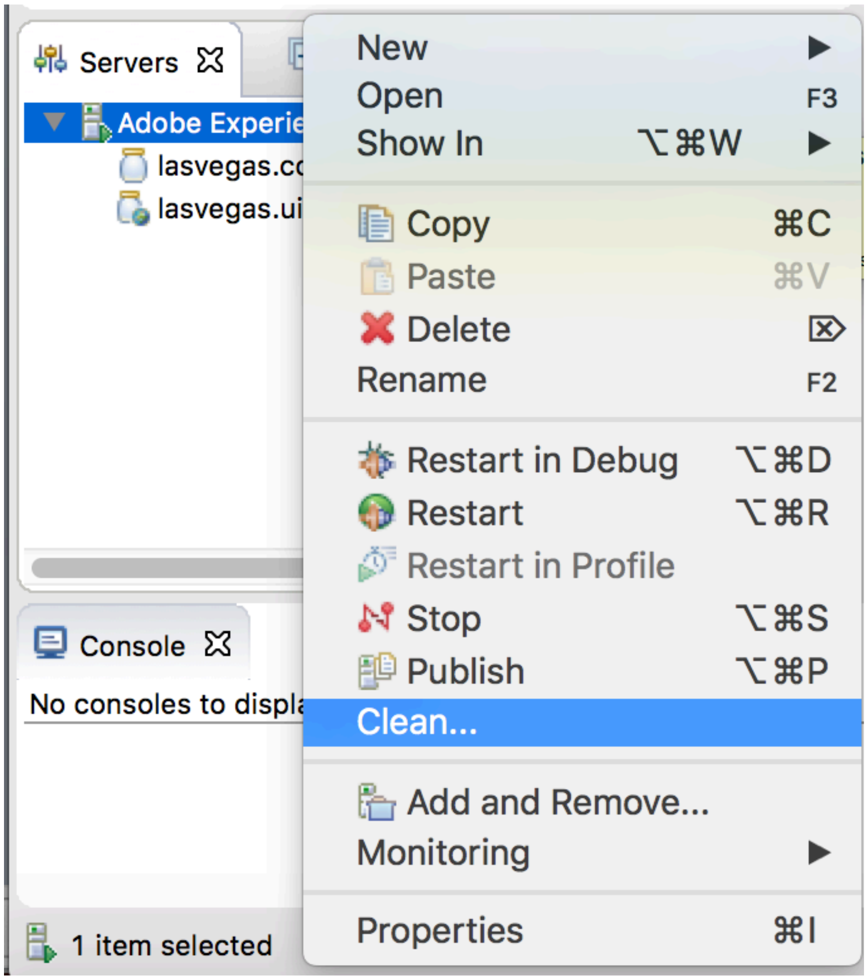
Port	4510
Debug Port	30303
Context path	/
Username	admin
Password	•••••

Make sure to save the changes after the change.

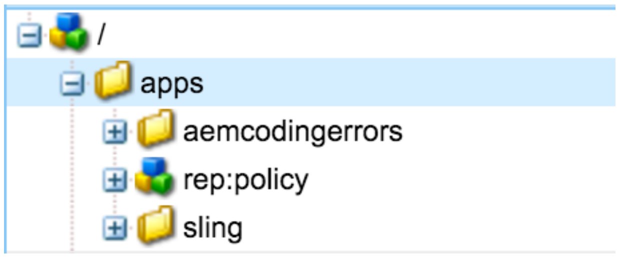
Now start the sever, this will automatically deploy the “core” and “apps” module into the AEM-instance.


Servers
Adobe Experience Manager at localhost [Started]
lasvegas.core [Synchronized]
lasvegas.ui.apps [Synchronized]

To make sure the modules are installed, select the “clean…” option



Final step is to validate this in crx-de lite (http://localhost:4510/crx/de/index.jsp)



This makes sure your Eclipse is synced with AEM, and any changes made in Eclipse are synced automatically to AEM.

Lesson 2 – Project validation

Objective

Goal of this lesson is to make sure the project is up-to-date with the Java and AEM-version used. A lot of times this is forgotten or pushed back till the end of the project.

Steps

First thing we want to check is whether the application is working, open the following url:

http://localhost:4510/editor.html/content/aemcodingerrors/en.html

Areas that we will focus on:

- Changing the Java-version of your project
- Using the correct maven dependency for AEM6.1

Java-version

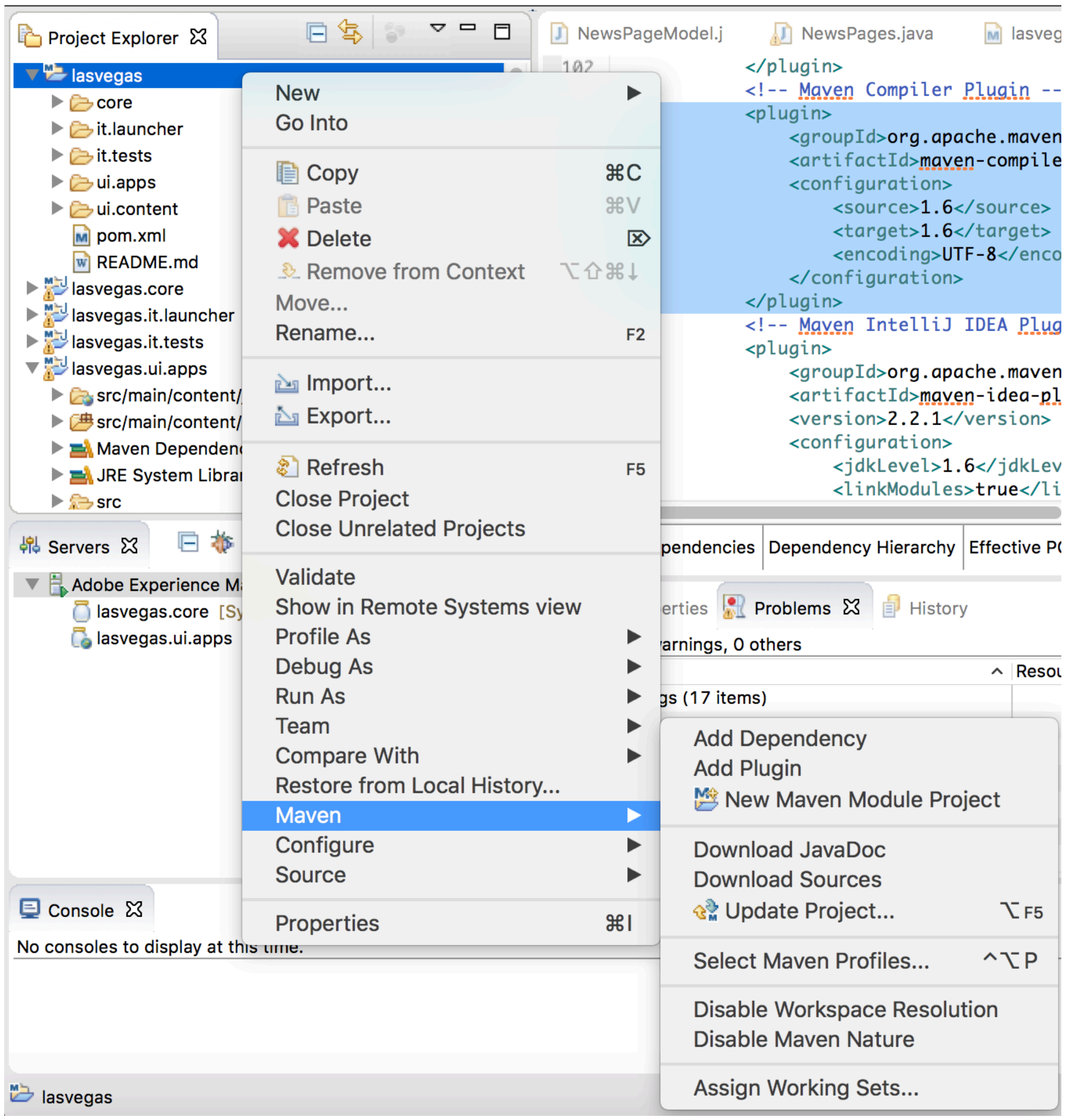
In the pom.xml of the main-project (lasvegas), you see the following notation:

```
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-compiler-plugin</artifactId>
    <configuration>
        <source>1.6</source>
        <target>1.6</target>
        <encoding>UTF-8</encoding>
    </configuration>
</plugin>
```

This indicates that the Java-code will be targeted for Java 1.6, as of AEM6.1 we can use Java 8. So let’s change this to

```
     <plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-compiler-plugin</artifactId>
    <configuration>
        <source>1.8</source>
        <target>1.8</target>
        <encoding>UTF-8</encoding>
    </configuration>
</plugin>
```

After this you need to do an “Update Project”, to make the changes are reflected



```
            </plugin>
            <!-- Maven Compiler Plugin --
            <plugin>
                <groupId>org.apache.maven
                <artifactId>maven-compile
                <configuration>
                    <source>1.6</source>
                    <target>1.6</target>
                    <encoding>UTF-8</enco
                </configuration>
            </plugin>
            <!-- Maven IntelliJ IDEA Plug
            <plugin>
                <groupId>org.apache.maven
                <artifactId>maven-idea-pl
                <version>2.2.1</version>
                <configuration>
                    <jdkLevel>1.6</jdkLev
                    <linkModules>true</li
```

After this change also some Warnings will have gone from your console, this because Java 1.6 was not available on the machine.

AEM6.1 Maven dependency (Optional, this is already applied on the machine)

General info: https://docs.adobe.com/docs/en/aem/6-1/develop/dev-tools/ht-projects-maven.html

Open the pom.xml of your "lasvegas-core" project, you will see a dependency for the uber-jar

```
<dependency>
    <groupId>com.adobe.aem</groupId>
    <artifactId>aem-api</artifactId>
 </dependency>
```

This relates to this dependency in the "lasvegas" project

```
<dependency>
    <groupId>com.adobe.aem</groupId>
    <artifactId>aem-api</artifactId>
    <version>6.0.0.1</version>
    <scope>provided</scope>
   </dependency>
```

As we can see here, this points to 6.0, while we are developing against 6.1. Therefore we need to change this to the dependency already added in your main pom.xml:

```
<dependency>
    <groupId>com.adobe.aem</groupId>
    <artifactId>uber-jar</artifactId>
    <version>6.1.0</version>
    <classifier>obfuscated-apis</classifier>
    <scope>provided</scope>
</dependency>
```

To make the switch you need to comment/remove the following element in the "lasvegas-core" project

```
<dependency>
        <groupId>com.adobe.aem</groupId>
        <artifactId>aem-api</artifactId>
</dependency>
```

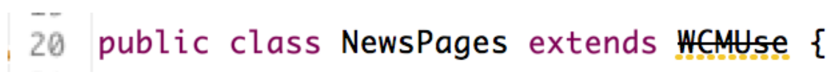
And you need to enable the following element:

```
<!-- <dependency> -->
<!-- <groupId>com.adobe.aem</groupId> -->
<!-- <artifactId>uber-jar</artifactId> -->
<!-- <classifier>obfuscated-apis</classifier> -->
<!-- </dependency> -->
```

Shortcut for commenting / uncommenting in Eclipse: CMD+SHIFT+C

Formatting the file: CMD+SHIFT+F

To validate this change you should see now the following deprecation warning in NewsPages.java



```
public class NewsPages extends ~~WCMUse~~ {
```

Lesson 3 Optimize Java-code

Objective

The page list component uses NewsPages-class to prepare the information to be displayed on the page.

Java-code is located in NewsPages and NewsPageModel

Sightly-code is located in /apps/aemcodingerrors/components/structure/page/partials/main.html

We will focus here to optimize the following elements

- Deprecated API usage
- Using @Reference inside WCMUse-objects
- Using admin-sessions in your components
- Constant classes available
- XSS escaping in Sightly
- Use of PageFilters
- Use of selectors

Steps:

During the session we will work together to optimize the code here.

Lesson 4 Using of servlets

Objective

In the provided source code there is a servlet (VegasServlet) that retrieves information around the requested page (via an url parameter).

We will focus in this course on:

- Disabling servlets in different runmodes
- Use of resourceTypes in the servlet-declaration
- Extending the of the right servlet classes

Steps:

During the session we will work together to optimize the code here.

Lesson 5: Using of Resource Merger

Objective

When using or overriding the default components, you don’t need to fully copy the components anymore. Via the Sling Resource Merger you can pick and choose the elements / properties you want to change.

Note: this is only available via the TouchUI

Steps

In the application you see a component ‘Summary list’ (/apps/aemcodingerrors/components/content/newslist) component where basically the whole cq:dialog is copied from the default list-component.

Together we will see how to extend the dialog without copying the whole dialog.