Setting up a Maven Project

This documentation describes how to set up a Maven project for CaptainCasa, using Eclipse (version “Neon”) as development environment.

We recommend to use the latest Eclipse version (J2EE installation of Eclipse): with Eclipse Luna we found quite some problems for running the Maven project inside Eclipse – the situation has much improved with the “Neon” version.

When creating and configuring the Eclipse project, please note: for quite some long time when going through the steps you will see problems showing up within the “Problems” view of Eclipse. Please ignore this problems when going through the steps – they will be resolved quite at the end...!

Overview

Dependencies

From a project's point of view a CaptainCasa Enterprise Client project is a web project. So the goal is to create some “.war”-structure.

CaptainCasa has two relevant artifacts:

- **eclntjsfserver.jar** – This is the jar file that you require for developing your server side classes. This jar contains the server-side components, the PageBean-management, the Workplace-management, etc. etc. So this jar is required from your side to build your Java-classes that are the ones you plug to the dialog processing. The jar is one to be part of WEB-INF/lib within the .war structure.
- **webappaddons** – This is a directory with non-Java files. It contains the standard .jsp dialog definitions (e.g. for OK, YesNo-Popup, Workplace-dialogs, etc.), images for these standard dialogs - and it contains all the client processing, which for the RISC-HTML client is all the JavaScript that is later on executed on client side. All this content is not relevant for developing your Java-classes, but it is relevant to run CaptainCasa Enterprise Client as part of your web application.

As consequence there are two CaptainCasa artifacts that you depend from: a compile time dependency for the eclntjsfserver.jar file, and an install time dependency to the webappaddons-content.

Project <==> Deployment Tomcat

By default the CaptainCasa installation comes with an own Tomcat instance running the application that you build. And it comes with some tooling (Layout Editor), that manages this Tomcat:

- Within the tools you “reload” (or “hot deploy”) the application. This means that actually the result of your project is copied into the corresponding Tomcat/webapps-application and that the application somehow is restarted so that changes take effect.

Of course you may also think about running some Tomcat instance which is managed by your development environment (here: Eclipse). But this is not the CaptainCasa-default scenario, so this documentation assumes to develop your project on the left side – and run your project in some Tomcat on the right side, having the CaptainCasa toolset as linkage between.

In other words:

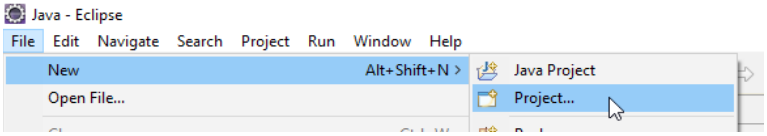
- Maven/Eclipse are the ones to produce the ".war"-like structure into the target directory of the project.
- The CaptainCasa toolset is the one to take over this data into the Tomcat runtime.

The result is a very efficient way of developing user interfaces – which is not burdened by always running full Maven builds and by always deploying full systems to the Tomcat instance.

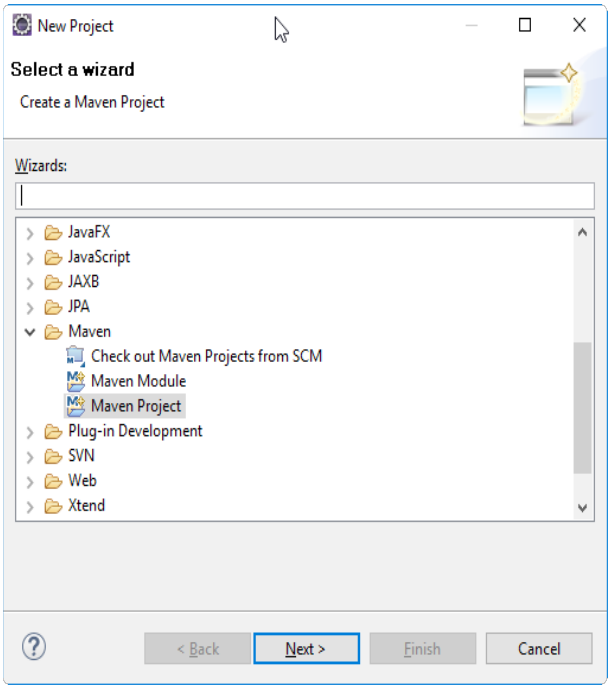
Creating and configuring the Eclipse project

Create the Maven project in Eclispe

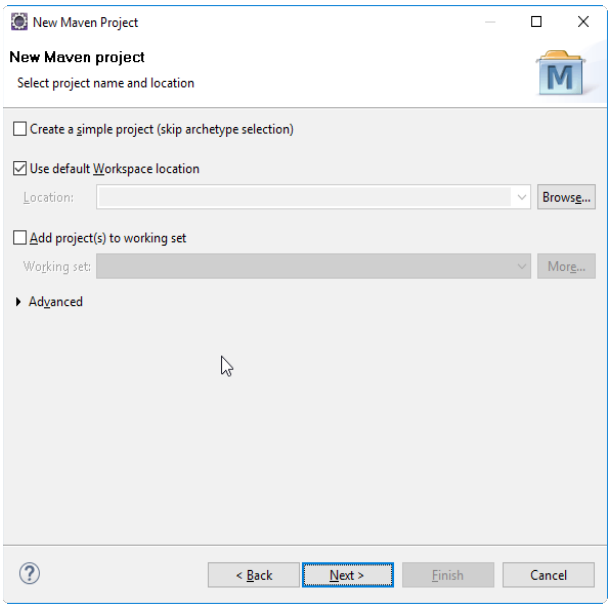
The project is created in Eclipse. Select File => New => Project...:



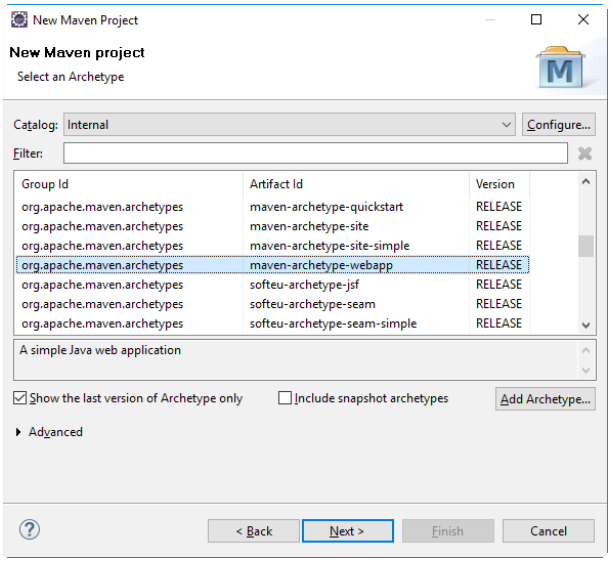
Select "Maven Project":



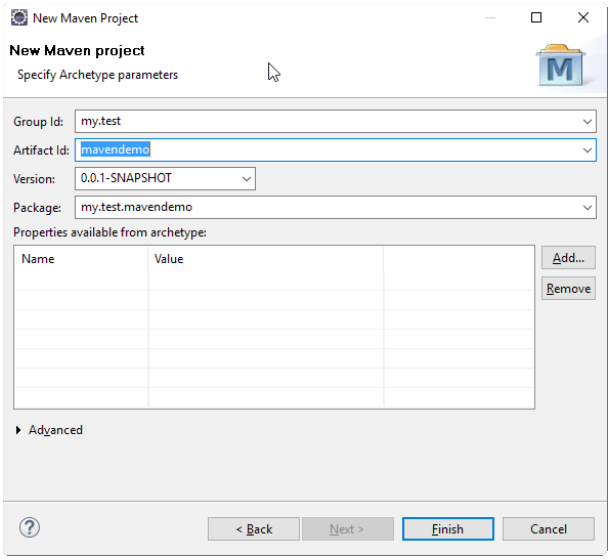
Store the project in the default workspace:



Select “maven-archetype-webapp” as archetype:

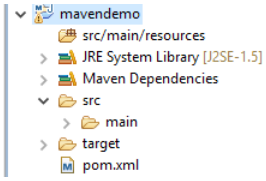


Define the Maven group id and the Maven artifact id. The artifact id will be the default name of your web-application and the name of the corresponding CaptainCasa project. So if you define “mavendemo” here, then this is the name of the project within the CaptainCasa toolset, and this is the default name of your web-application within your Tomcat (“localhost:50000/mavendemo”).



Setting up Maven

The project contains some pom.xml:



Update the POM to be:

```
<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/maven-
v4_0_0.xsd">
        <modelVersion>4.0.0</modelVersion>
        <groupId>my.test</groupId>
        <artifactId>mavendemo</artifactId>
        <packaging>war</packaging>
        <version>0.0.1-SNAPSHOT</version>
        <name>mavendemo Maven Webapp</name>
        <url>http://maven.apache.org</url>
        <repositories>
                <repository>
                        <id>org.eclnt</id>
                        <url>http://www.captaincasademo.com/mavenrepository</url>
                </repository>
        </repositories>
        <dependencies>
                <dependency>
                        <groupId>junit</groupId>
                        <artifactId>junit</artifactId>
                        <version>3.8.1</version>
                        <scope>test</scope>
                </dependency>
                <dependency>
                        <groupId>javax.servlet</groupId>
                        <artifactId>javax.servlet-api</artifactId>
                        <version>3.1.0</version>
                        <scope>provided</scope>
                </dependency>
                <dependency>
                        <groupId>org.eclnt</groupId>
                        <artifactId>eclntjsfserver</artifactId>
                        <version>20161031</version>
                </dependency>
                <dependency>
                        <groupId>javax.faces</groupId>
                        <artifactId>jsf-api</artifactId>
                        <version>2.2.14</version>
            <scope>provided</scope>
                </dependency>
                <dependency>
                        <groupId>org.glassfish</groupId>
                        <artifactId>javax.faces</artifactId>
```

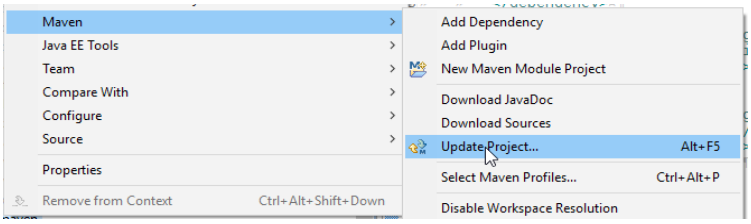
```
                        <version>2.2.14</version>
                <scope>provided</scope>
                    </dependency>
        </dependencies>
        <build>
                <finalName>mavendemo</finalName>
                <plugins>
                        <plugin>
                                <groupId>org.apache.maven.plugins</groupId>
                                <artifactId>maven-compiler-plugin</artifactId>
                                <version>3.1</version>
                                <configuration>
                                        <source>1.6</source>
                                        <target>1.6</target>
                                </configuration>
                        </plugin>
                        <!-- Unpack webapp addons into webcontentcc -->
                        <plugin>
                                <groupId>org.apache.maven.plugins</groupId>
                                <artifactId>maven-dependency-plugin</artifactId>
                                <executions>
                                        <execution>
                                                <id>unpack</id>
                                                <phase>package</phase>
                                                <goals>
                                                        <goal>unpack</goal>
                                                </goals>
                                                <configuration>
                                                        <artifactItems>
                                                                <artifactItem>

        <groupId>org.eclnt</groupId>
        <artifactId>eclntwebappaddonsRISC</artifactId>
        <version>20161031</version>
        <outputDirectory>webcontentcc</outputDirectory>
                                                                </artifactItem>
                                                        </artifactItems>
                                                </configuration>
                                        </execution>
                                </executions>
                        </plugin>
                        <!-- Add webcontentcc directory to the .war file -->
                        <plugin>
                                <groupId>org.apache.maven.plugins</groupId>
                                <artifactId>maven-war-plugin</artifactId>
                                <version>2.6</version>
                                <configuration>
                                        <webResources>
                                                <resource>
                                                        <directory>webcontentcc</directory>
                                                </resource>
                                        </webResources>
                                </configuration>
                        </plugin>
                </plugins>
        </build>
</project>
```

What is the basic content within the pom.xml?

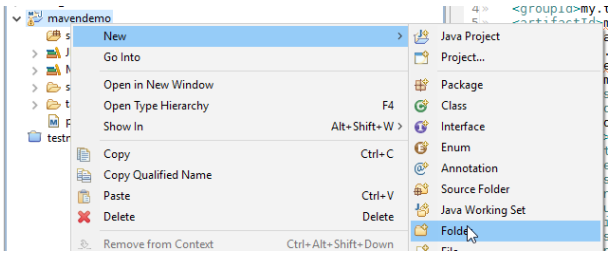
- You define the repository location, where CaptainCasa provides its artifacts: http://www.captaincasademo.com/mavenrepository
- You define some dependency to servlet-API, JSF-API and JSF-implementation. Please note: all of them are defined with scope “provided”. The servlet-API comes with the Tomcat you deploy to, the JSF-API and JSF-implementations come with CaptainCasa and are part of the “webappaddons” package.
- The plugin content is:
  - “maven-compiler-plugin”: You define that the project is sing Java version “1.6”.
  - “maven-dependency-plugin”: You define that the webappadons-part of CaptainCasa Enterprise Client is copied in the “webcontentcc” directory of your project.
  - “maven-war-plugin”: You define that the content of the project's “webcontentcc” directory is copied into the .war-structure.

After having updated the pom.xml you may already refresh the project by selecting:

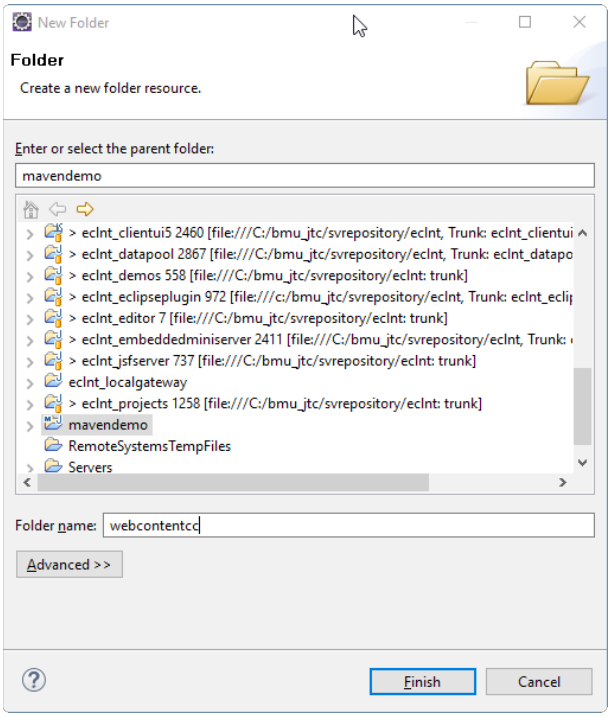


You will see some errors in the project – but may ignore them at this point of time.

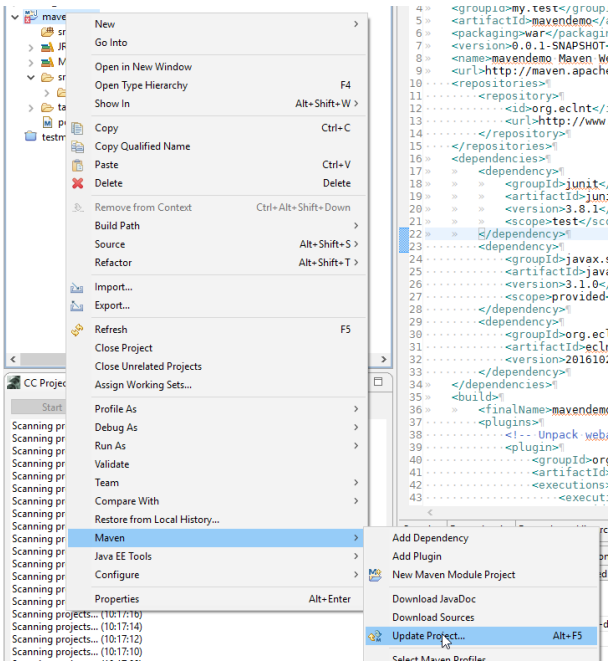
Now you create this “webcontentcc” folder, so open the corresponding menu...



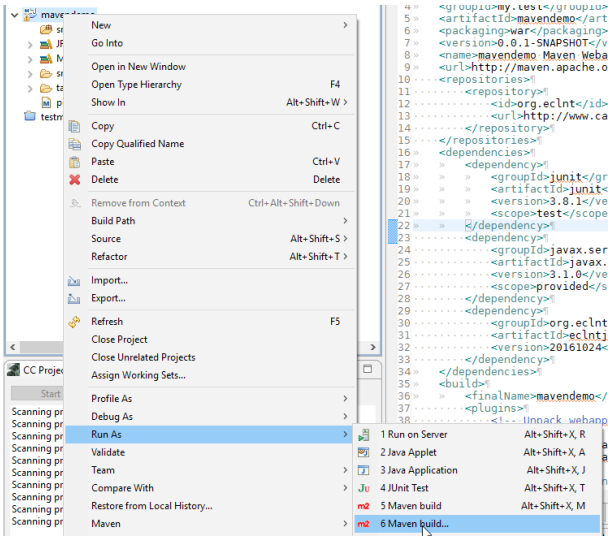
...and create the folder:



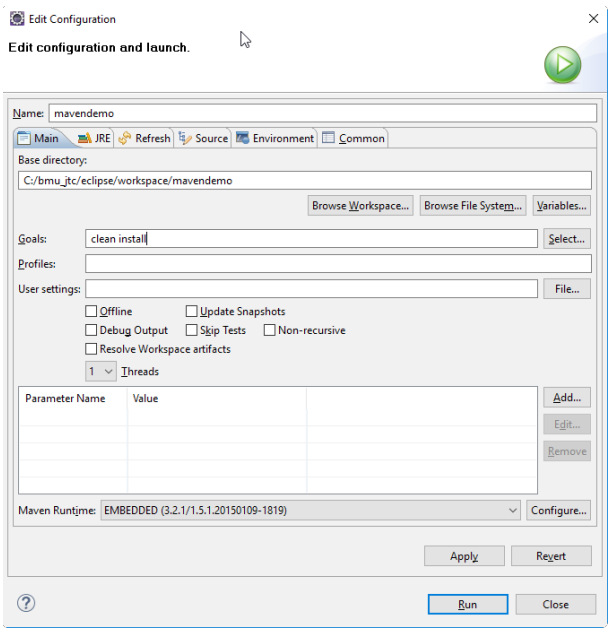
And now, finally you can first time execute the Maven build to check if things come together as planned. So call “Maven => Update project...” first:



And then start the Maven build by right clicking onto the project and selecting “Run => Maven build..”:

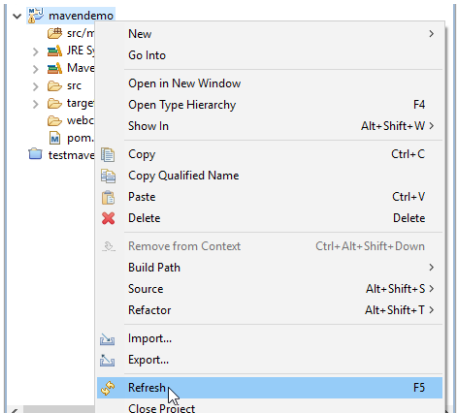


In the dialog that pops up define “clean install” as goal:

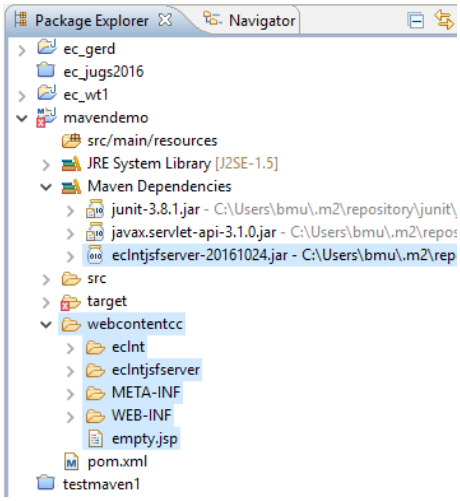


Now press “Run”. You may have to wait a bit of time when executing the build the first time because the CaptainCasa artifacts are loaded into your Maven repository in the background.

After the build you may want to see the results... So refresh your project first:



The project file structure now looks like:



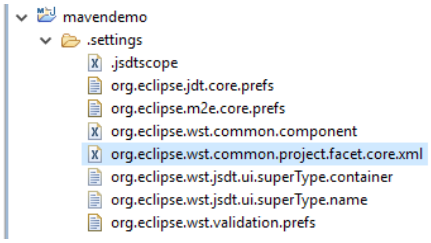
You see that the webcontentcc directory now was populated and contains the webappadons-content of CaptainCasa Enterprise Client.

Updating Eclipse Project Setup

Update Project Facets

The default project facets have to be updated. Reason: the default Maven project assumes to use the servlet standard 2.3 while the CaptainCasa web application required servlet standard 3.0.

Open the facet configuration directly on file level. It is part of the ".settings" of your project.



Please note: you need to switch into the "Navigator" view to see these files, you do not e.g. see them in the "Package Explorer".

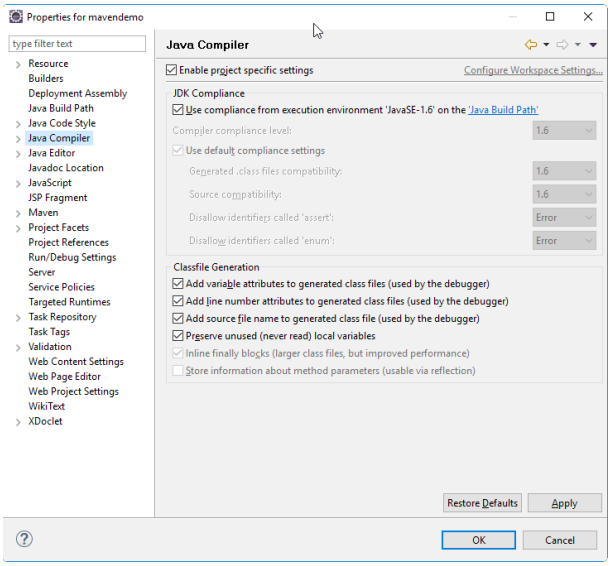
Update the versions in the following way:

```
<?xml version="1.0" encoding="UTF-8"?>
<faceted-project>
  <fixed facet="wst.jsdt.web"/>
  <installed facet="java" version="1.6"/>
  <installed facet="wst.jsdt.web" version="1.0"/>
  <installed facet="jst.web" version="3.0"/>
  <installed facet="jst.jsf" version="2.2"/>
</faceted-project>
```

Clean the project ("Project => Clean..." in the top menu) after applying the changes.

Check Project's Java Version

Check that your project is using Java 1.6 (or higher). This is part of the project properties:



Creating the CaptainCasa Project

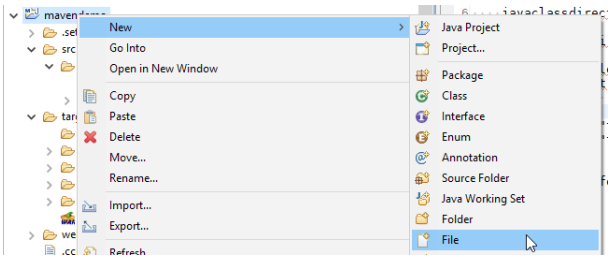
In order to use the CaptainCasa toolset for your project you need to create a CaptainCasa project definition, which tells the toolset where to find certain information it requires.

Examples:

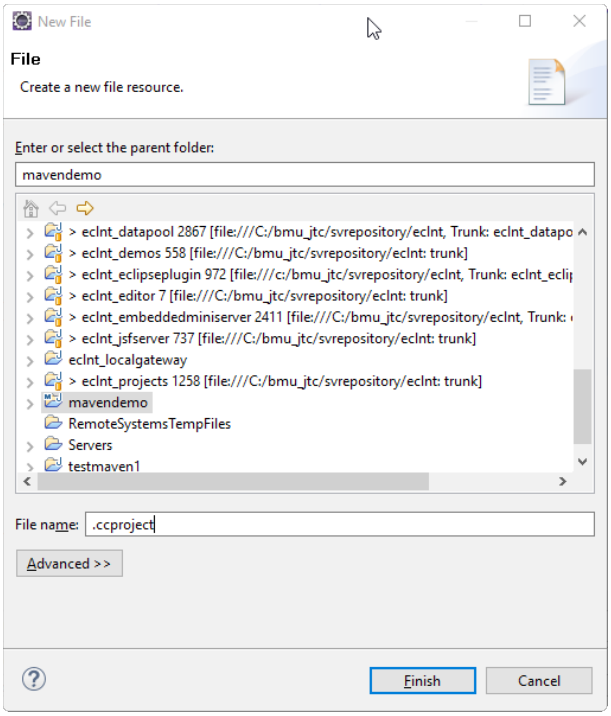
- Within the CaptainCasa toolset dialog definitions (.jsp) are created. These are stored in the project, so CaptainCasa needs to know where to store.
- Within the toolset you reload the project, this means the toolset needs to know where the project result (“.war”-structure) is located and where to copy it to.

Define .ccproject

This information is contained in the “.ccproject” file, which is part of your project's top directory. You now need to create this file, so in Eclipse select “New => File”:



And then define “.ccproject” as file name:



The file content is some simple XML definition:

```
<project

    managedbycctoolset="false"

    webcontentdirectory="${project}/src/main/webapp"
    javasourcedirectory="${project}/src/main/java"

    javaclassdirectory="${project}/target/classes"

    webappaddonsdirectory="${project}/webcontentcc"
```

```
webcontentdeploydirectory="C:/bmu_jtc/EnterpriseClient/server/tomcat/webapps/maven
demo"
    webcontextroot           ="mavendemo"
    webhostport              ="localhost:50000"

    copywebapp  ="true"
    reloadwebapp="true"
    >

    <deploycopyinfo fromdir="${project}/target/mavendemo" todir="$
{projectdeploy}"/>


</project>
```

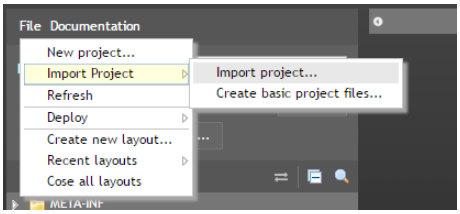
When defining your own file, apply the following changes:

- Make sure to replace “mavendemo” with the name of your artifact id.
- Replace that the parameter “webcontentdeploydirectory” is pointing into your CaptainCasa Tomcat, which is located in the “<CaptainCasaInstallDir>/server/tomcat” directory.

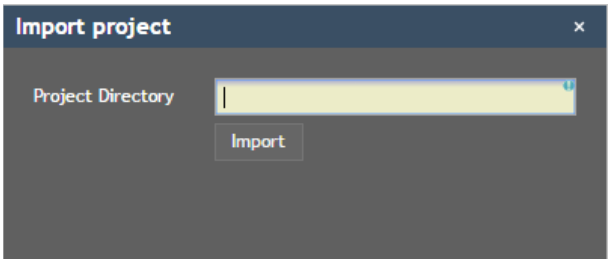
Import project into CaptainCasa Toolset

Please note: the import functions are part of CaptainCasa 20161031. Update your CaptainCasa environment if using an earlier version.

Start your CaptainCasa toolset, and Select “File => Import project...”.



Define the directory location of your project and press “Import”:



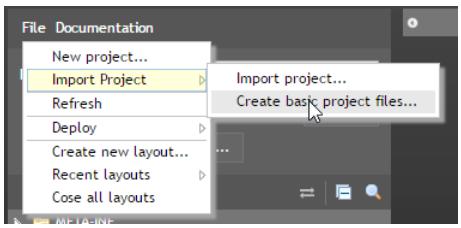
The project will be registered within the CaptainCasa toolset.

Create Project Basic Files

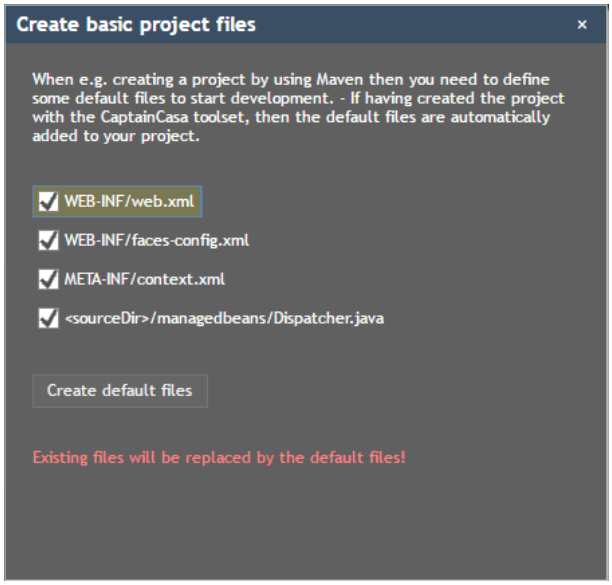
When using CaptainCasa Enterprise Client within an web application then there are certain items you need to do first:

- Update the WEB-INF/web.xml so that CaptainCasa filters and servlet are included.
- Add the WEB-INF/faces-config.xml to the project.
- Define a META-INF/context.file
- Start programming by typically using some Dispatcher-class.

In order to simplify the process of jump-starting the development there is a helper function:



Select “File => Import Project => Create basic project files”. A dialog will show up:



For a “fresh” project select all the items and press “Create default files”.

Refresh Eclipse Project

After registering the project in the CaptainCasa toolset and after adding the basic files, please refresh the project in the Eclipse environment.

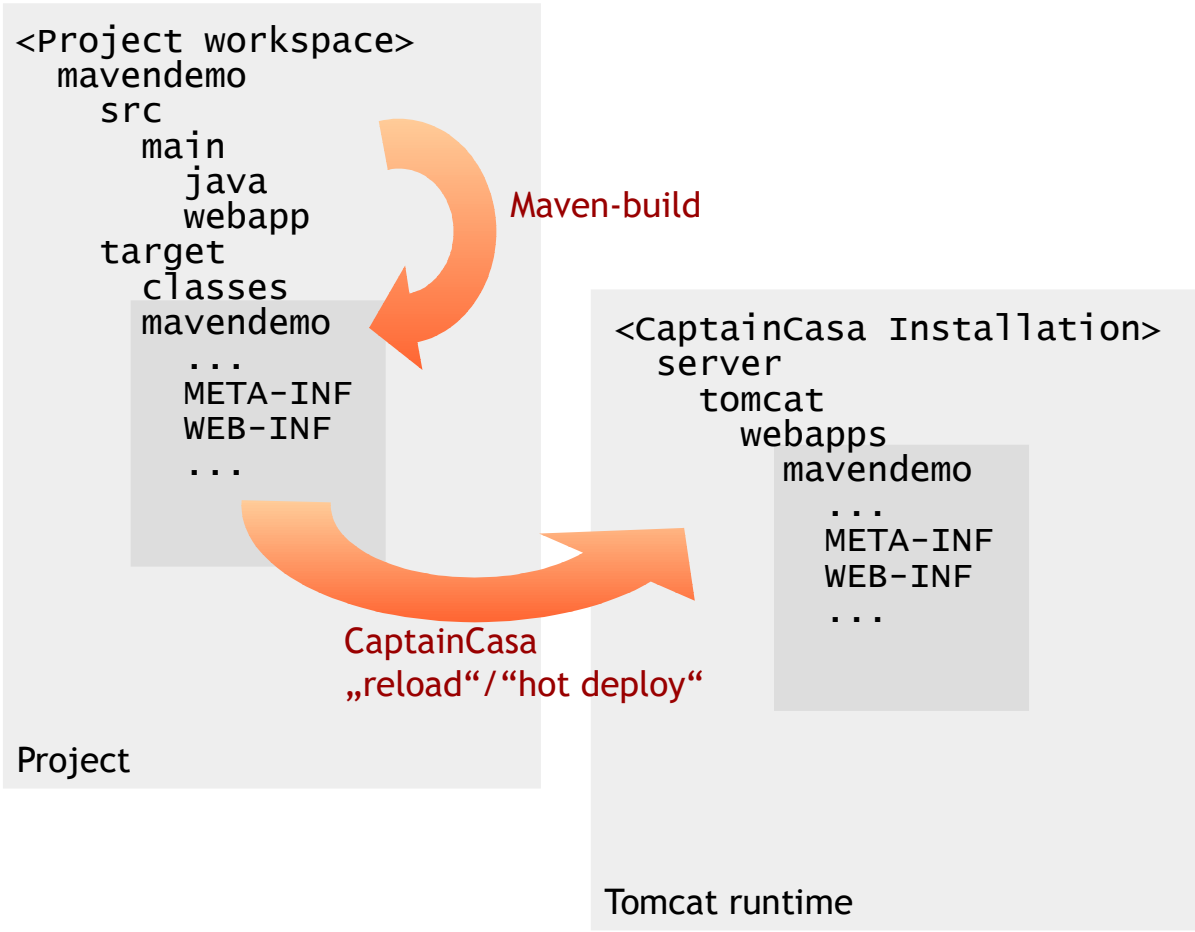
And now, finally!..., the project should not show any problems anymore!

….finished!

And, now you have reached the end of creating a Maven project! You now can develop/make your project with Eclipse/Maven and you can in parallel work with the CaptainCasa toolset editing dialogs (.jsp) and for deploying the application (“Reload”) to the Tomcat-runtime.

Develop - Make - Deploy - Test

The environment that was created during all this procedure now is:



- In the project the compilation and building of the web application is managed by Maven. The result is the target/mavendemo directory, in which the web application is produced.
- From the target/mavendemo directory the content of the web application is copied over into the Tomcat runtime – using the “reload” (or “hot deploy”) function with the CaptainCasa toolset.