Setting up a Maven Project

This documentation describes how to set up a Maven project for CaptainCasa.

Please use a CaptainCasa version higher than 20180102. There were quite some nice changes which were applied for CaptainCasa update 20180102 – a summary of changes is listed in the last chapter. These ones of you who already have used the CaptainCasa Maven artifacts should take a look into this chapter.

The setup of the project is described by using Eclipse as development environment. But the general steps are easily transfer-able to other development environments.

Overview

Repository Location

CaptainCasa provides the following Maven repository location:

http://www.captaincasademo.com/mavenrepository

Artifacts

In this repository there are the following artifacts:

```
/mavenrepository
    /org
        /eclnt
            /eclntjsfserver           => server side runtime library (.jar)
            /eclntwebappaddonsRISC     => web application addons, only RISC
            /eclntwebapparchetype      => arche type for creating maven projects
            /eclntpbc                  => Page Bean Components (addons)
```

There are two CaptainCasa artifacts that you depend from within your prohect: a compile time dependency to "eclntjsfserver", and an install time dependency to "webappaddonsRISC".

- eclntjsfserver: this is the jar file that is containing the CaptainCasa server side runtime library. It is the "eclntjsfserver.jar" that is added into WEB-INF/lib. The library is required at compile time because you directly interact with the classes of this library when developing dialogs.

- eclntwebappaddonsRISC: this is a collection of file that CaptainCasa adds to the web application: layout definitions of default dialogs (OK, Yes-No, ...), images, other resources – and last but not least: the whole JavaScript client processing. It's the non-Java part that is added to the web application content.

There are two additional artifacts:

- eclntwebapparchetype: this is the maven archetype for creating Maven projects in a simple way. It is not referenced within your project – it is only required as a template for creating your project.

- eclntpbc: this is an add-on library containing useful application components. It is not required for starting with CaptainCasa.

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 Maven Project

For Maven-professionals – the ultra quick guide

You need to create a Maven project with archetype "eclntwebapparchetype". The archetype is an extension of the normal webapp-archetype. It is contained in the catalog at http://www.captaincasademo.com/mavenrepository/archetypecatalog.xml. - That's it!
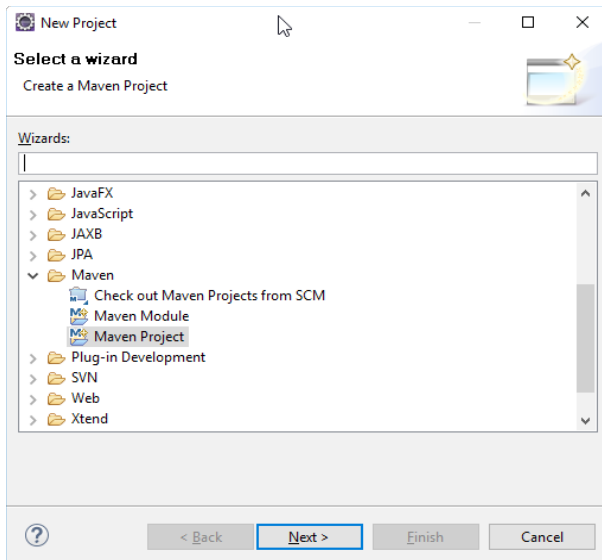
For all non-Maven-professionals: the following chapter describes the creation of the project step by step using Eclipse as development environment.
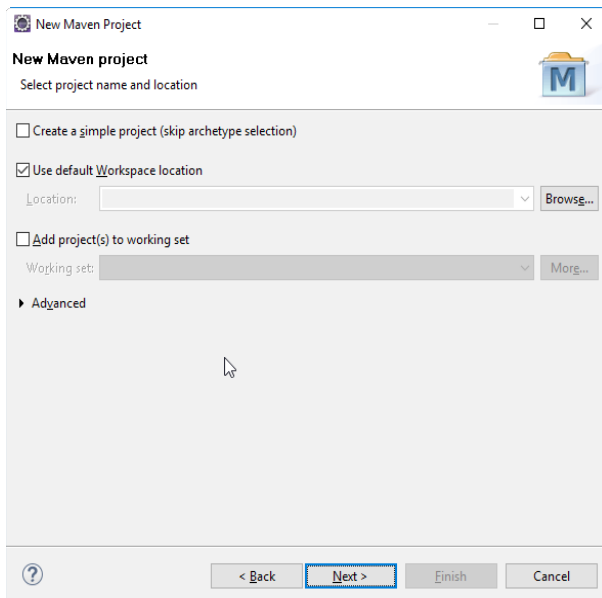
Create the Maven project in Eclipse

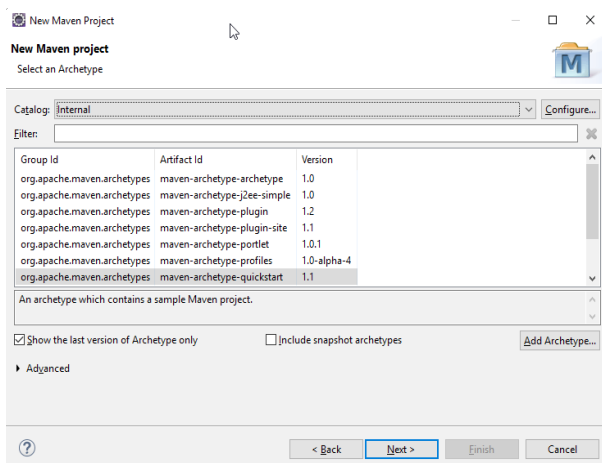The project is created in Eclipse. Select File => New => Project...:
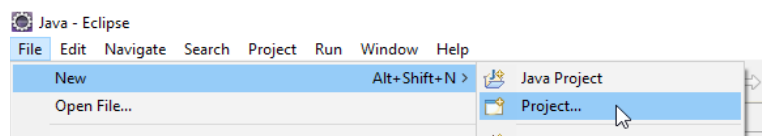
Select "Maven Project":

Store the project in the default workspace:

Now select the CaptainCasa project archetype: the dialog by default lists a couple of default archetypes that are included in some local catalog of archetypes:


The CaptainCasa archetypes are available via URL:

http://www.captaincasademo.com/mavenrepository/archetypecatalog.xml

So open the catalog configuration via the "Configure"-button, select "Add Remote Catalog..." and define the URL and a text description:
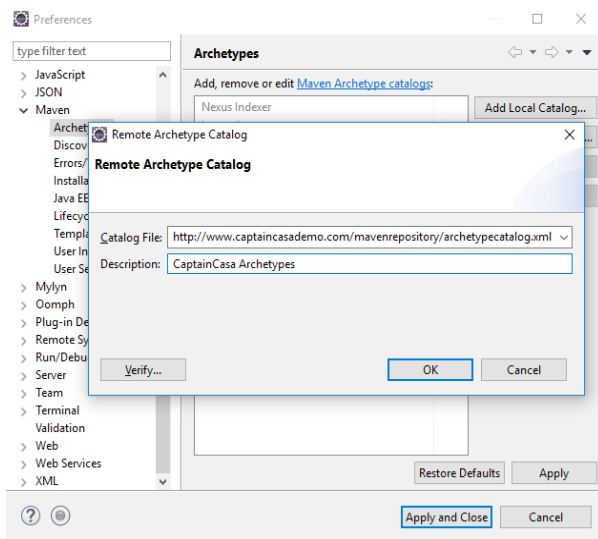
After applying the catalog, closing the dialog and sometimes waiting some seconds you can select the CaptainCasa project archetype:
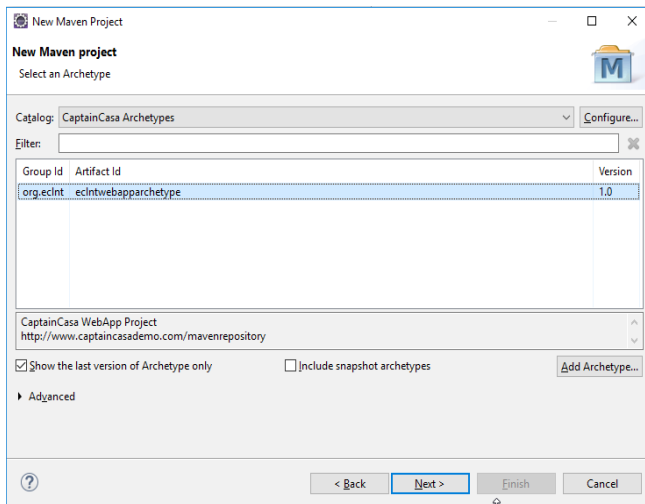
Press "Next!" and define the Maven group id and the Maven artifact id. The artifact id will be the default name of your web-application and the name of the corresponding CaptainCasa project. So if you define "mavendemo" here, then this is the name of the project within the CaptainCasa toolset, and this is the default name of your web-application within your Tomcat ("localhost:50000/mavendemo").
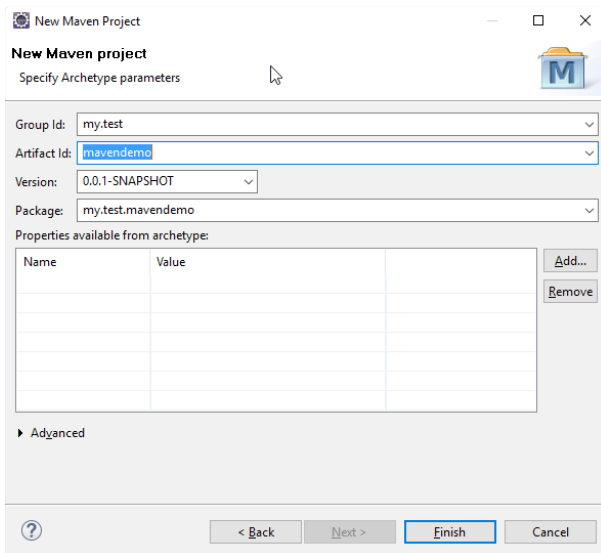
After pressing "Finish" the project is created. The nature of the project is a normal web-application project, due to the selection of the CaptainCasa archetype the project already contains the right configuration files in order to quickly start with first development steps.

Take a look into the pom.xml-file

The pom.xml-file

The project contains a pom.xml in its root directory:
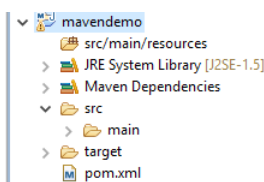
The content looks like:

```
<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>mavendemo</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>
        <!-- CaptainCasa repository -->
        <repository>
            <id>org.eclnt</id>
            <url>http://www.captaincasademo.com/mavenrepository</url>
        </repository>
        <!-- Repository for dependent artifacts -->
        <repository>
            <id>central.maven.org</id>
            <url>http://central.maven.org/maven2/</url>
        </repository>
        <repository>
            <id>mvnrepository.com</id>
            <url>https://mvnrepository.com/repos/central</url>
        </repository>
        <repository>
            <id>org.outerj</id>
            <url>https://maven.atlassian.com/3rdparty/</url>
        </repository>
    </repositories>
    <dependencies>
        <dependency>
```

```xml
        <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>20171229_INT6</version>
    </dependency>
    <!-- The following libraries are defined with scope "provided" within
         org.eclnt.eclntjsfserver - so this is the place to really provide them...! -->
    <dependency> <!-- JSF -->
        <groupId>com.sun.faces</groupId>
        <artifactId>jsf-api</artifactId>
        <version>2.2.14</version>
    </dependency>
    <dependency> <!-- JSF -->
        <groupId>com.sun.faces</groupId>
        <artifactId>jsf-impl</artifactId>
        <version>2.2.14</version>
    </dependency>
    <dependency> <!-- JSF dependency -->
        <groupId>javax.servlet.jsp.jstl</groupId>
        <artifactId>jstl-api</artifactId>
        <version>1.2</version>
    </dependency>
    <dependency> <!-- PDF creation in grid processing -->
        <groupId>com.lowagie</groupId>
        <artifactId>itext</artifactId>
        <version>2.1.7</version>
    </dependency>
    <dependency> <!-- HTML sanitizing -->
        <groupId>org.jsoup</groupId>
        <artifactId>jsoup</artifactId>
        <version>1.10.3</version>
    </dependency>
    <dependency> <!-- HTFILEUPLOAD component -->
        <groupId>commons-fileupload</groupId>
        <artifactId>commons-fileupload</artifactId>
        <version>1.3.3</version>
    </dependency>
    <dependency> <!-- log4j bridge -->
        <groupId>org.apache.logging.log4j</groupId>
        <artifactId>log4j-core</artifactId>
        <version>2.8.2</version>
    </dependency>
    <dependency>
        <groupId>log4j</groupId>
        <artifactId>log4j</artifactId>
        <version>1.2.17</version>
    </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. Due to the "excludes"
             statement no library jar files are copied (which are normally part of the
             webappaddons package. -->
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-dependency-plugin</artifactId>
            <executions>
                <execution>
                    <id>unpack</id>
                    <phase>prepare-package</phase>
                    <goals>
                        <goal>unpack</goal>
                    </goals>
                    <configuration>
                        <artifactItems>
                            <artifactItem>
                                <groupId>org.eclnt</groupId>
                                <artifactId>eclntwebappaddonsRISC</artifactId>
                                <version>20171229_INT6</version>
                                <outputDirectory>webcontentcc</outputDirectory>
                                <!-- Do not copy the libraries! -->
                                <excludes>WEB-INF/lib/**.*</excludes>
                            </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>2.2</version>
             <configuration>
                 <webResources>
                     <resource>
                         <directory>webcontentcc</directory>
                     </resource>
                 </webResources>
             </configuration>
         </plugin>
     </plugins>
   </build>
</project>
```

What is the basic content within the pom.xml?

• The pom contains several maven repositories including the one of CaptainCasa:
  http://www.captaincasademo.com/mavenrepository

• There are a couple of dependencies:

  • eclntjsfserver – this is is the CaptainCasa server runtime library

  • servlet-api, jsf-api, jsf-impl, jstl-api - these are the prerequistes of a JSF based web application

  • ...other libraries used for special tasks – please check the corresponding comments within the pom.xml

• The plugin content is:

  • "maven-compiler-plugin": the project is using Java version "1.6".

  • "maven-dependency-plugin": the webappadonsRISC-part of CaptainCasa Enterprise Client is copied in the "webcontentcc" directory of your project. When copying the WEB-INF/lib/-files are excluded, because the libaries are added to the project via Maven.

    This is a quite important issue – especially for "old CaptainCasa users", so for these ones, this issue is explicitly repeated: the libraries are picked via Maven – and are not copied from some CaptainCasa default project content!

  • "maven-war-plugin": the content of the project's "webcontentcc" directory is copied into the .war-structure, so all the files added to the webcontent are copied as part of the packaging.

Update the project

Update the project by calling "Maven → Update Project..." after right clicking onto the project. And because this is a step that you have to do quite frequently...: keep in mind the short key to do this, it's "alt-F5"...
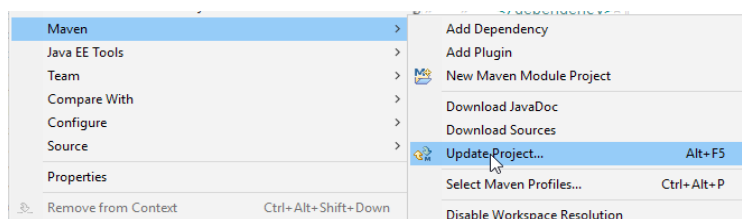
Now you can first time execute the Maven build to check if things come together as planned. Start the Maven build by right clicking onto the project and selecting "Run => Maven build...":
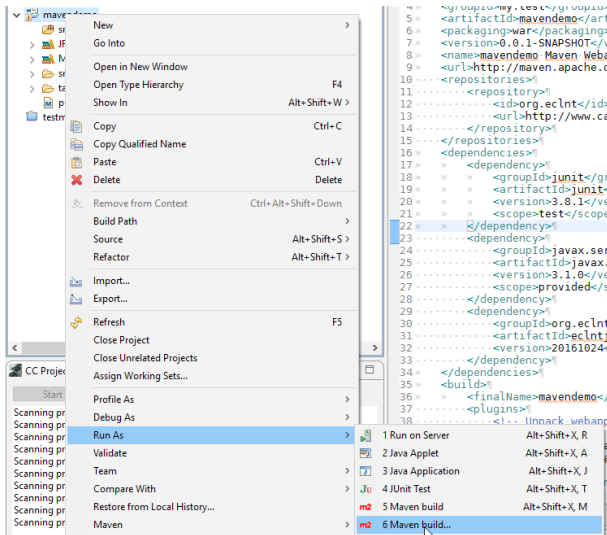
In the dialog that pops up define "clean install" as goal:
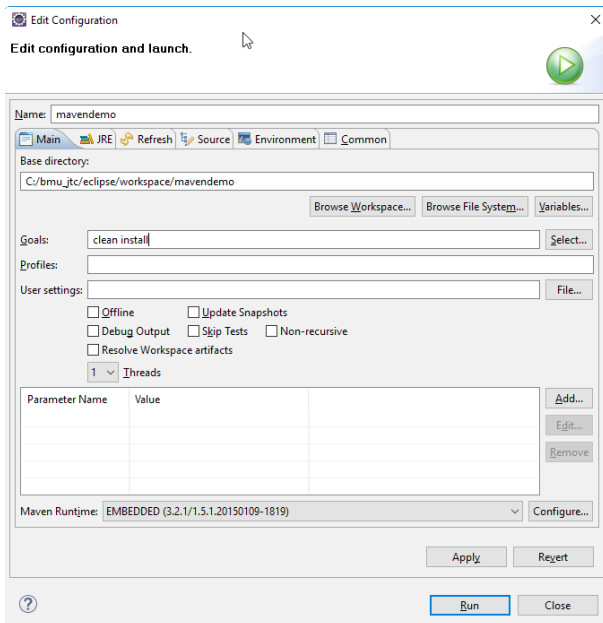
Now press "Run". You may have to wait a bit of time when executing the build the first time because the CaptainCasa artifacts are loaded into your Maven repository in the background. You may observe the execution of the build within the Console view of Eclipse.

After the build you may want to see the results... So refresh your project first:
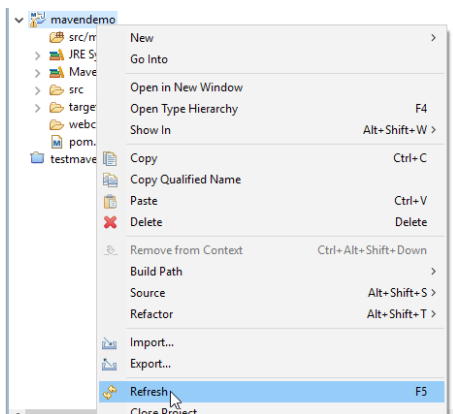
The project file structure now looks like:
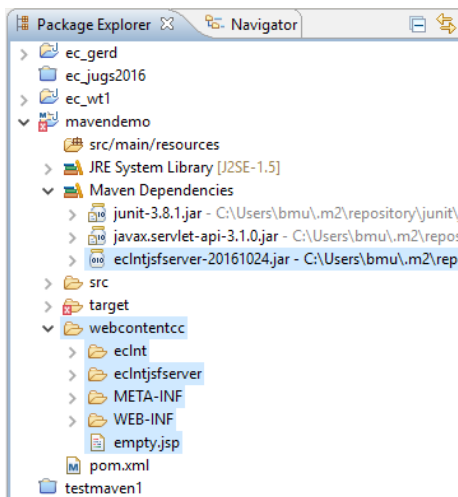
You see that the webcontentcc directory now was populated and contains the webappadonsRISC-content of CaptainCasa Enterprise Client.

That's the Maven side

The project is up and running – and it delivers a .war file as result within its target directory. You now could manually edit XML-layout definitions (.jsp files) in the webapp part of the project, add images, add managed bean implementations, etc. - But of course we recommend to use the CaptainCasa toolset in order to more easily do this, and in order to deploy this to some Tomcat runtime in an efficient way. This is what the next chapter is about.

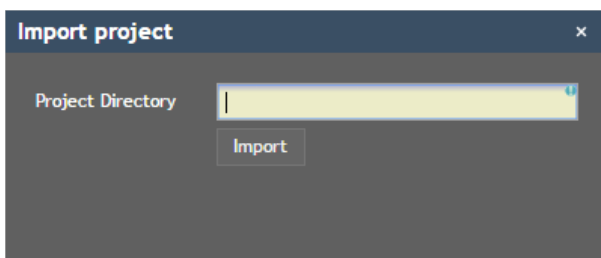# Creating the CaptainCasa Project

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.

Import project into CaptainCasa Toolset

Start your CaptainCasa toolset, and Select "File => Import project...".
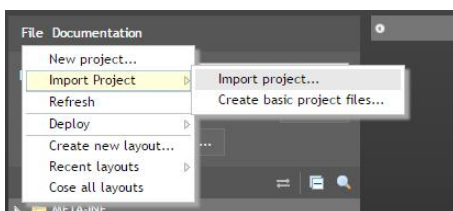
Define the directory location of your project's root directory and press "Import":

The project will be registered within the CaptainCasa toolset.

...finished!

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

Develop & Make

The environment that was created during all this procedure now is:
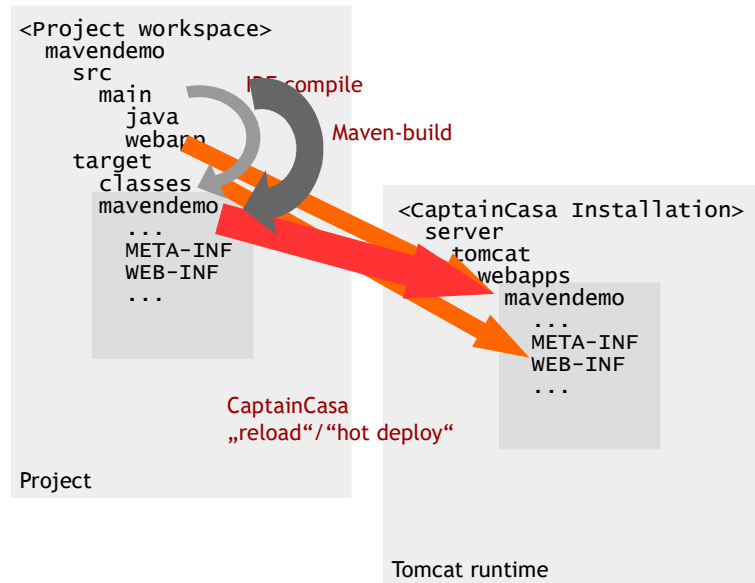
The rounded arrows are showing the typical steps during development:

- You compile your Java programs from "src/main/java" to "target/classes"

- You from time to time make full Maven builds into the "target/<projectName>" directory

The straight arrows are showing what happens when a "Reload" is executed within the CaptainCasa toolset:

- The complete "target/<projectName>" directory is copied into the corresponding "tomcat/webapps/<projectName>" directory.

- The content of the "src/main/webapps" folder is copied into the corresponding "tomcat/webapps/<projectName>" directory.

- The content of the "target/classes" folder is copied into the corresponding "tomcat/webapps/<projectName>/WEB-INF/classes" or into the "tomcat/webapps/<projectName>/eclnthotdeploy/classes" folder – according to hot deployment rules.

After the copying the "Reload" then restarts the "<projectName>"-web application by using the Tomcat manager API.

You may wonder: why are there three copying steps for loading the application into the Tomcat – and not only one, copying the Maven-target into the Tomcat-webapps?

The reason is that by this procedure the number of Maven builds that you need to do during development time is drastically reduced. A Maven build typically takes quite some

time, and it is just annoying to execute this build after every simple development activity.

Due to the copying above the Tomcat is constantly updated not only from the "target/<projectName>" directory, but also from the directories in which you typically work in when doing UI developments. You only have to run a full Maven build, if really e.g. changing dependencies, so that new .jar files are to be copied into the Tomcat environment.
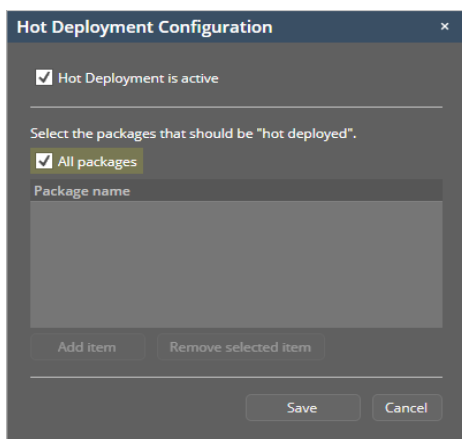
Use of hot deployment

Just as with usual CaptainCasa projects you can use hot deployment to significantly reduce deployment times. Hot deployment is activated from the CaptainCasa toolset:

Open "Configuration → Hot Deployment"...
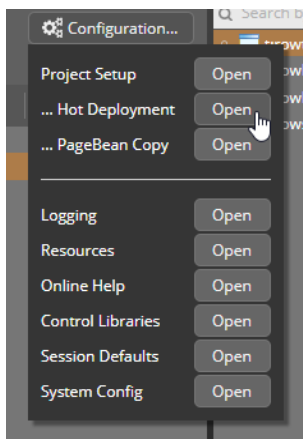
...and defined the packages to be hot deployed (or "all packages") in the follow-on dialog:

More details on Hot Deployment are contained in the Developers' Guide.

Updating the CaptainCasa Version

After having created your project with e.g. CaptainCasa version "20180102" you may later on update to a newer CaptainCasa version (e.g. 20180205"). For updating your project to a new CaptainCasa version you just need to execute the following steps:

• Update the version number within the pom.xml. - Pay attention it occurs two times: first time in the dependency section (artifact "eclntjsfserver"), second time in the plugin section (artifact "eclntwebappaddonsRISC").

• Execute a "Maven refresh" within your project (at least if you are using Eclipse).

- Execute a Maven-"clean install" with your project.

- Execute a "Clean & Reload" of the project within the CaptainCasa toolset to deploy the updates to the Tomcat runtime.

Changes with update 20180102

(This chapter is only relevant if having used Maven projects before update 20180102.)

Update 20180102 contained quite some nice changes in the area of "CaptainCasa & Maven":

Own project archetype "eclntwebapparchetyp"

When creating a new project then you directly start with some CaptainCasa archetype which installs all the files that are required within a CaptainCasa project.

Before, we explained how to integrate CaptainCasa into a project which was created with the just default archetype for creating web application projects. As consequence there were much more steps to follow.

Library Management

All libraries are now picked from Maven-dependencies. The libraries "around CaptainCasa" are no longer copied from the CaptainCasa eclntwebappaddonsRISC-artifact.

The artifact "eclntjsfserver" (which is the central runtime library of CaptainCasa) is properly set up now, containing all its dependencies – but expecting them to be provided. In the pom.xml definition of a project these libraries then are explicitly provided.

Plugin Execution

The unpacking of the artifactg "eclntwebappaddonsRISC" is now done in the phase "prepare-package" - explicitly before the WAR file is packaged. Before the execution was in the "package" phase – after the WAR file creation. This was clearly some bug – and you had to initially execute the "clean install" twice before.

Smarter Integration into CaptainCasa Toolset

The import of the Maven-project into the CaptainCasa toolset is now much easier – there is no manual interaction required anymore (like setting the deployment directory).