Tycho tutorial

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME
	January 2013		CP

Contents

Initial version

1 Tycho build 1: Building plug-ins

This tutorial is taken from: Code and me blog

Tycho is a great build tool for all your RCP build needs. It is a plug-in to maven and helps you to set up a reproducible build process which can be run interactively from your IDE or in headless mode (eg. on a build server).

While there are already some tutorials out there (Mattias Holmqvist, Lars Vogel) how to integrate tycho, I could not find one that focuses on UI integration of maven. This article was heavily inspired by a great talk during EclipseCon by Jan Sievers and Tobias Oberlies.

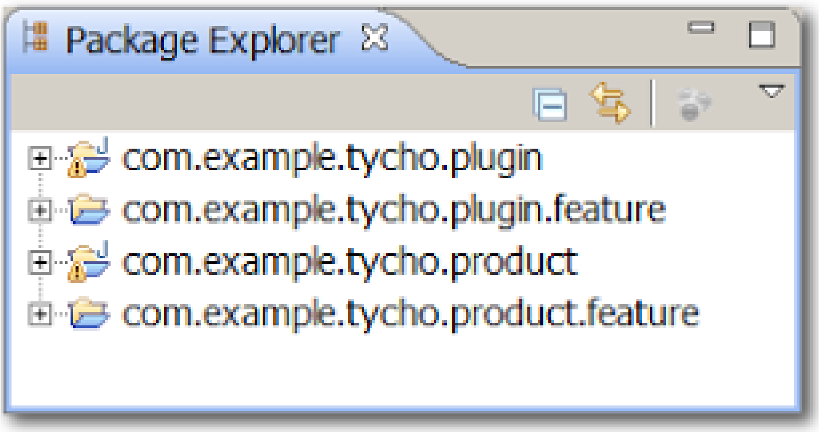
I will set this up as a series of posts. We will start by building a single plug-in and end up with a whole application built with tycho.

During the tutorial I will use a plain installation of Eclipse for RCP and RAP Developers, Juno SR1. No external programs are needed (so you don't need to install maven separately).

Source code for this tutorial is available on googlecode as a single zip archive, as a Team Project Set or you can checkout the SVN projects directly.

1.1 Preparations

Before we start using tycho I created a short sample project consisting of a small eclipse-like product along with an additional feature. The latter contains one plug-in that provides a custom toolbar entry and an (almost empty) view.



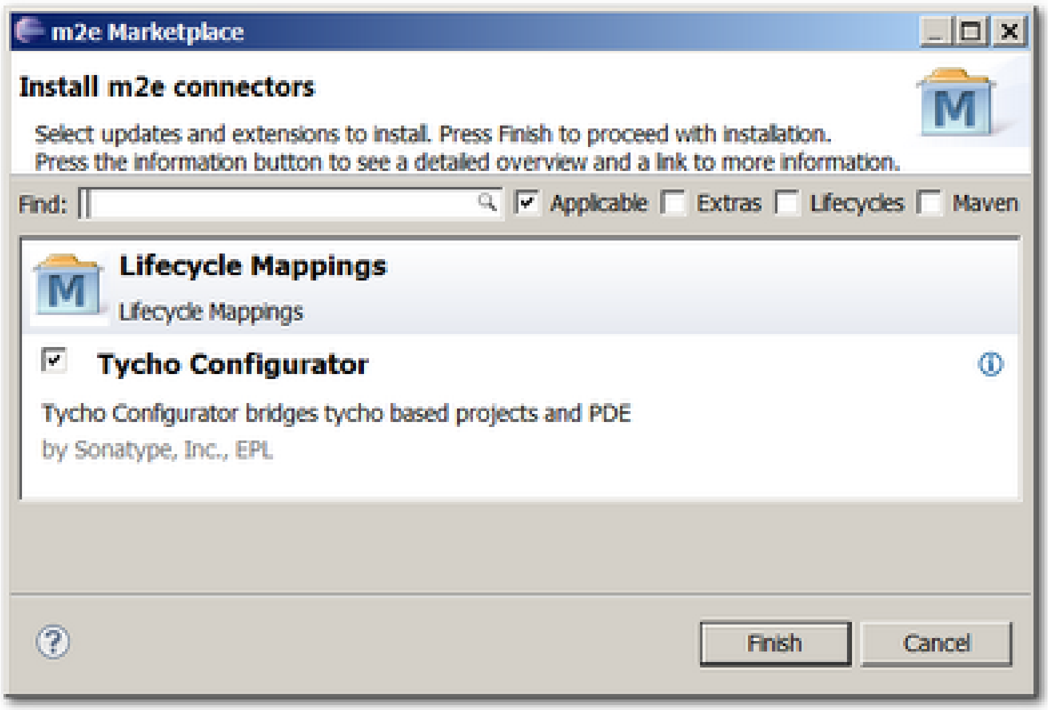
com.example.tycho.plugin
com.example.tycho.plugin.feature
com.example.tycho.product
com.example.tycho.product.feature

You can grab the initial sources from googlecode as a single zip file.

1.1.1 Step 1: Install Tycho connector

Eclipse for RCP and RAP Developers already comes with m2e, the maven integration tools of eclipse. Maven itself needs to be extended with tycho to allow to build plug-ins and other RCP like things.

Go to Preferences/Maven/Discovery and click on Open Catalog. Find and select the Tycho Configurator.


m2e Marketplace

Install m2e connectors

Select updates and extensions to install. Press Finish to proceed with installation.
Press the information button to see a detailed overview and a link to more information.

Find: [✓] Applicable [ ] Extras [ ] Lifecycles [ ] Maven

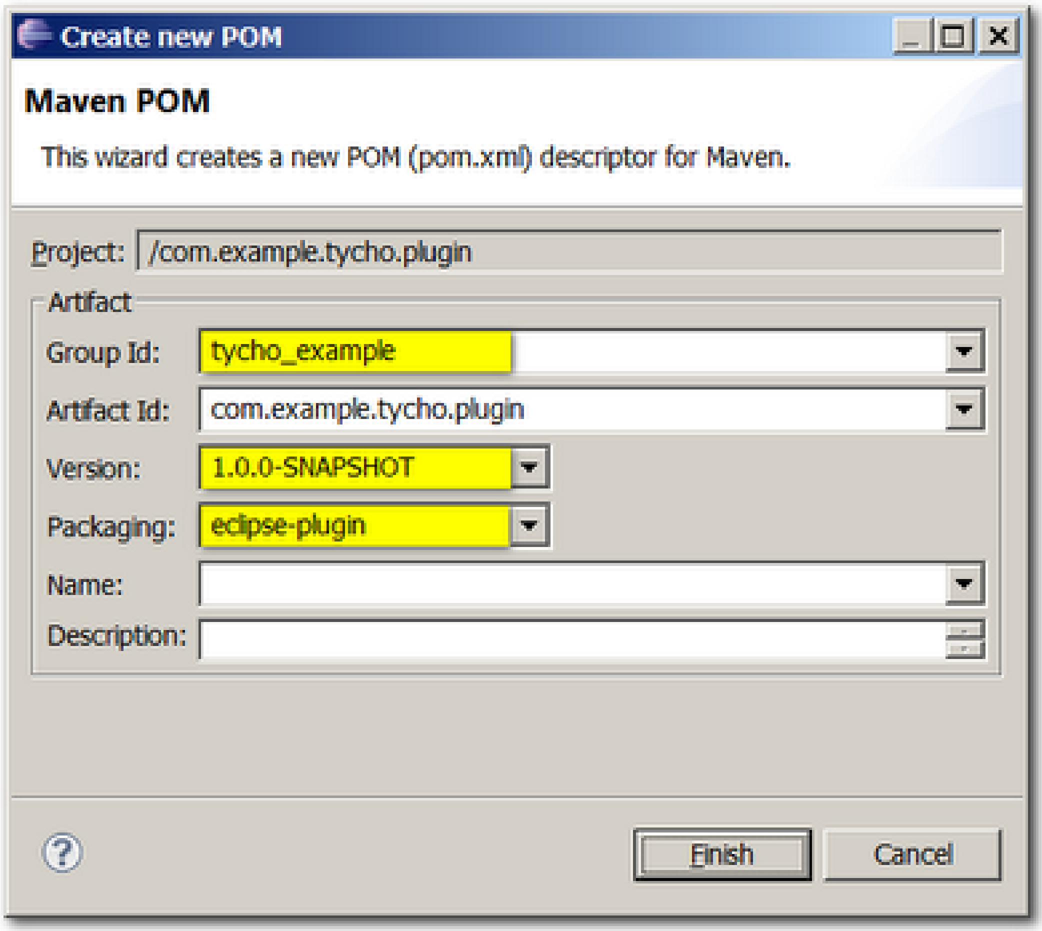
Lifecycle Mappings
Lifecycle Mappings

[✓] Tycho Configurator
Tycho Configurator bridges tycho based projects and PDE
by Sonatype, Inc., EPL

Install it and restart eclipse.

1.1.2 Step 2: Build a simple plug-in project

We will start by building com.example.tycho.plugin. Right click on the project and select Configure/Convert to Maven Project.



Maven POM

This wizard creates a new POM (pom.xml) descriptor for Maven.

Project: /com.example.tycho.plugin

Artifact

Group Id: tycho_example

Artifact Id: com.example.tycho.plugin

Version: 1.0.0-SNAPSHOT

Packaging: eclipse-plugin

Name:

Description:

The wizard asks for a Group Id. Use a name that represents the component you want to build. Think of a component as a thing to assemble like my_product or JDT, PDE, . . . you get the point. All projects that belong together will get the same Group Id.

Leave the Artifact Id to the name of your project. Actually it should match the Bundle-SymbolicName found in MANIFEST.MF.

Version should be set accordingly to Bundle-Version from the manifest. Later we will see how to keep those consistent. SNAPSHOT in maven is similar to qualifier in you plug-in version.

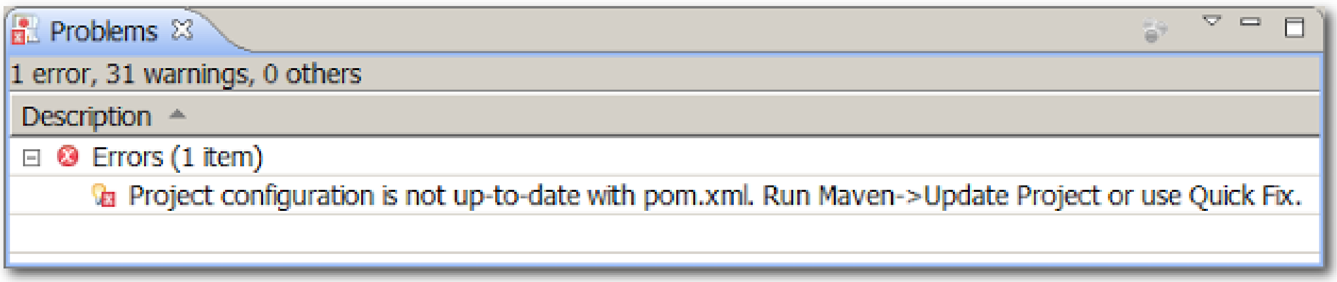
Finally Packaging tells maven what type of build instructions to use. Set it to eclipse-plugin.

Maven creates a pom.xml for you which we immediately replace with this one:

```
<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>tycho_example</groupId>
 <artifactId>com.example.tycho.plugin</artifactId>
 <version>1.0.0-SNAPSHOT</version>
 <packaging>eclipse-plugin</packaging>

 <properties>
  <tycho-version>0.16.0</tycho-version>
 </properties>

 <repositories>
  <!-- add Juno repository to resolve dependencies -->
  <repository>
   <id>Juno</id>
   <layout>p2</layout>
   <url>http://download.eclipse.org/releases/juno/</url>
  </repository>
 </repositories>

 <build>
  <plugins>
   <plugin>
    <!-- enable tycho build extension -->
    <groupId>org.eclipse.tycho</groupId>
    <artifactId>tycho-maven-plugin</artifactId>
    <version>${tycho-version}</version>
    <extensions>true</extensions>
   </plugin>
  </plugins>
 </build>
</project>
```

Lines 9-11 add a property for the tycho version to be used. For future tycho versions you will need to upgrade this. Lines 13-20 add the Juno p2 repository for resolving dependencies during build time. The build section (lines 22-32) tells maven to use the tycho plug-in for the build process.

Now you will see one error in your Problems View:



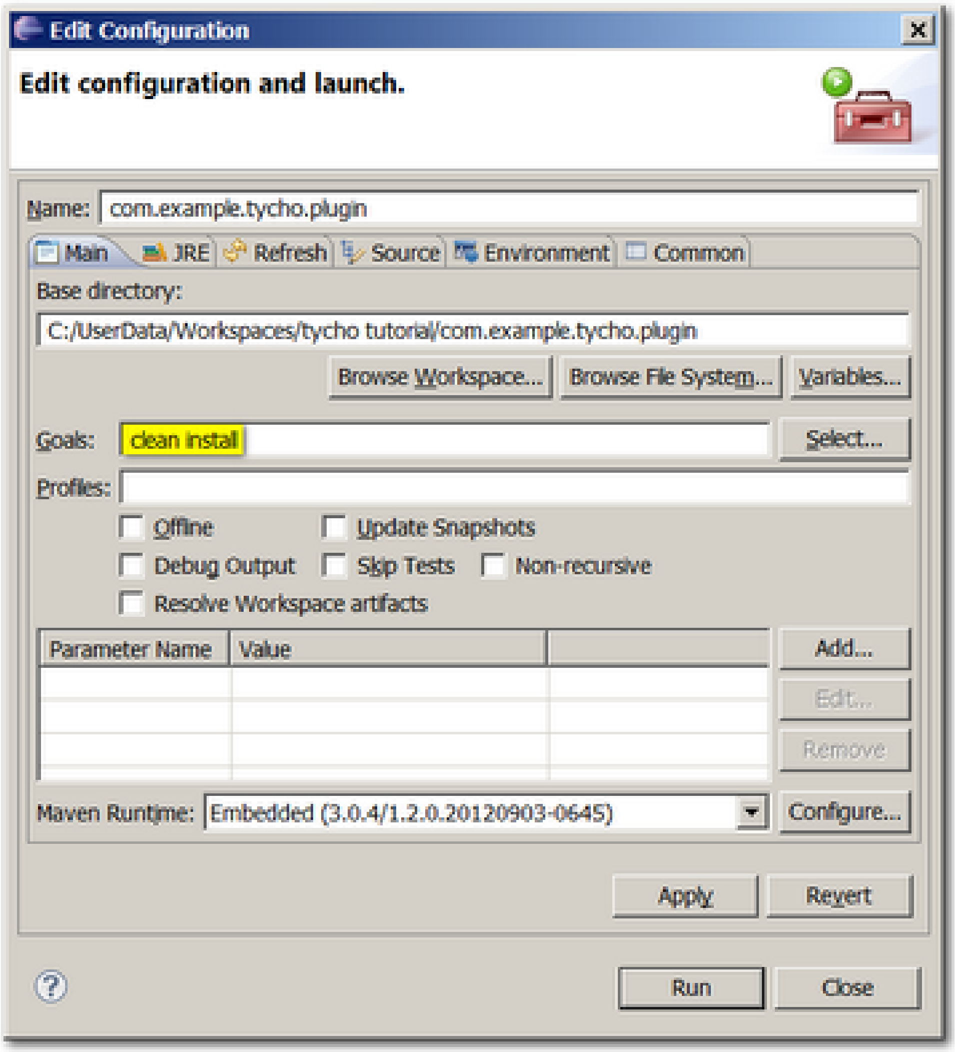
1 error, 31 warnings, 0 others

Errors (1 item)

Project configuration is not up-to-date with pom.xml. Run Maven->Update Project or use Quick Fix.

We will face this error each time we convert a project to maven. To get rid of it select it in the Problems View and use the Quick Fix (from the context menu or by pressing Ctrl-1).

Time to build our bundle: Right click on the project and select Run As/Maven build... Under Goals enter clean install. Goals are related to the maven lifecycle. See it as something similar to make targets if you are used to that. Basically we tell maven to delete previous build artifacts, to build our plug-in and to install build results.



Name: com.example.tycho.plugin

Base directory: C:/UserData/Workspaces/tycho tutorial/com.example.tycho.plugin

Goals: clean install

Profiles:

[ ] Offline [ ] Update Snapshots

[ ] Debug Output [ ] Skip Tests [ ] Non-recursive

[ ] Resolve Workspace artifacts

Maven Runtime: Embedded (3.0.4/1.2.0.20120903-0645)

The first run will take some time as dependencies need to be downloaded from the Juno p2 site. At the end you should see something like this in your Console View:

```
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 18.128s
[INFO] Finished at: Tue Nov 27 15:08:40 CET 2012
[INFO] Final Memory: 58M/139M
[INFO] ------------------------------------------------------------------------
```

Now check your project to find a new folder called target. It contains your build artifacts along with intermediate build files. Tycho will not refresh your workspace, so you have to do that manually to see the content of your target folder. If you want eclipse to do this automatically then open the run target you created before, switch to the Refresh tab and refresh The project containing the selected resource.

Additionally tycho changed your .classpath file to write output to target/classes instead of the default bin folder.

Congratulations, you’ve just built your first plug-in with tycho.

2 Tycho build 2: Global maven settings

When maven is executed, it reads a global settings file which you should adapt to your needs. You can find its location in Preferences/Maven/User Settings. If you update the file you should either restart eclipse or click Update Settings on the preferences page.

2.1 Setting a network proxy

Maven needs network access. At least during the first runs it typically needs to download some catalogs. If you are located behind a proxy you might expect that maven honors your eclipse proxy settings. But as maven is an external tool, it does not honor your eclipse proxy settings. You have to enter them manually in the configuration file:

```
<settings>
 .
 .
 <proxies>
  <proxy>
   <active>true</active>
   <protocol>http</protocol>
   <host>proxy.somewhere.com</host>
   <port>8080</port>
   <username>proxyuser</username>
   <password>somepassword</password>
   <nonProxyHosts>www.google.com|*.somewhere.com</nonProxyHosts>
  </proxy>
 </proxies>
 .
 .
</settings>
```

Read the documentation for more details.

Setting repository mirrors

In the first tutorial we used the Juno download site to resolve our build dependencies. Maven allows to define mirrors for repositories which we can put in the global settings file:

```
<settings>
 .
 .
 <mirrors>
  <mirror>
   <id>Juno_local</id>
   <mirrorOf>Juno</mirrorOf>
   <name>Local mirror of Juno repository</name>
   <url>file://C/<some_folder/</url>
   <layout>p2</layout>
   <mirrorOfLayouts>p2</mirrorOfLayouts>
  </mirror>
 </mirrors>
 .
 .
</settings>
```

The important parameter is mirrorOf. It contains the id of the original repository as defined in our pom file. As a consequence maven will favor our local mirror to resolve dependencies. If it is not available maven will go online and fetch from the Juno download site.

When you share your poms with your team everybody will be able to build the project. Still you can use local mirrors for faster (or network independent) builds.

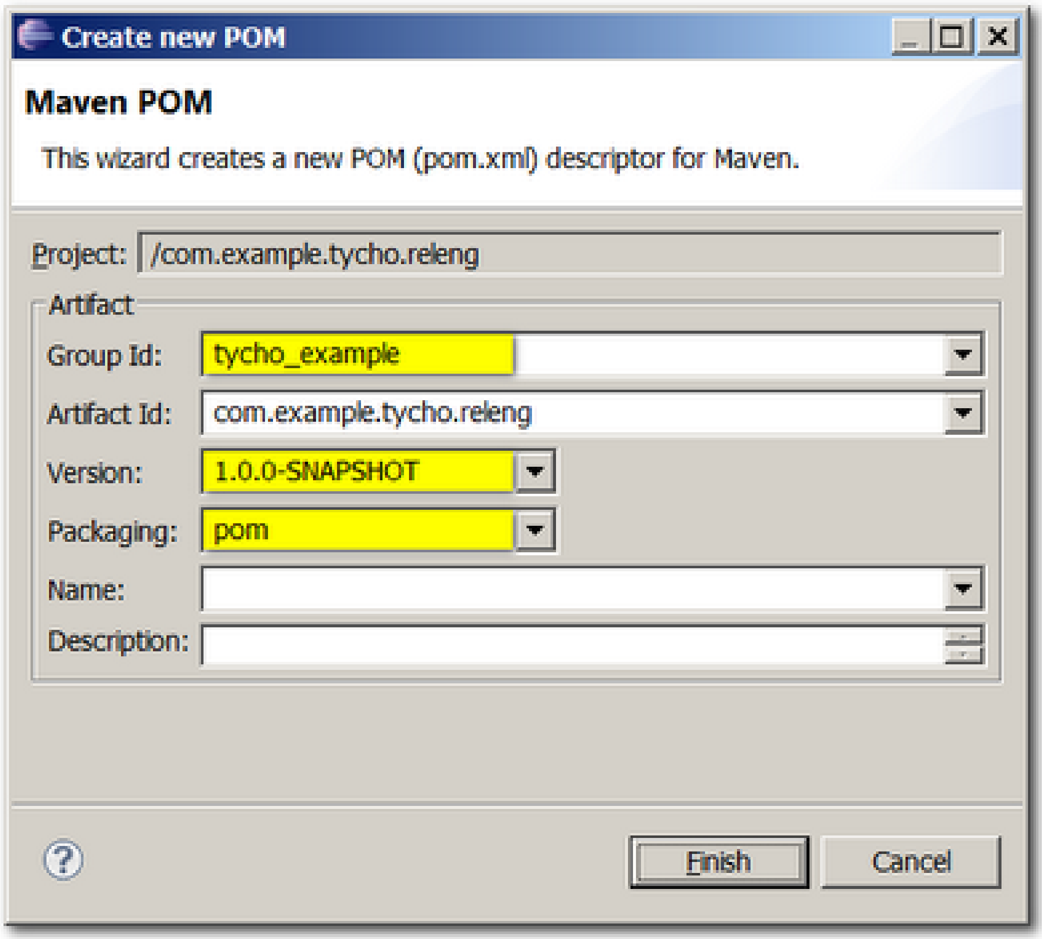
3 Tycho build 3: Creating a global build project

During the first tycho tutorial we created a pom file to store our build instructions. Some of them will repeat for all the other things we will build later on. Therefore we will refactor our first project to extract common settings to a global pom file.

Actually tycho already did something very similar for us. Open up com.example.tycho.plugin/pom.xml and look at the Effective POM tab. Our pom file is augmented with a lot of additional settings. So pom files support a cascaded approach.

3.1 Step 1: Create a generic build project

Create a new General/Project named com.example.tycho.releng and convert it to a maven project (select Configure/Convert to Maven Project from the context menu). use the same Group ID as before, set the Version to 1.0.0-SNAPSHOT and the Packaging to pom. These steps will be pretty much the same for all the projects we will build with tycho.



Maven POM

This wizard creates a new POM (pom.xml) descriptor for Maven.

Project: /com.example.tycho.releng

Artifact

Group Id:	tycho_example
Artifact Id:	com.example.tycho.releng
Version:	1.0.0-SNAPSHOT
Packaging:	pom
Name:	
Description:	

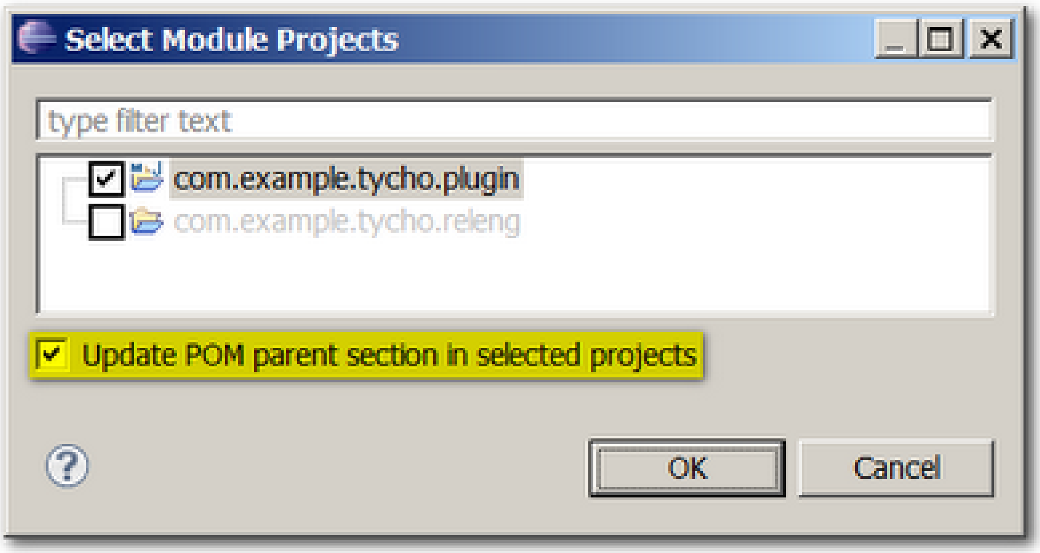
3.2 Step 2: Refactor pom files

Move the properties, repositories and build section from our com.example.tycho.plugin/pom.xml to the new one. (Remember the source code formatter works on xml files too). Of course our plug-in is unhappy now as information is missing in its pom file. Therefore we need to connect those poms somehow.

3.3 Step 3: Adding modules to poms

Open the Overview tab of com.example.tycho.releng/pom.xml and Add… a Module. Select the com.example.tycho.plugin.

Do not forget to check Update POM parent section in selected projects! This will tell the module where to get its additional information from.



[✓] com.example.tycho.plugin
[ ] com.example.tycho.releng
[✓] Update POM parent section in selected projects

After refactoring your poms should look like this:

com.example.tycho.releng/pom.xml:

```
<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>tycho_example</groupId>
 <artifactId>com.example.tycho.releng</artifactId>
 <version>1.0.0-SNAPSHOT</version>
 <packaging>pom</packaging>

 <properties>
  <tycho-version>0.16.0</tycho-version>
 </properties>

 <repositories>
  <!-- add Juno repository to resolve dependencies -->
  <repository>
   <id>Juno</id>
   <layout>p2</layout>
   <url>http://download.eclipse.org/releases/juno/</url>
  </repository>
 </repositories>

 <build>
  <plugins>
   <plugin>
    <!-- enable tycho build extension -->
    <groupId>org.eclipse.tycho</groupId>
    <artifactId>tycho-maven-plugin</artifactId>
    <version>${tycho-version}</version>
    <extensions>true</extensions>
   </plugin>
  </plugins>
 </build>
 <modules>
  <module>../com.example.tycho.plugin</module>
 </modules>
</project>
```

com.example.tycho.plugin/pom.xml:

```
<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>
 <artifactId>com.example.tycho.plugin</artifactId>
 <packaging>eclipse-plugin</packaging>
 <parent>
  <groupId>tycho_example</groupId>
  <artifactId>com.example.tycho.releng</artifactId>
  <version>1.0.0-SNAPSHOT</version>
  <relativePath>../com.example.tycho.releng</relativePath>
 </parent>
</project>
```

The parent section tells maven to integrate the parent pom when building this project.

Now we can separate global settings from project specific ones and are ready to add additional projects to our build.

Verify that your setup is correct by triggering a maven build on com.example.tycho.releng. The goals are clean install. You can still build individual plug-ins by triggering a maven run on those projects.

3.4 Step 4: Fix build warnings

Our build log reveals two different warnings we have to fix.

```
[WARNING] No explicit target runtime environment configuration. Build is platform dependent ↩
    .
...
[WARNING] Using platform encoding (Cp1252 actually) to copy filtered resources, i.e. build ↩
    is platform dependent!
```

First we configure an environment to build for. Open your master pom file and add the following code:

```
<build>
  <plugins>
   <plugin>
    <groupId>org.eclipse.tycho</groupId>
    <artifactId>target-platform-configuration</artifactId>
    <version>${tycho-version}</version>
    <configuration>
     <environments>
      <environment>
       <os>win32</os>
       <ws>win32</ws>
       <arch>x86</arch>
      </environment>
     </environments>
    </configuration>
   </plugin>
  </plugins>
 </build>
```

Of course this is only valid if you build for windows 32 bit. Of course you can build for other platforms too.

The second warning is even easier to fix. Add a property defining the source encoding:

```
<properties>
  ...
  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
 </properties>
```

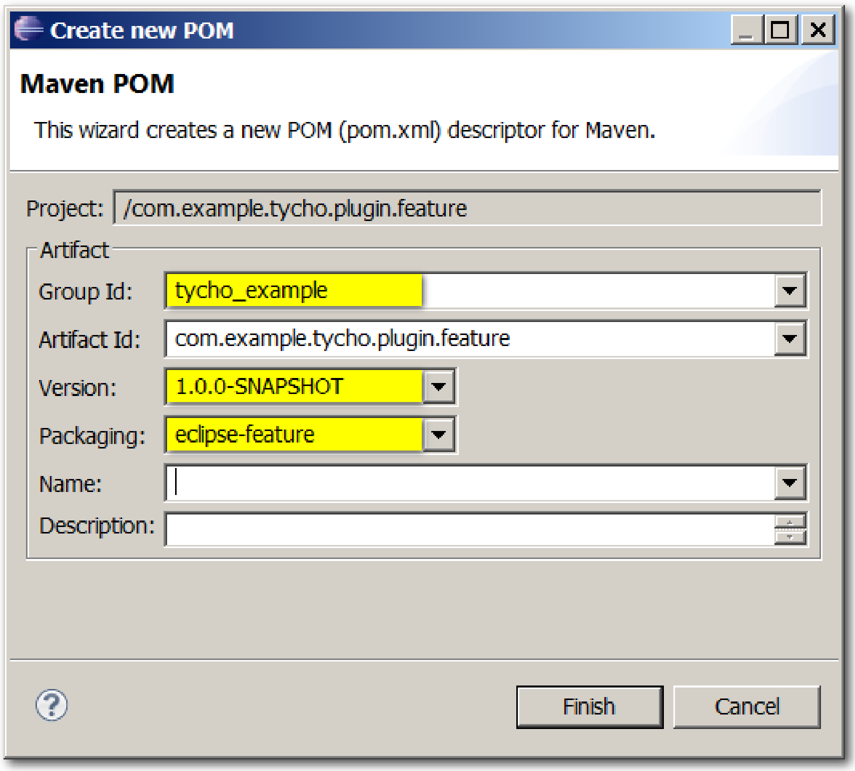
4 Tycho build 4: Building features

For the following tycho tutorials I expect you to have set up a global build project. Now we are going to build a feature, which is really easy with tycho.

Features are used to cluster bundles to convenient units. P2 will show features to your users, it will not show bundles which can be selected for install. You might want to keep your code in multiple bundles (eg UI/non UI) but still they serve the same topic (like Java programming). Then features are used to put it all together.

4.1 Step 1: Convert feature project to maven project

The feature com.example.tycho.plugin.feature already includes our plug-in com.infineon.tycho.plugin. The only thing to do is convert the feature project to a maven project. Do this using the context menu Configure/Convert to Maven Project. I guess you are familiar with the procedure right now.



Create new POM

Maven POM

This wizard creates a new POM (pom.xml) descriptor for Maven.

Project: /com.example.tycho.plugin.feature

Artifact

Group Id: tycho_example

Artifact Id: com.example.tycho.plugin.feature

Version: 1.0.0-SNAPSHOT

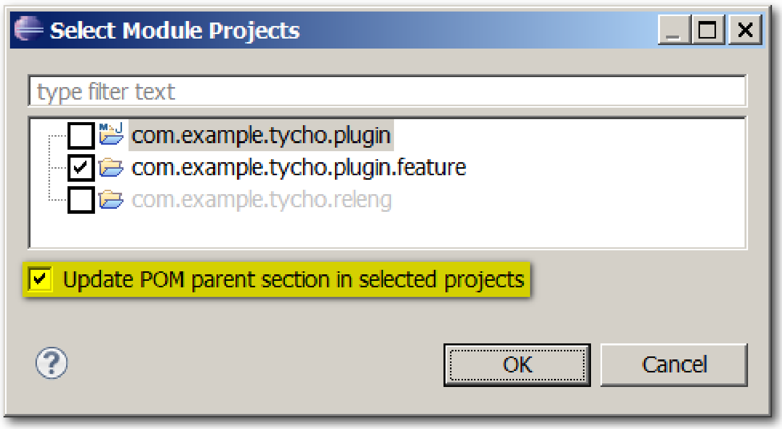
Packaging: eclipse-feature

Name:

Description:

Make sure you set Packaging to eclipse-feature.

Now open the pom file of your releng project (com.example.tycho.releng/pom.xml) and add our feature project as a new module. Don't forget to check Update POM parent section in selected projects.


Select Module Projects

type filter text

- [ ] com.example.tycho.plugin
- [x] com.example.tycho.plugin.feature
- [ ] com.example.tycho.releng

[✓] Update POM parent section in selected projects

As before you will see an error marker on your feature project. To get rid of it switch to the Problems View, locate the error and use the quick fix feature to solve it.

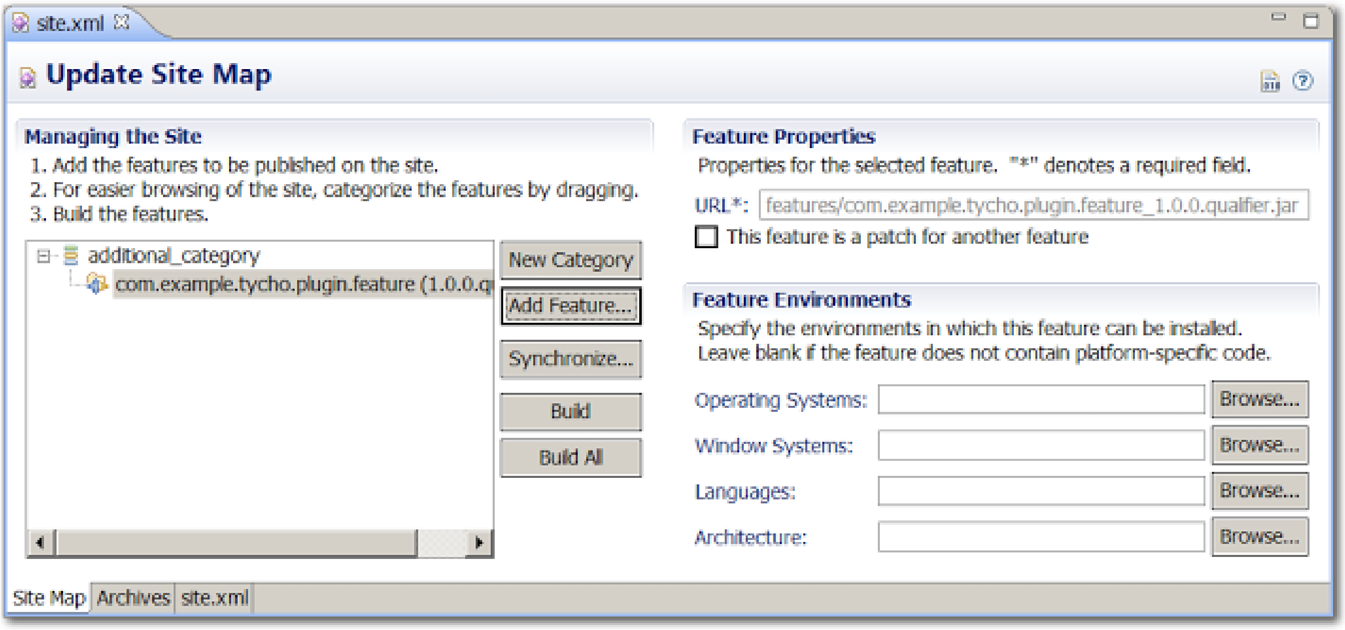
Run your build to verify your settings.

5 Tycho build 5: Building p2 update sites

Now that we can build plug-ins and features the next step will be to build an update site.

5.1 Step 1: Create an update site project

Create a new project of type Plug-in Development/Update Site Project. Name it com.example.tycho.releng.p2 and leave all the other settings with their defaults. You will end up in the Site Manifest Editor of your site.xml file. Add a New Category with some ID and a nice Name. Afterwards add the com.example.tycho.plugin.feature to the category.


site.xml

Update Site Map

Managing the Site

1. Add the features to be published on the site.
2. For easier browsing of the site, categorize the features by dragging.
3. Build the features.

additional_category
com.example.tycho.plugin.feature (1.0.0.q

Feature Properties

Properties for the selected feature. "*" denotes a required field.

URL*: features/com.example.tycho.plugin.feature_1.0.0.qualifier.jar

[ ] This feature is a patch for another feature

Feature Environments

Specify the environments in which this feature can be installed.
Leave blank if the feature does not contain platform-specific code.

Operating Systems:

Window Systems:

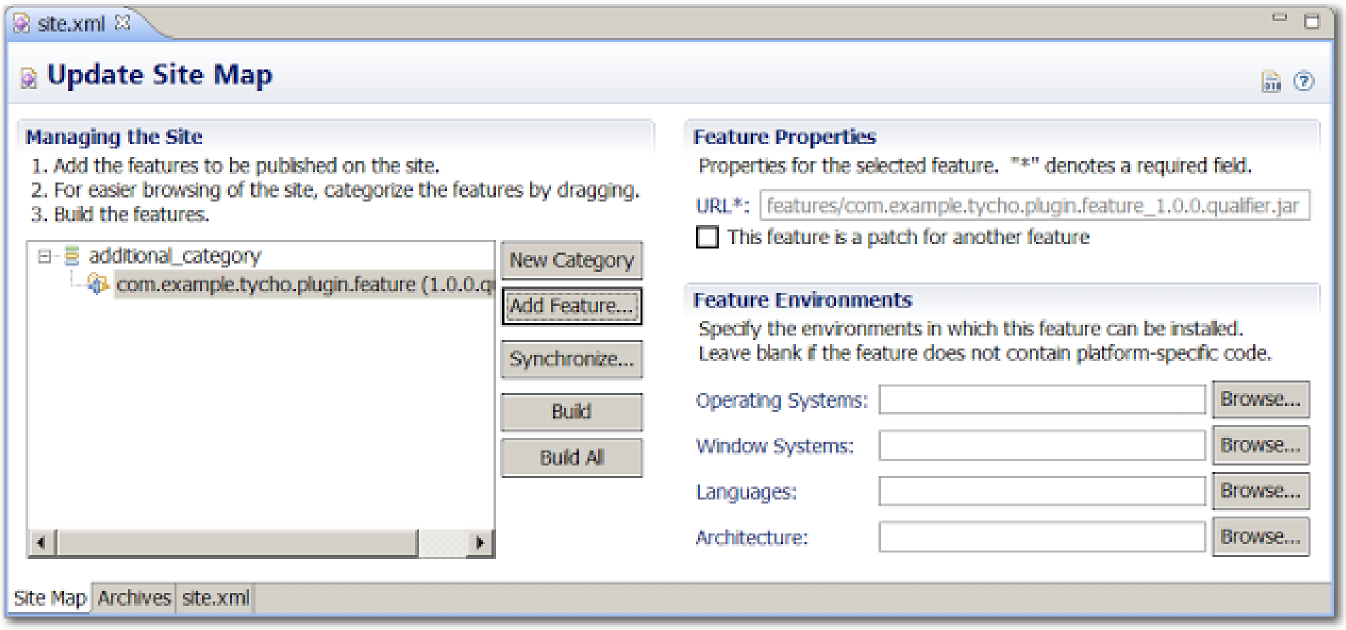
Languages:

Architecture:

5.2 Step 2: Convert to maven project

Tycho expects the update site content to be stored in a file called category.xml. So rename site.xml to that name. You can still use the Site Manifest Editor to update your site contents afterwards.

Now convert the p2 project to a maven project. The procedure is the same as before, only Packaging should be set to eclipse-repository.


site.xml
Update Site Map
Managing the Site
1. Add the features to be published on the site.
2. For easier browsing of the site, categorize the features by dragging.
3. Build the features.
additional_category
com.example.tycho.plugin.feature (1.0.0.q
New Category
Add Feature...
Synchronize...
Build
Build All
Feature Properties
Properties for the selected feature. "*" denotes a required field.
URL*: features/com.example.tycho.plugin.feature_1.0.0.qualifier.jar
This feature is a patch for another feature
Feature Environments
Specify the environments in which this feature can be installed.
Leave blank if the feature does not contain platform-specific code.
Operating Systems:
Window Systems:
Languages:
Architecture:
Browse...
Site Map
Archives
site.xml

Switch to your com.example.tycho.releng/pom.xml and add the com.example.tycho.releng.p2 project as a module. Remember to check Update POM parent section in selected projects. Fix the build error as before and run your maven build.

You can find your p2 update site in com.example.tycho.releng.p2/target/repository. If you like you can immediately use this location to install your feature in your running eclipse instance.

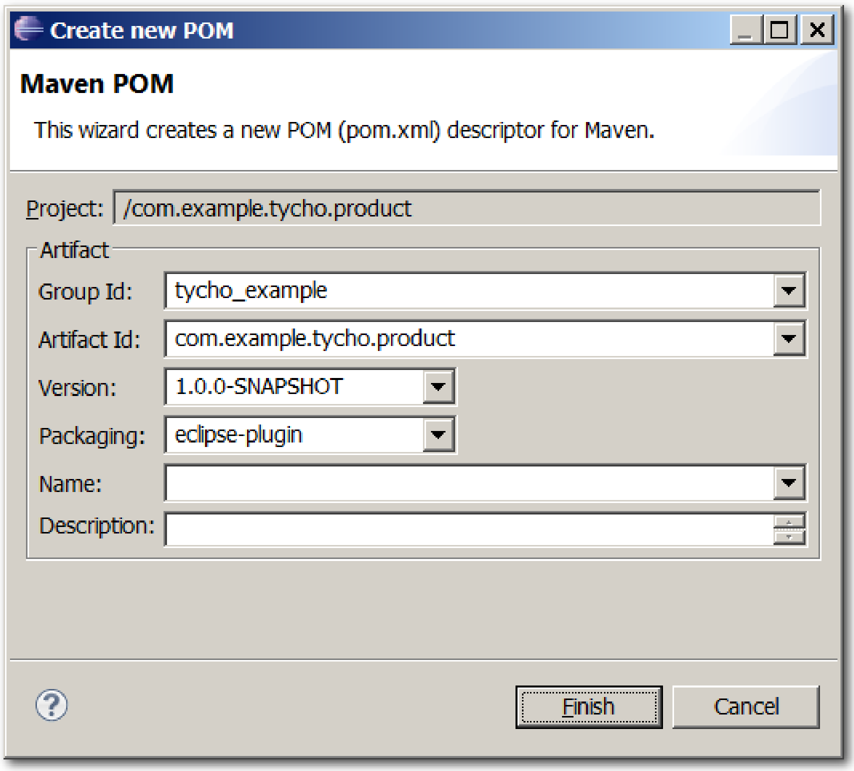
6 Tycho build 6: Building products

Now that we can build almost everything, there is just one step missing: building an RCP application.

Step 1: Convert required projects

Our example product has additional dependencies to com.example.tycho.product and com.example.tycho.product.feature which are not part of our maven build yet. To build our product we need to convert them first.

So convert those two projects to maven projects. Afterwards add both of them to our releng pom file. Nothing new so far.



Maven POM

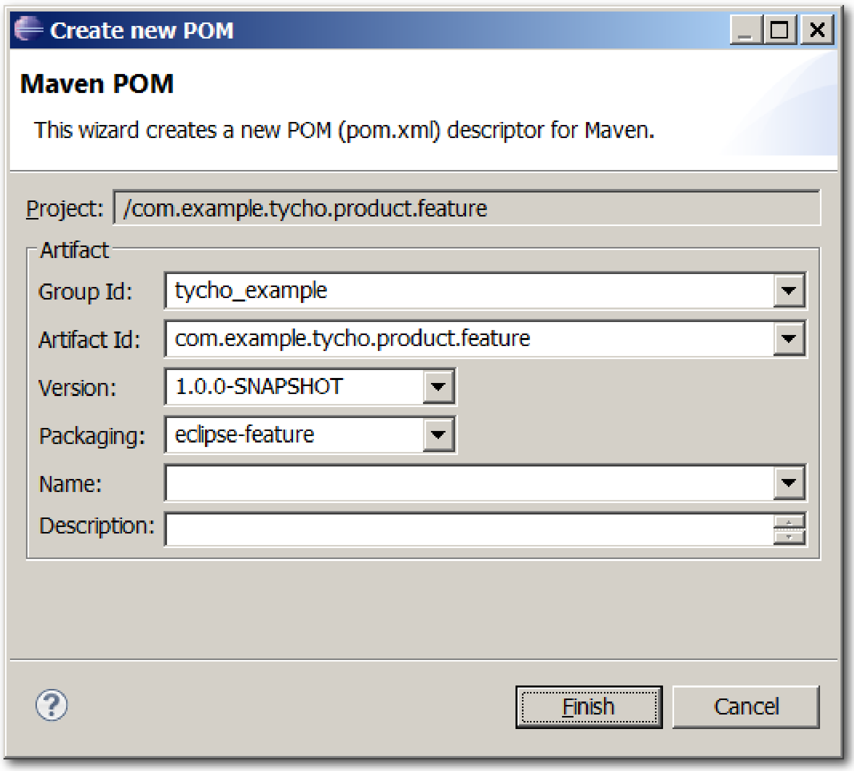
This wizard creates a new POM (pom.xml) descriptor for Maven.

Project: /com.example.tycho.product

Artifact

Group Id:	tycho_example
Artifact Id:	com.example.tycho.product
Version:	1.0.0-SNAPSHOT
Packaging:	eclipse-plugin
Name:	
Description:	

Remember to remove the build section from the pom file for eclipse-plugins.



Maven POM

This wizard creates a new POM (pom.xml) descriptor for Maven.

Project: /com.example.tycho.product.feature

Artifact

Group Id:	tycho_example
Artifact Id:	com.example.tycho.product.feature
Version:	1.0.0-SNAPSHOT
Packaging:	eclipse-feature
Name:	
Description:	

6.1 Step 2: Create a project for our product

Create a new General/Project called com.example.tycho.releng.product. Convert it to a maven project with Packaging set to eclipse-repository. Add the new project to our releng pom file as a module like we did for all the other projects.

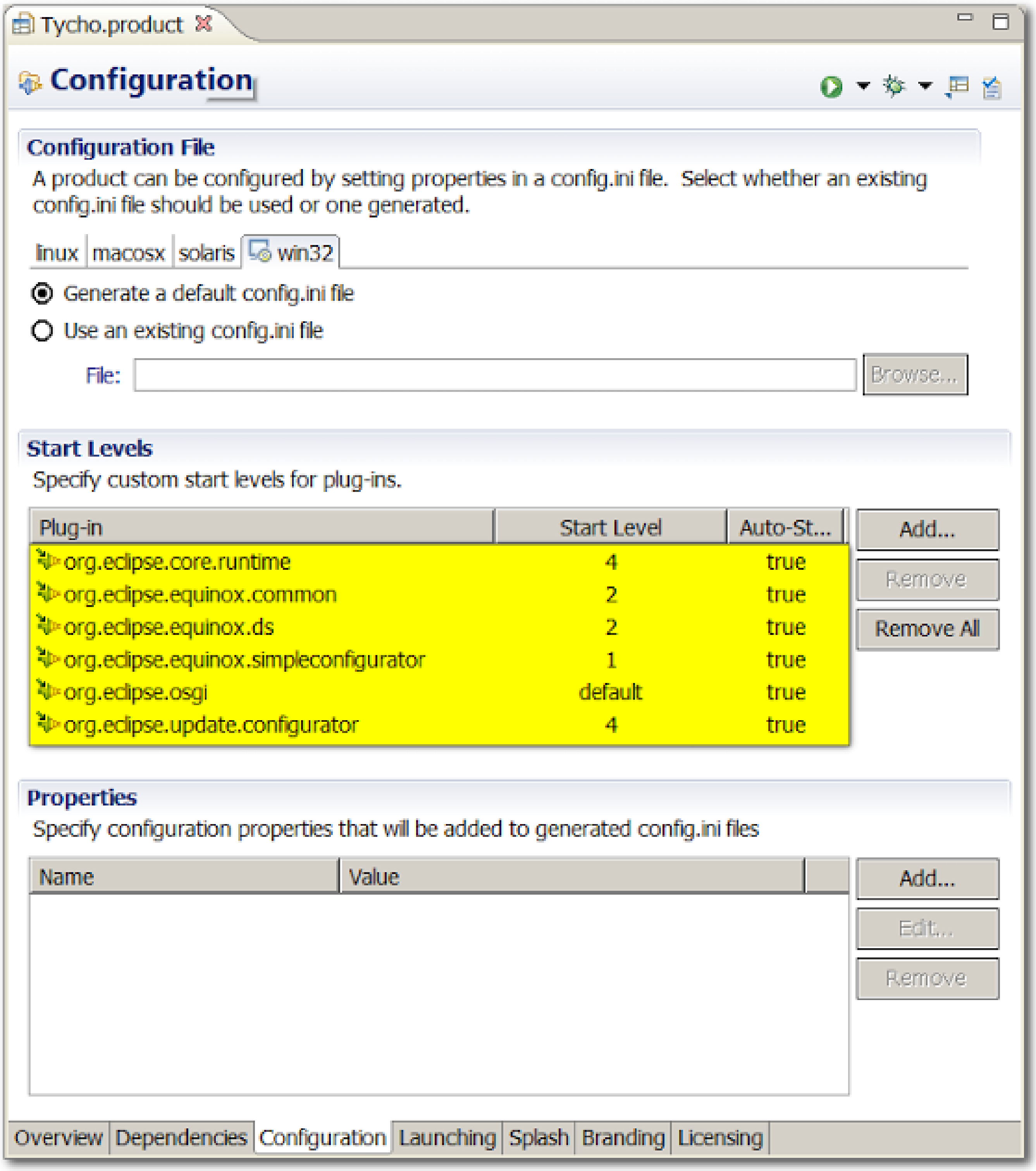
Now move the file com.example.tycho.product/Tycho.product to the root folder of our newly created project. Tycho will not pick up product files by default, so we need to adjust our com.example.tycho.releng.product/pom.xml file a little. Add the following section within the project node:

```
<build>
  <plugins>
   <plugin>
    <groupId>org.eclipse.tycho</groupId>
    <artifactId>tycho-p2-director-plugin</artifactId>
    <version>${tycho-version}</version>
    <executions>
     <execution>
      <!-- install the product using the p2 director -->
      <id>materialize-products</id>
      <goals>
       <goal>materialize-products</goal>
      </goals>
     </execution>
     <execution>
      <!-- create zip file with the installed product -->
      <id>archive-products</id>
      <goals>
       <goal>archive-products</goal>
      </goals>
     </execution>
    </executions>
   </plugin>
  </plugins>
 </build>
```

6.2 Step 3: Set start levels of your product bundles

There is one more step to take before we can run the build. It seems that PDE build (the thing that runs when you export an RCP product) adds some magic regarding bundle start levels. In fact it adjusts some autostart settings which we need to teach tycho manually.

Open your product definition file and switch to the Configuration tab. Add all plug-ins from the screenshot and set their start levels accordingly.



Configuration

Configuration File

A product can be configured by setting properties in a config.ini file. Select whether an existing config.ini file should be used or one generated.

linux | macosx | solaris | win32

[✓ Generate a default config.ini file]

Use an existing config.ini file

File:

Start Levels

Specify custom start levels for plug-ins.

Plug-in	Start Level	Auto-St...
org.eclipse.core.runtime	4	true
org.eclipse.equinox.common	2	true
org.eclipse.equinox.ds	2	true
org.eclipse.equinox.simpleconfigurator	1	true
org.eclipse.osgi	default	true
org.eclipse.update.configurator	4	true

Properties

Specify configuration properties that will be added to generated config.ini files

Name	Value

Save your product and start the build process. You should find your product in com.example.tycho.releng.product/target/products/tycho.p
There will also be a zipped version available in the target/products folder.

If you have problems with the startup levels for your build, then use the Eclipse Product export wizard from the Overview tab of your product file. Switch to the folder where you exported your product to and open the file configuration/org.eclipse.equinox.simpleconf
Each line holds an entry of type

bundle_name,version,location,startlevel,autostart

Find all bundles where autostart is set to true and add them in your product configuration file with their according start levels.

6.3 Optional: Adding p2.inf files

If you want to add additional p2 information to your product build, place your p2 file in the same folder as your project file and name it <project_name>.p2.inf.

Eg. create a file com.example.tycho.releng.product/Tycho.p2.inf with following content to add the Juno repository to the preconfigured update sites:

```
instructions.configure=\
org.eclipse.equinox.p2.touchpoint.eclipse.addRepository(type:0,location:http${#58}// ↩
    download.eclipse.org/releases/juno,name:Juno);\
org.eclipse.equinox.p2.touchpoint.eclipse.addRepository(type:1,location:http${#58}// ↩
    download.eclipse.org/releases/juno,name:Juno);
```

6.4 Optional: Build for multiple platforms

Adding another platform is simple: just ad a new environment to your master pom file:

```
<environment>
       <os>macosx</os>
       <ws>cocoa</ws>
       <arch>x86_64</arch>
</environment>
```

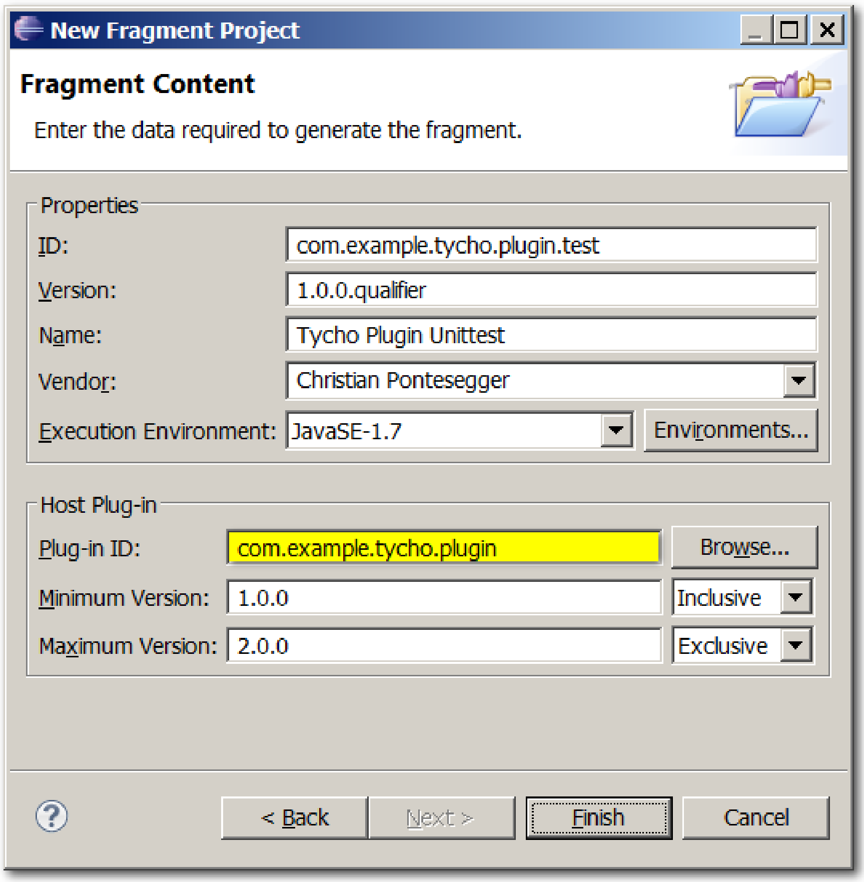
7 Tycho build 7: Plug-in unit tests

During the previous tutorials we focused on building stuff from our productive code. But good development is test driven and requires a lot of unit testing. Tycho can integrate tests in the build process which automatically gives you test coverage on every build you do.

Step 1: Create a sample unit test

Unit tests for plug-ins should normally be provided as a fragment to the plug-in under test. Fragments allow to access all classes of the main plug-in without exporting them. If you use the same package name for your unit test class and the class under test you can even access package private methods.

Create a new Fragment Project named com.example.tycho.plugin.test. Set the Host Plug-in ID to com.example.tycho.plugin and adjust the version ranges accordingly.



ID: com.example.tycho.plugin.test
Version: 1.0.0.qualifier
Name: Tycho Plugin Unittest
Vendor: Christian Pontesegger
Execution Environment: JavaSE-1.7
Plug-in ID: com.example.tycho.plugin
Minimum Version: 1.0.0 Inclusive
Maximum Version: 2.0.0 Exclusive

Create a new JUnit Test Case named com.example.tycho.plugin.ExampleViewTest.java with following content:

```
package com.example.tycho.plugin;

import static org.junit.Assert.*;

import org.junit.Test;

public class ExampleViewTest {

 @Test
 public void testID() {
  assertEquals("com.example.tycho.views.example", ExampleView.VIEW_ID);
 }
}
```

I admit, this is a very stupid test, but this tutorial is not about writing good unit tests, right?

7.1 Step 2: Add test fragment to tycho build

Once again convert your fragment project to a maven project. Set Packaging to eclipse-test-plugin and like before add the project to our master pom file.

That's it! Build your product and your tests will automatically run as part of your build process. Try changing the assertion to provoke an error.

The tycho surefire plugin works a bit differently compared to a JUnit Plug-in Test. Surefire will only load plugins that are referenced via dependencies by the test plugin. this means it will start the minimal set of bundles defined by the dependency tree. In contrary JUnit Plug-in Tests will load everything available in your workspace.

Check the documentation to see how to add additional dependencies.

Tycho Surefire tests run in the integration-test phase while regular Surefire tests run in the test phase. The integration-test phase occurs between the package and install phases.

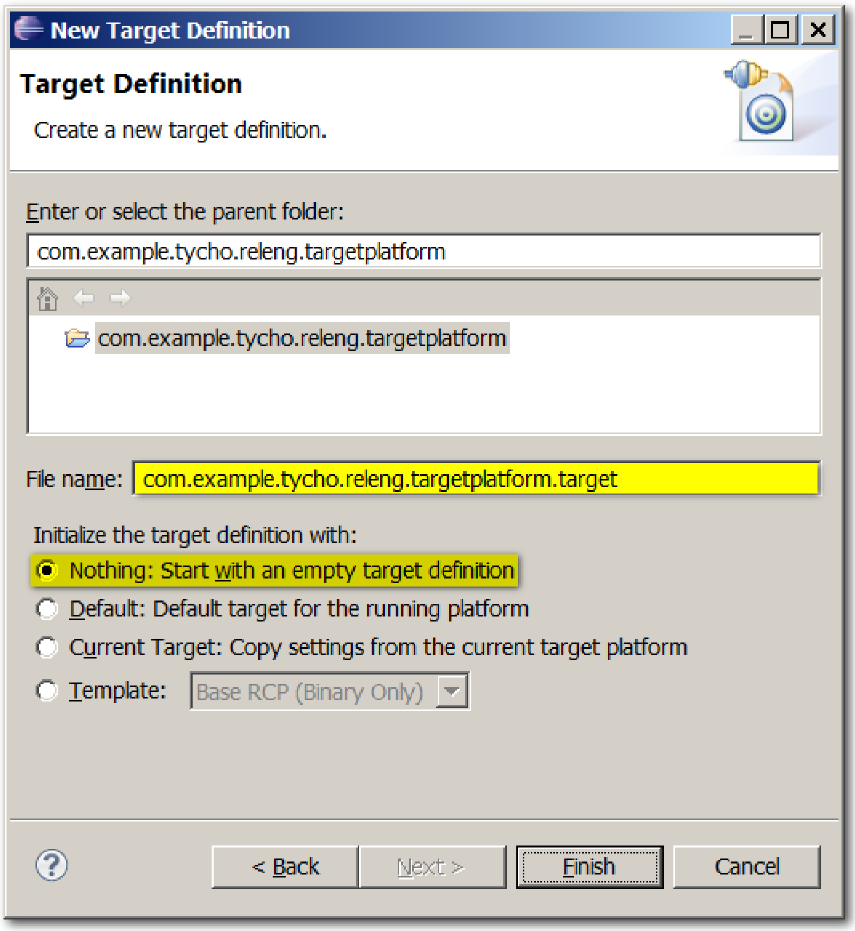
8 Tycho build 8: Using a target platform

A target platform describes, which features and plug-ins are used to build your product. It further defines where to look for those components.

Using a target platform makes your build reproducible and therefore predictable. Try to use it right from the beginning of your product development.

8.1 Step 1: Creating a target platform

Create a new General/Project named com.example.tycho.releng.targetplatform. Further create a new Target Definition file named com.example.tycho.releng.targetplatform.target within that project. We want to Start with an empty target definition.



New Target Definition

Target Definition

Create a new target definition.

Enter or select the parent folder:

com.example.tycho.releng.targetplatform

com.example.tycho.releng.targetplatform

File name: com.example.tycho.releng.targetplatform.target

Initialize the target definition with:

[✓ Nothing: Start with an empty target definition]

Default: Default target for the running platform

Current Target: Copy settings from the current target platform

Template: Base RCP (Binary Only)

Once the Target Editor opens Add… a new location of type Software Site. On the next page you can either create a new site entry or simply enter http://download.eclipse.org/releases/juno under Work with. Uncheck Group by Category to see a list of all features. Now add following features:

- Eclipse e4 Rich Client Platform
- Eclipse Platform

- Eclipse Platform Launcher Executables
- Eclipse Java Development Tools

The Java Development Tools are only needed for JUnit dependencies. Before you click Finish make sure that the filter textbox is empty. Below the table you should see the text 4 items selected. As long as the filter is enabled previously selected features that are hidden by the filter will not be considered.

Back in the Target Editor click the link in the upper right corner Set as Target Platform to activate it.

8.2 Step 2: Integrate in maven build

Convert the project to a maven project with Packaging set to eclipse-target-definition. Then add it to your master pom. Now we need to change the master pom a little: remove the repositories section as we want to use our target platform instead of the Juno repository. To activate the target definition in maven add (or modify) a new plugin section:

```
<plugin>
    <groupId>org.eclipse.tycho</groupId>
    <artifactId>target-platform-configuration</artifactId>
    <version>${tycho-version}</version>
    <configuration>
     <target>
      <artifact>
       <groupId>tycho_example</groupId>
       <artifactId>com.example.tycho.releng.targetplatform</artifactId>
       <version>1.0.0-SNAPSHOT</version>
      </artifact>
     </target>
     <environments>
      <environment>
       <os>win32</os>
       <ws>win32</ws>
       <arch>x86</arch>
      </environment>
     </environments>
    </configuration>
   </plugin>
```

If you followed my previous tutorials you will only need to add the target section as the rest should already be part of your pom file. The target/artifact/artifactId node refers to the name of the project, that contains the target definition file.

Your build now uses the target definition.

9 Tycho build 9: Updating version numbers

When you start releasing your products you need to start dealing with version numbering of your plug-ins and features. It is generally a good idea to do this the eclipse way. When you use API Tooling and API baselines eclipse will automatically tell you when to increase your versions. The problem is: once you increase your plug-in version number you have to adapt your maven version too. Otherwise your build will fail.

Keeping version numbers consistent is a tedious task. But help is on the way: let tycho do this for you!

9.1 Step 1: Add tycho-versions plugin

Open your master pom file and add a new plugin section within the project/build/plugins node:

```
<plugin>
 <groupId>org.eclipse.tycho</groupId>
 <artifactId>tycho-versions-plugin</artifactId>
 <version>${tycho-version}</version>
</plugin>
```

9.2 Step 2: Execute new maven target

Go to the project com.example.tycho.releng, right click and select Run As/Maven build.... Set tycho-versions:update-pom in Goals and execute maven. Now all your pom file version numbers will be updated according to the plug-in and feature version numbers.