





Tutorial

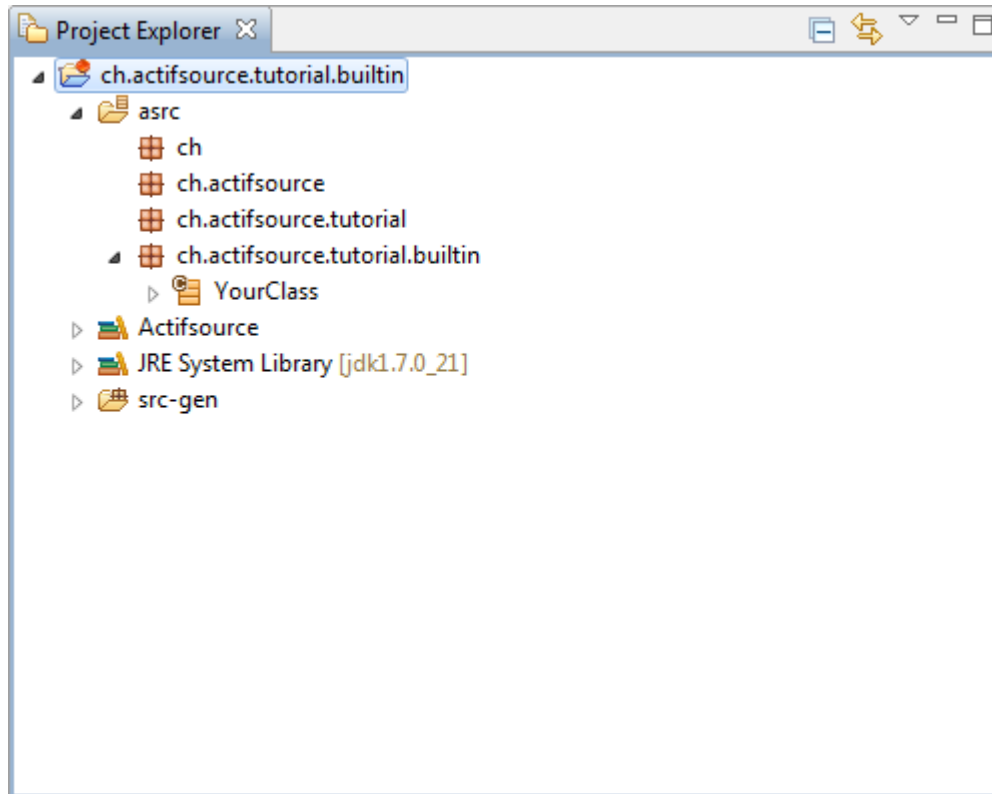
Builtin Models

Tutorial	Actifsource Tutorial – Builtin-Models
Required Time	<ul style="list-style-type: none"> • 45 Minutes
Prerequisites	<ul style="list-style-type: none"> • Actifsource Tutorial – Installing Actifsource • Actifsource Tutorial – Simple Service
Goal	<ul style="list-style-type: none"> • Creating an eclipse plugin containing an actifsource builtin model
Topics covered	<ul style="list-style-type: none"> • Create an actifsource project • Convert an actifsource project to a plugin project • Define the exported resource folders • Define a builtin • Create a feature and setup an updatesite • Install the example feature • Using the builtin
Notation	<ul style="list-style-type: none"> •  To do •  Information • Bold: Terms from actifsource or other technologies and tools • <u>Bold underlined</u>: actifsource Resources • <u>Underlined</u>: User Resources • <u><i>UnderlinedItalics</i></u>: Resource Functions • <code>Monospaced</code>: User input • <i>Italics</i>: Important terms in current situation
Disclaimer	<p>The authors do not accept any liability arising out of the application or use of any information or equipment described herein. The information contained within this document is by its very nature incomplete. Therefore the authors accept no responsibility for the precise accuracy of the documentation contained herein. It should be used rather as a guide and starting point.</p>
Contact	<p>actifsource GmbH Täferenstrasse 37 5405 Baden-Dättwil Switzerland www.actifsource.com</p>
Trademark	<p>actifsource is a registered trademark of actifsource GmbH in Switzerland, the EU, USA, and China. Other names appearing on the site may be trademarks of their respective owners.</p>

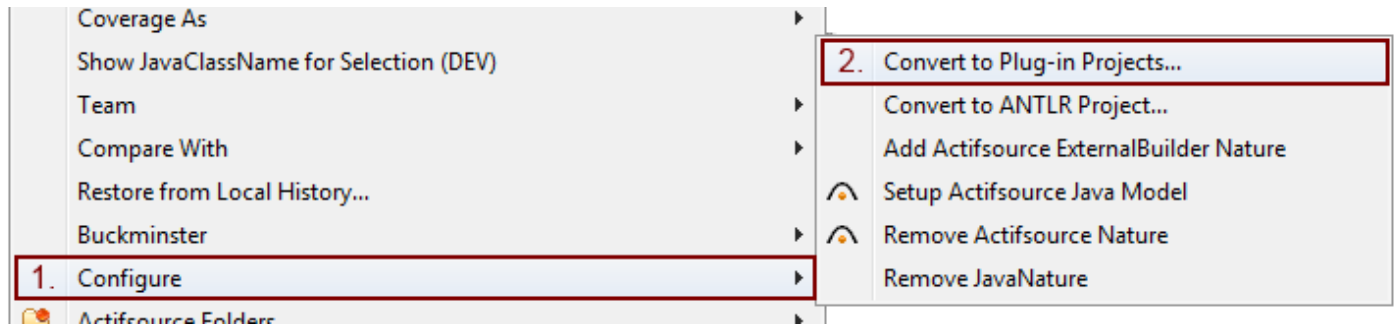
- ① Learn how to specify an actifsource Builtin
 - Create an actifsource plugin
 - Define the extensions needed to provide a model as Builtin
 - Create a feature and a updatesite for deploying the plugin
- ① Install the feature and use builtin

Create an actifsource project

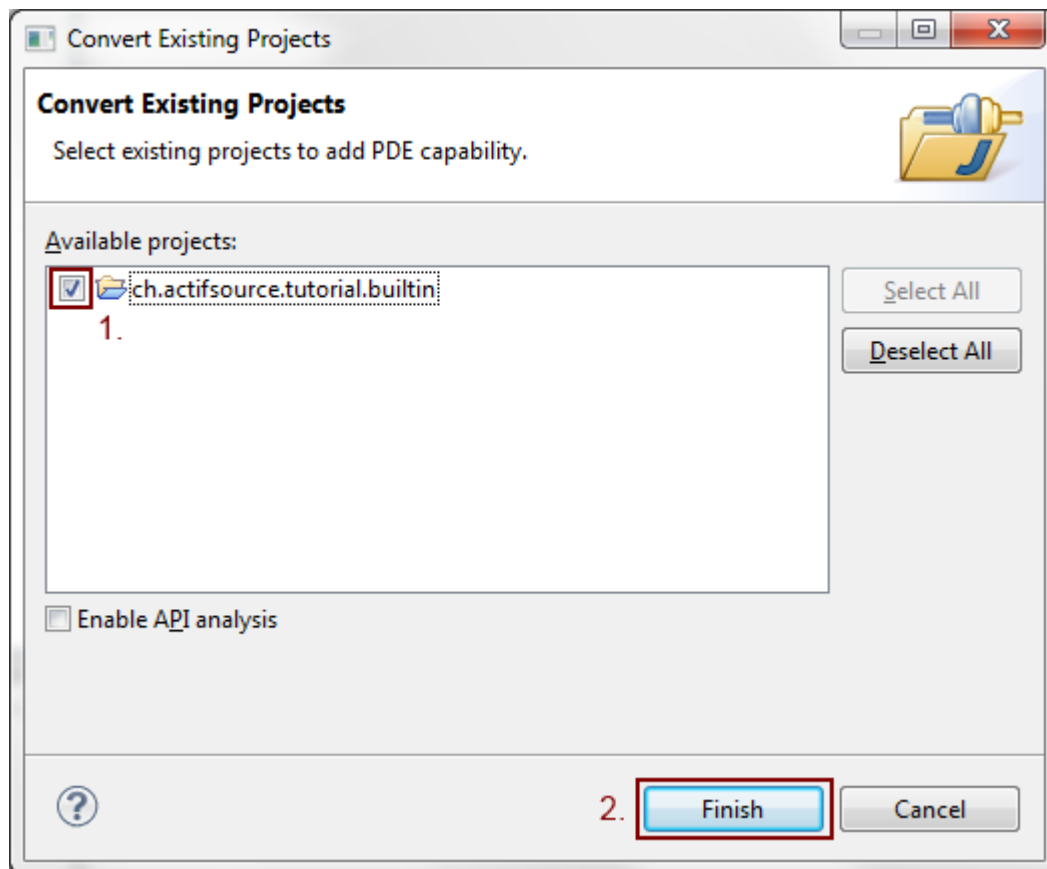
- ① Since creating a builtin doesn't make sense without having a model, just create an **actifsource project** name `ch.actifsource.tutorial.builtin` with some resources.



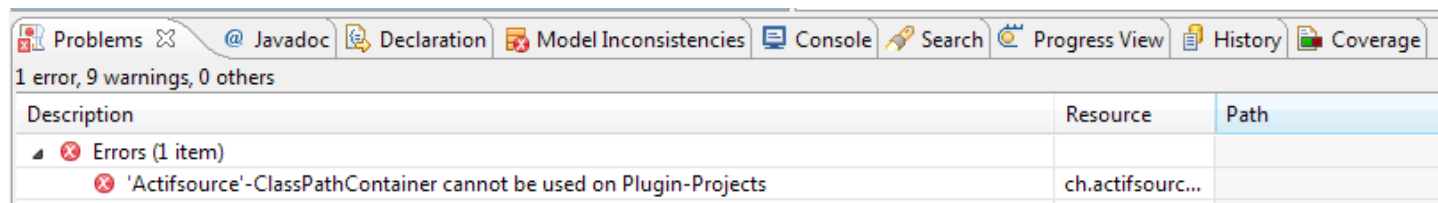
Convert an actifsource project into a plugin project

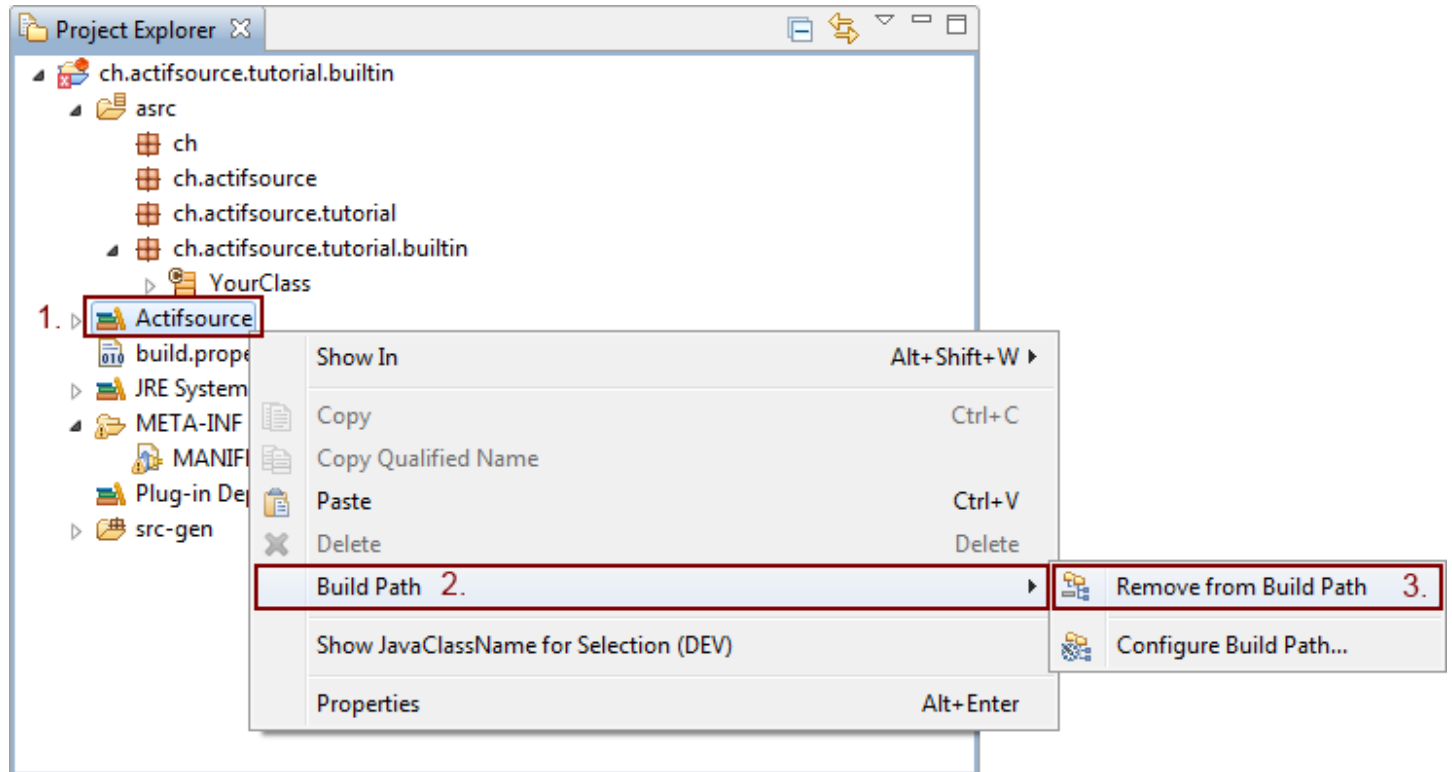


- ↖ Right click on the project `ch.actifsource.tutorial.builtin` in the navigator to open the context menu.
- ↖ Go into the submenu **Configure** and click on **Convert to Plug-in Projects...**

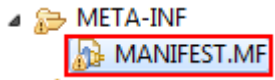


- ↖ Check the checkbox in front of ch.actifsource.tutorial.builtin
- ↖ Click **Finish**
- ↖ This will convert the project however there will be some error and warning needed to be fixed.

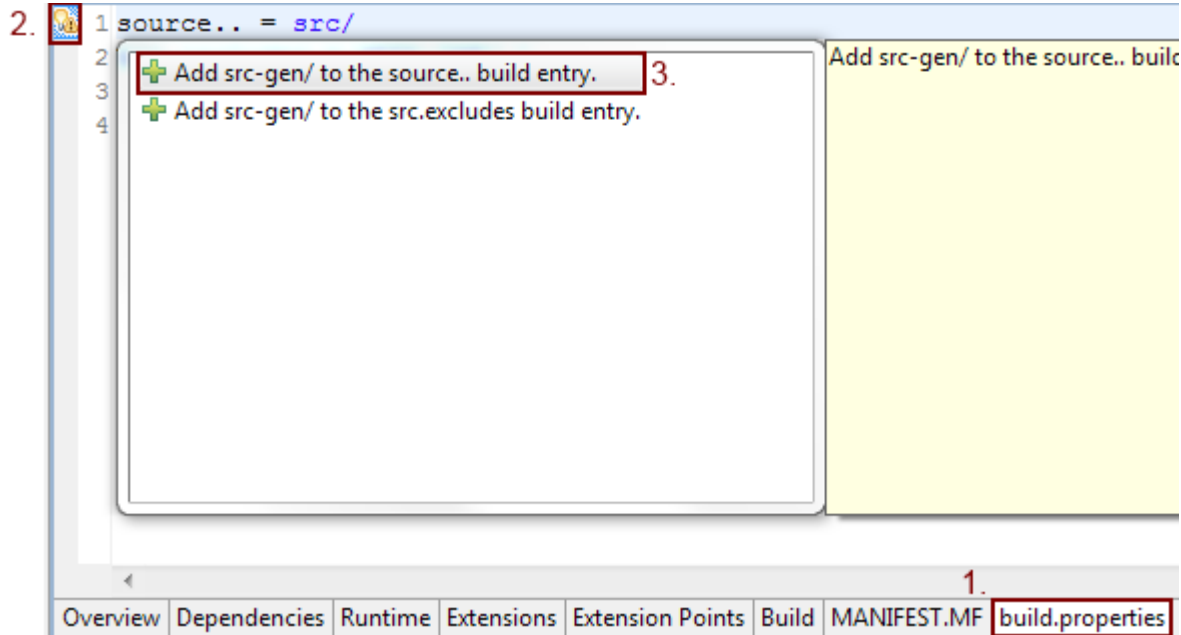




- ↖ Right click on the **Actifsource** classpath container
- ↖ Go into the submenu **Build Path** and select **Remove from Build Path**



↵ Double click on the MANIFEST.MF to open the manifest editor



↵ Go to the **build.properties** tab

↵ Click on the yellow warning sign left to the source.. line

↵ Select **Add src-gen/ to the source.. build entry**

ⓘ This will add the src-gen folder to the source build when exporting a plugin including source code. This is not a requirement to export the actifsource model as builtin however we want to get rid of this warning

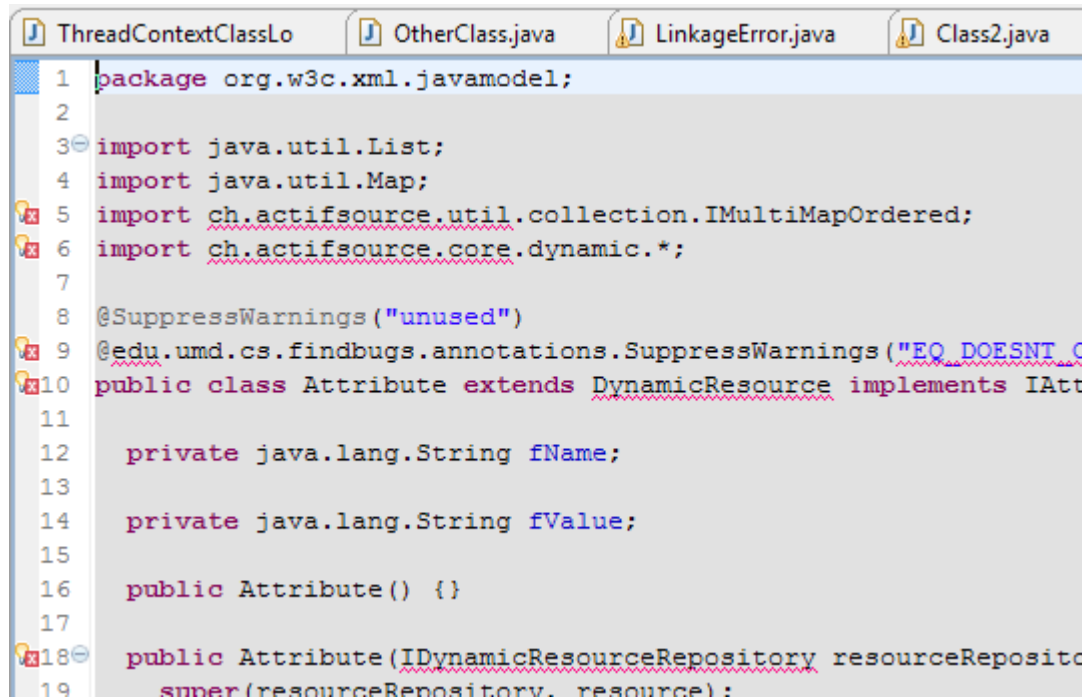
ⓘ There may be more warning depending on your eclipse configuration, you may fix them the same way if you like.

The screenshot shows the Eclipse IDE's **Dependencies** tab. The **Required Plug-ins** section is active, with the instruction: "Specify the list of plug-ins required for the operation of this plug-in." To the right of this section are buttons: **Add...** (labeled 2.), **Remove**, **Up**, **Down**, and **Properties...**. Below these buttons is a "Total: 0" indicator.

An inset window titled **Plug-in Selection** is shown. It has a search field labeled "Select a Plug-in:" containing the text `ch.actifsource.cor` (labeled 3.). Below the search field is a list of "Matching items:" with one item selected: `ch.actifsource.core (5.8.0.qualifier)` (labeled 4.). At the bottom of the inset window are **OK** (labeled 5.) and **Cancel** buttons. A question mark icon is also present.

At the bottom of the IDE, the **Dependencies** tab is selected in the breadcrumb navigation (labeled 1.). Other tabs include Overview, Runtime, Extensions, Extension Points, Build, MANIFEST.MF, plugin.xml, and build.

- ↵ Activate the **Dependencies** tab
- ↵ Click on **Add...**
- ↵ Enter `ch.actifsource.cor` in the textbox to start filtering
- ↵ Select **ch.actifsource.core**
- ↵ Click on **OK**

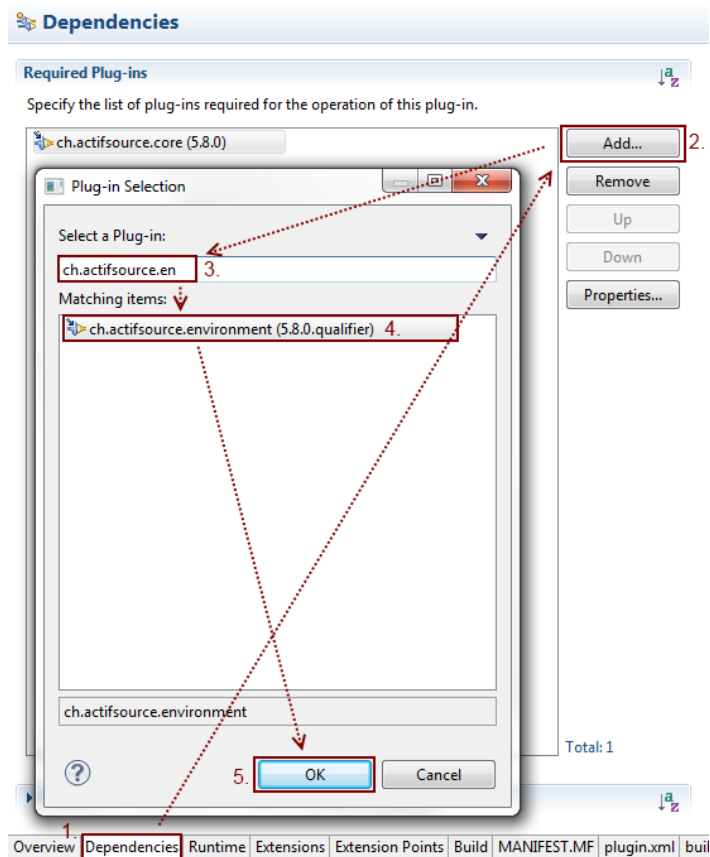


```
1 package org.w3c.xml.javamodel;
2
3 import java.util.List;
4 import java.util.Map;
5 import ch.actifsource.util.collection.IMultiMapOrdered;
6 import ch.actifsource.core.dynamic.*;
7
8 @SuppressWarnings("unused")
9 @edu.umd.cs.findbugs.annotations.SuppressWarnings("EQ_DOESNT_OVERRIDE_EQUALS")
10 public class Attribute extends DynamicResource implements IAttribute {
11
12     private java.lang.String fName;
13
14     private java.lang.String fValue;
15
16     public Attribute() {}
17
18     public Attribute(IDynamicResourceRepository resourceRepository,
19                     DynamicResource resource) {
```

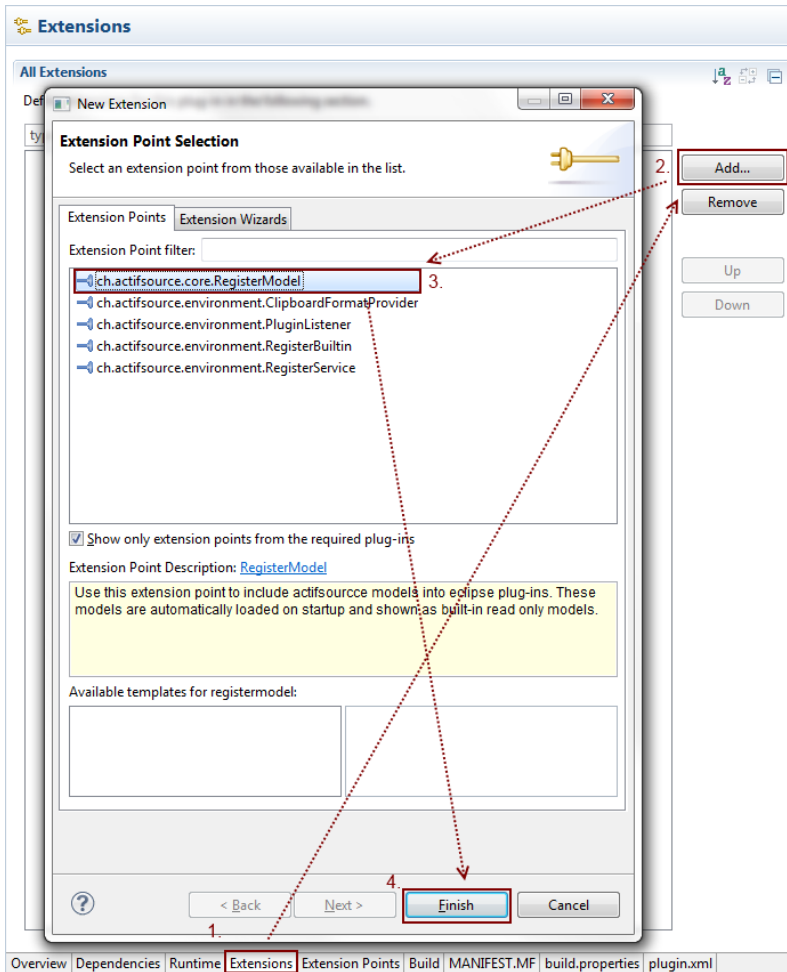
- ① If you do the conversion on an existing project, you may need to add additional dependencies. You might see this by looking into the generated files, the actifsource project dependencies and the java build path.
- ① When exporting a model as plugin, you have to make sure that you have a plugin dependency to all required projects. This means all required projects need to be converted into plugins.

Define the exported resource folders

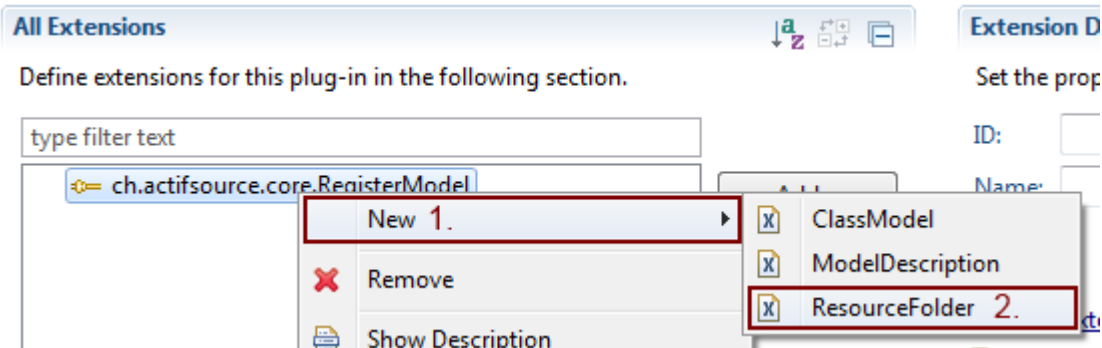
- ① Before defining exports you need to have access to the required extension point declarations. This is done by adding a dependency to **ch.actifsource.environment**.



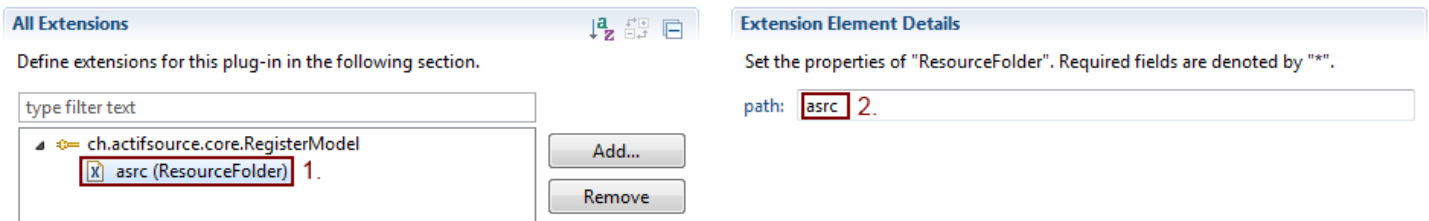
- ↵ Go back to the **Dependencies** tab in the manifest editor
- ↵ Click on **Add...**
- ↵ Enter `ch.actifsource.en` to start filtering
- ↵ Select **ch.actifsource.environment**
- ↵ Click on **OK**



- ↵ Activate the **Extension** tab
- ↵ Click on **Add...**
- ↵ Select **ch.actifsource.core.RegisterModel**
- ↵ Click on **Finish**



- ① You don't need to specify an **ID** and **Name** for the extension, we leave it empty.
- ① Open the context menu by right click on the extension
- ① Go into the Submenu **New** and select **ResourceFolder**




- ① Select the empty (ResourceFolder) entry
- ① Enter the project relative path `asrc` to the resource folder
- ① You may define different resource folders as defined in your actifsource project configuration. This is useful if you want to write templates inside your plugin to generated code and don't want to deliver the template and related resources. Just put the templates in a separate resourcefolder which is not registered in the extension.

Build Configuration

 Custom Build

Runtime Information

Define the libraries, specify the order in which they should be built, and list the source folders that should be compiled into each selected library.

	<input type="button" value="Add Library..."/>	<input type="button" value="Add Folder..."/>
	<input type="button" value="Up"/>	
	<input type="button" value="Down"/>	

Binary Build

Select the folders and files to include in the binary build.

<input type="checkbox"/> .asproject
<input type="checkbox"/> .classpath
<input type="checkbox"/> .project
<input type="checkbox"/> .settings
<input checked="" type="checkbox"/> META-INF
<input checked="" type="checkbox"/> <u>asrc</u> 2.
<input type="checkbox"/> bin
<input type="checkbox"/> build.properties
<input checked="" type="checkbox"/> plugin.xml
<input type="checkbox"/> src-gen

Source Build

Select the folders and files to include in the source build.

<input type="checkbox"/> .asproject
<input type="checkbox"/> .classpath
<input type="checkbox"/> .project
<input type="checkbox"/> .settings
<input type="checkbox"/> META-INF
<input type="checkbox"/> asrc
<input type="checkbox"/> bin
<input type="checkbox"/> build.properties
<input checked="" type="checkbox"/> plugin.xml
<input type="checkbox"/> src-gen


Extra Classpath Entries

Overview | Dependencies | Runtime | Extensions | Extension Points | **Build** | MANIFEST.MF | build.properties | plugin.xml

1. Switch to **Build** tab
1. Check the asrc-Folder in the binary build
1. This is needed to include the asrc-Folder in the binary release, if you fail to do so the plugin won't contain any resources.

Define a builtin

- ① A builtin provides a way to define dependencies to actifsource model defined in a plugin inside a non-plugin projects. This step is not required if you want to use your model only in plugin projects.

 **Overview**

General Information
This section describes general information about this plug-in.

ID:

Version:

Name:

Vendor:

Platform Filter:

Activator:

Activate this plug-in when one of its classes is loaded

This plug-in is a singleton

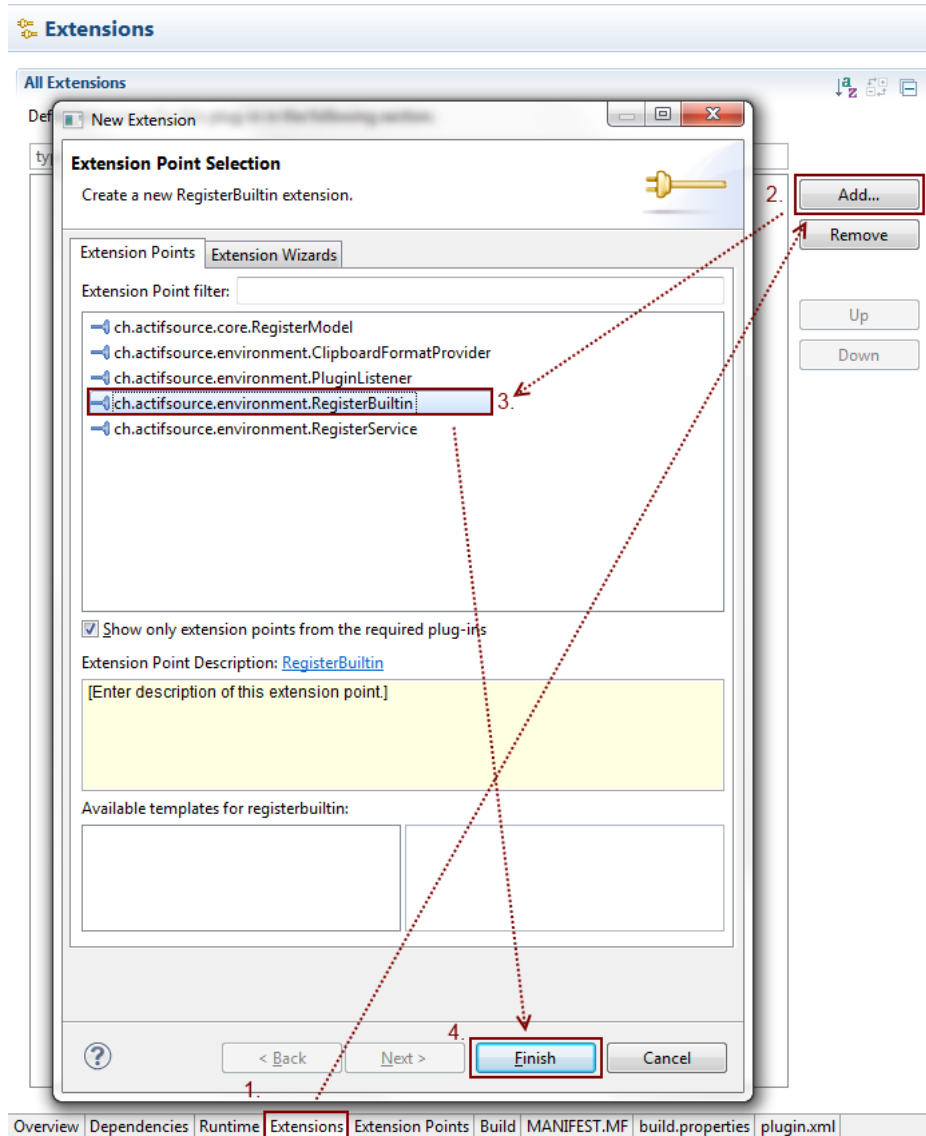
Execution Environments
Specify the minimum execution environments required to run this plug-in.

[Configure JRE associations...](#)

[Update the classpath settings](#)

1. **Overview** | Dependencies | Runtime | Extensions | Extension Points | Build | MANIFEST.MI

- ↶ Go to the **Overview** tab
- ↶ Make sure you the plugin ID is ch.actifsource.tutorial.builtin



- ↶ Switch back to **Extension** tab
- ↶ Click on **Add...**
- ↶ Select **ch.actifsourc.environment.RegisterBuiltin**
- ↶ Click on **Finish**

All Extensions

Define extensions for this plug-in in the following section.

- ▶ ch.actifsource.core.RegisterModel
- ▶ ch.actifsource.environment.RegisterBuiltin
- ▶ EXAMPLE_BUILTIN (Builtin) 1.

Extension Element Details

Set the properties of "Builtin". Required fields are denoted by "**".

name*:

defaultBuiltin:

isExtendable:

- ↩ Select the predefined (Builtin) entry
- ↩ Define a name `EXAMPLE_BUILTIN`
- ⓘ The other options are not used in this example:

defaultBuiltin

If true, the builtin will be added by default whenever a new actifsource project is created and must be explicitly removed.

isExtendable

If true, other plugins are allowed to define a builtin with the same name. This will result in merging the dependencies together. You may find this useful, if you provide extensions to your builtin, which you want to have automatically applied as soon as a plugin is installed.

All Extensions

Define extensions for this plug-in in the following section.

- ▶ ch.actifsource.core.RegisterModel
- ▶ ch.actifsource.environment.RegisterBuiltin
- 1. EXAMPLE_BUILTIN (Builtin)
- 2. ch.actifsource.tutorial.builtin (Plugin)

Extension Element Details

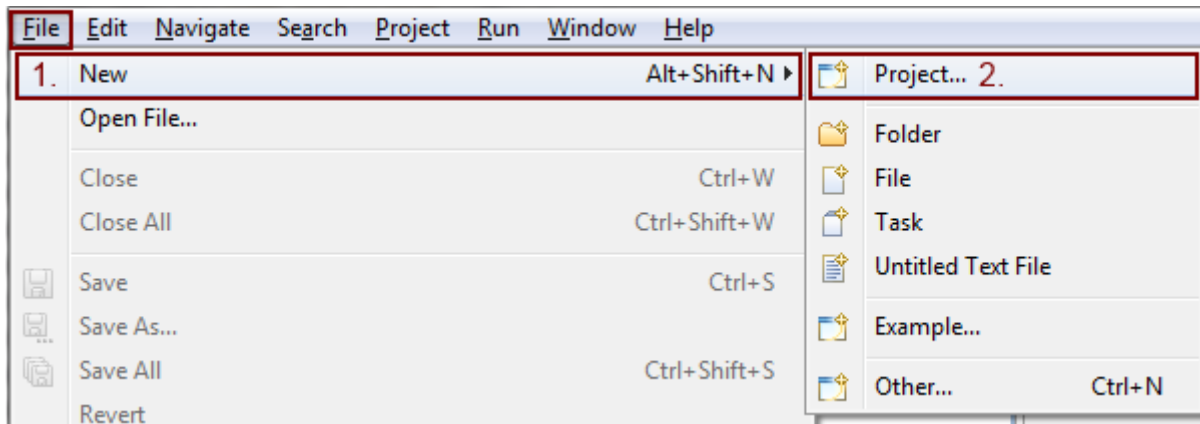
Set the properties of "Plugin". Required fields are denoted by "**".

pluginId*:

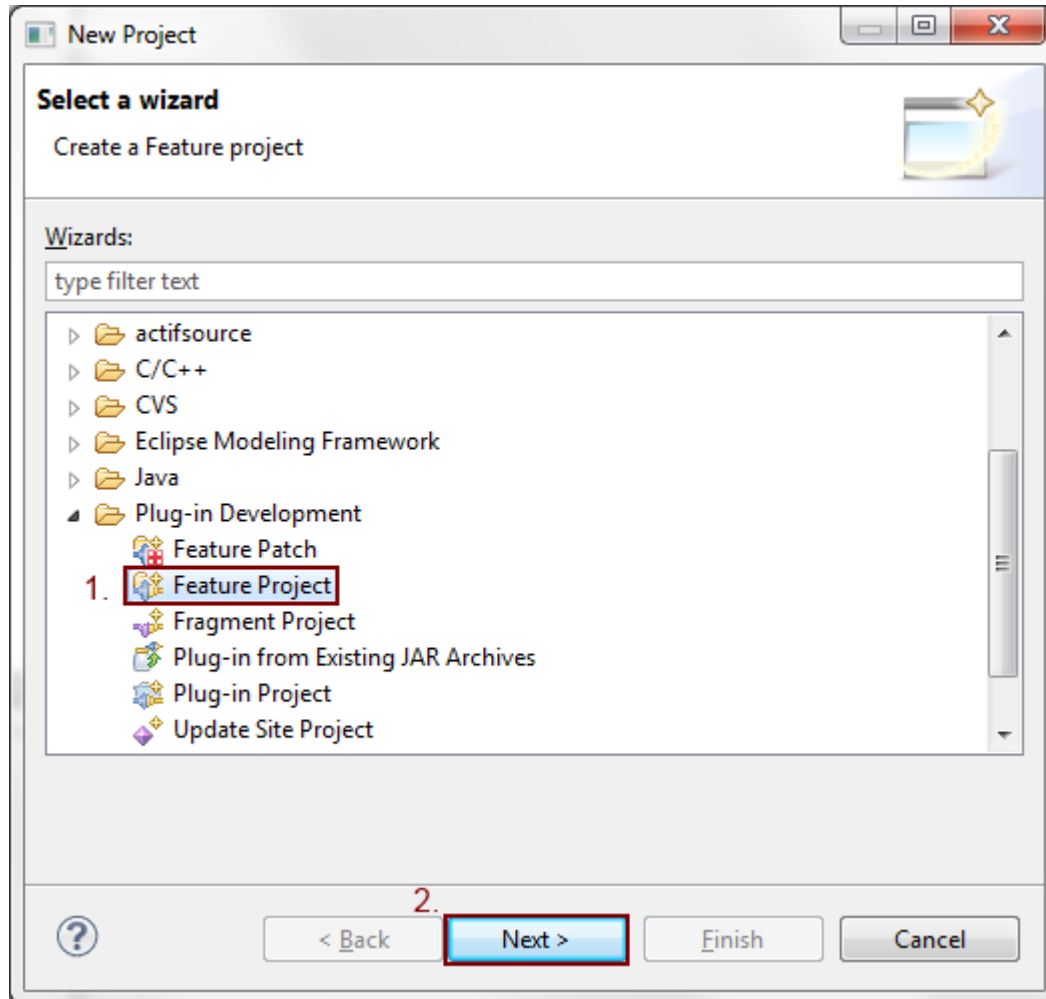
- ↩ Select the predefined (Plugin) entry
- ↩ Enter the **pluginId** of your plugin `ch.actifsource.tutorial.builtin`
- ⓘ This is not required to be the plugin defining extension point, this might be useful if you don't want to have a plugin dependency to **ch.actifsource.environment** in the plugin providing the resourcefolder.

Create a feature and setup an updatesite

- ① In order to deliver a plugin you need to create an eclipse feature containing the plugin and a updatesite where the feature is published



- ↵ Open the menu **File**
- ↵ Go into the submenu **New**
- ↵ Select **Project...**



↩ Select **Feature Project**

↩ Go to the **next** page.

New Feature

Feature Properties
Define properties that will be placed in the feature.xml file

Project name:

Use default location

Location:

Feature properties

Feature ID:

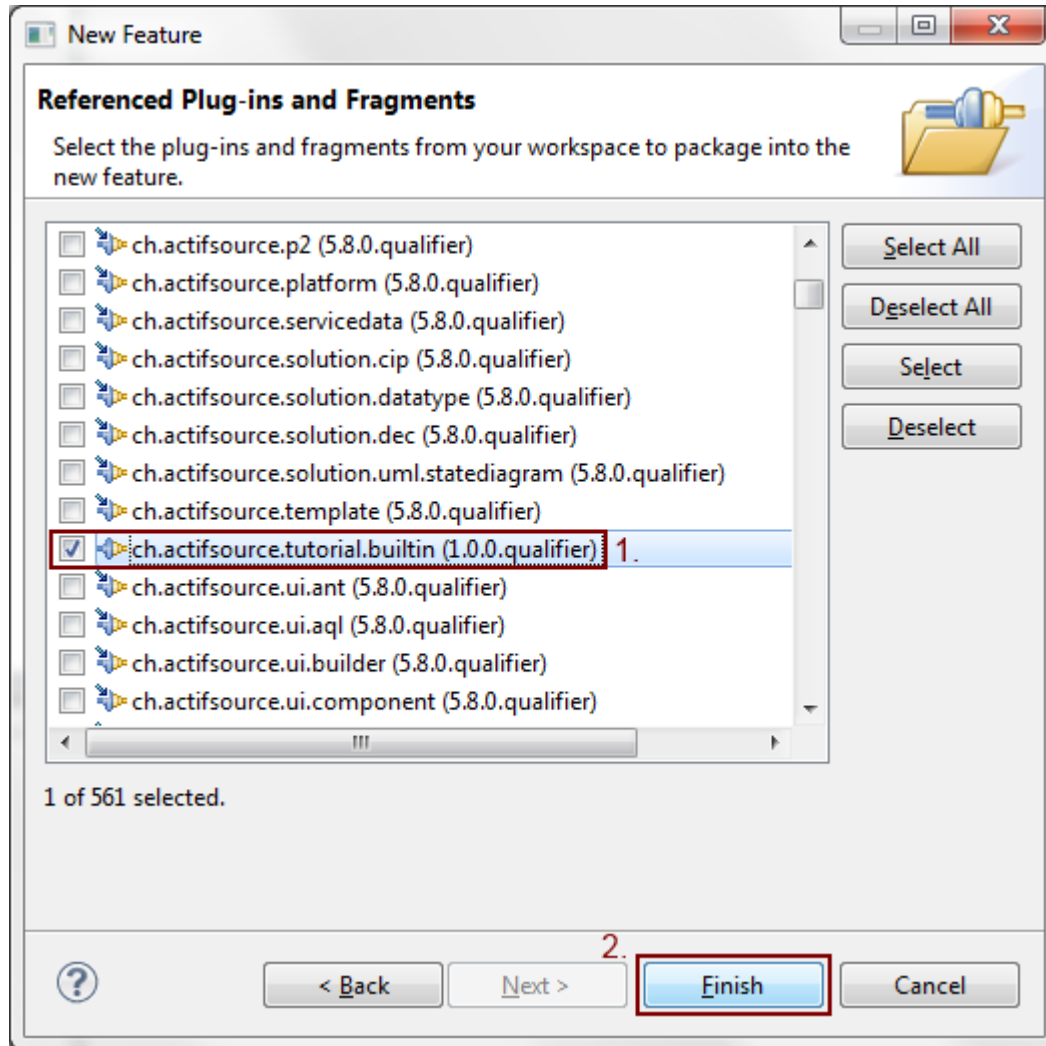
Feature Name:

Feature Version:

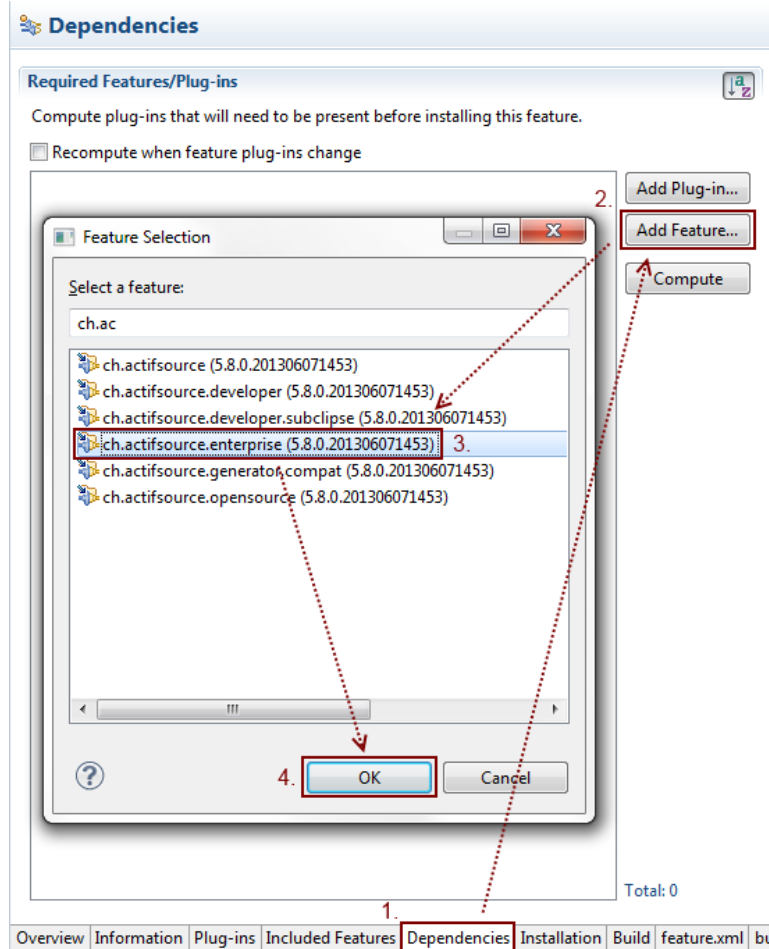
Feature Vendor:

Install Handler Library:

- ↩ Enter the project name `ch.actifsource.builtin.feature` for the feature
- ⓘ As you can see, the default **feature ID** is set to the project name.
- ↩ Press **next** to step further

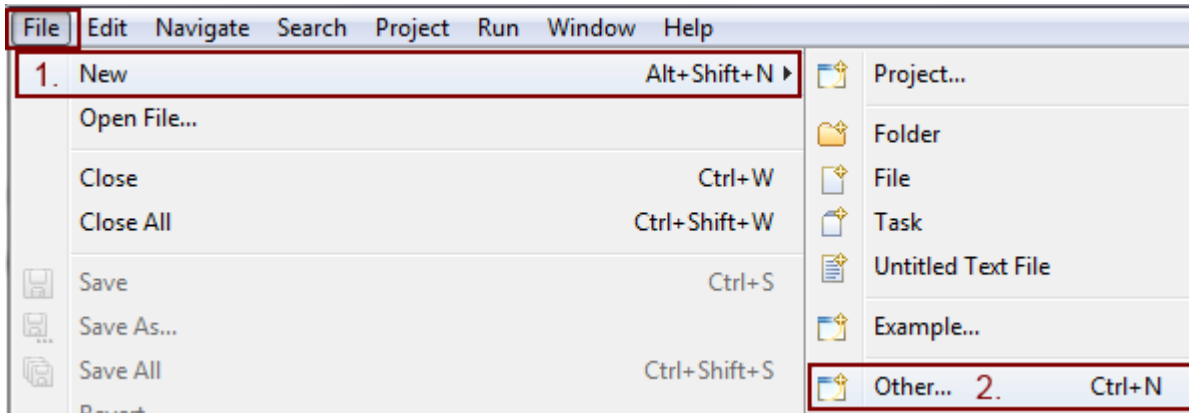


- ↵ Select the plugin ch.actifsource.tutorial.builtin
- ↵ Exit the wizard using **Finish**

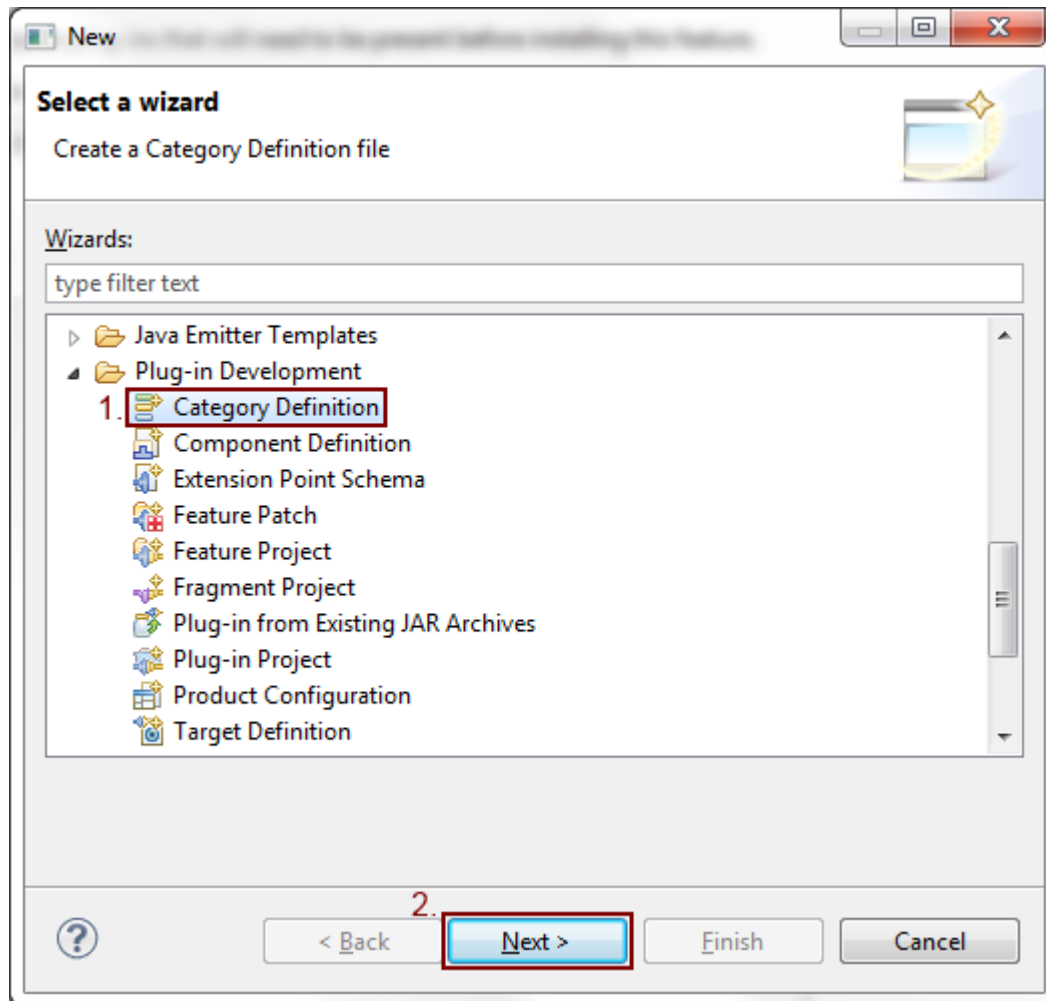


- ① After finishing the wizard of feature editor will open automatically.
- ↶ Switch to the **Dependencies** tab
- ↶ Press **Add Feature...**
- ↶ Select **ch.actifsource.enterprise**
- ↶ Hit **OK**
- ① This will add **actifsource enterprise** as dependency, disallowing to installing the feature without having actifsource installed. As a result eclipse will automatically select actifsource enterprise for installation if the location of actifsource is known and contacting other updatesites is enabled (see later)

- ① In order to make the feature visible on the updatesite, you need to create a category otherwise eclipse will hide the feature in the categorized view of the installation wizard.

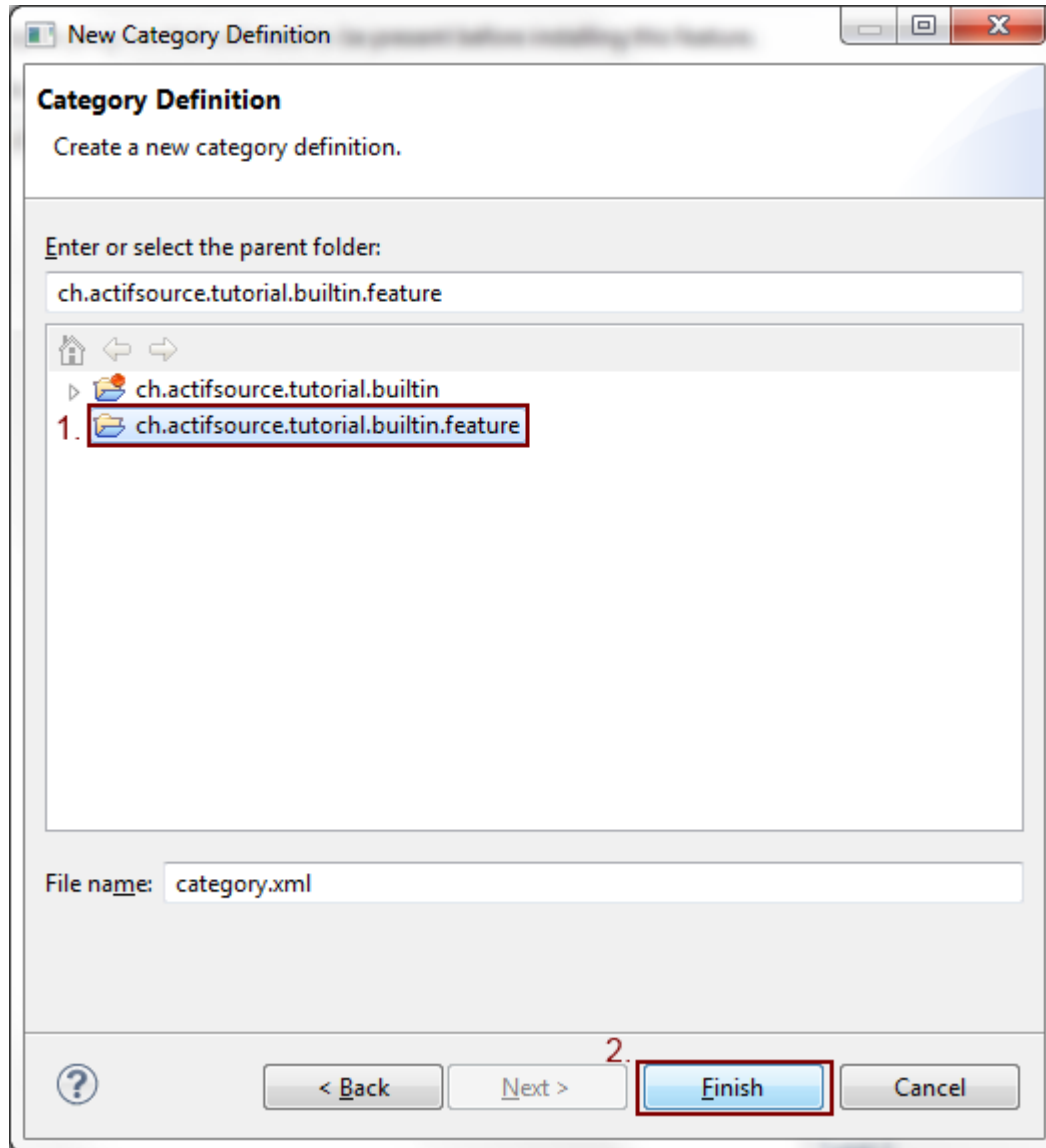


- ↖ Open the **File** menu
- ↖ Select **New**
- ↖ Click on **Other...**



↵ Select **Category Definition**

↵ Press the **Next** button



- ↖ Select your feature ch.actifsource.tutorial.builtin.feature
- ⓘ This is only the location of the file, this doesn't associate the feature with the category.
- ↖ Leave the filename unchanged and press **Finish**

Managing the Categories

1. Add the categories to be published into the repository.
2. For easier browsing of the repository, add the features to the desired categories.

1.

Category Properties

Provide a unique id, a name and a description for each category.
"*" denotes a required field.

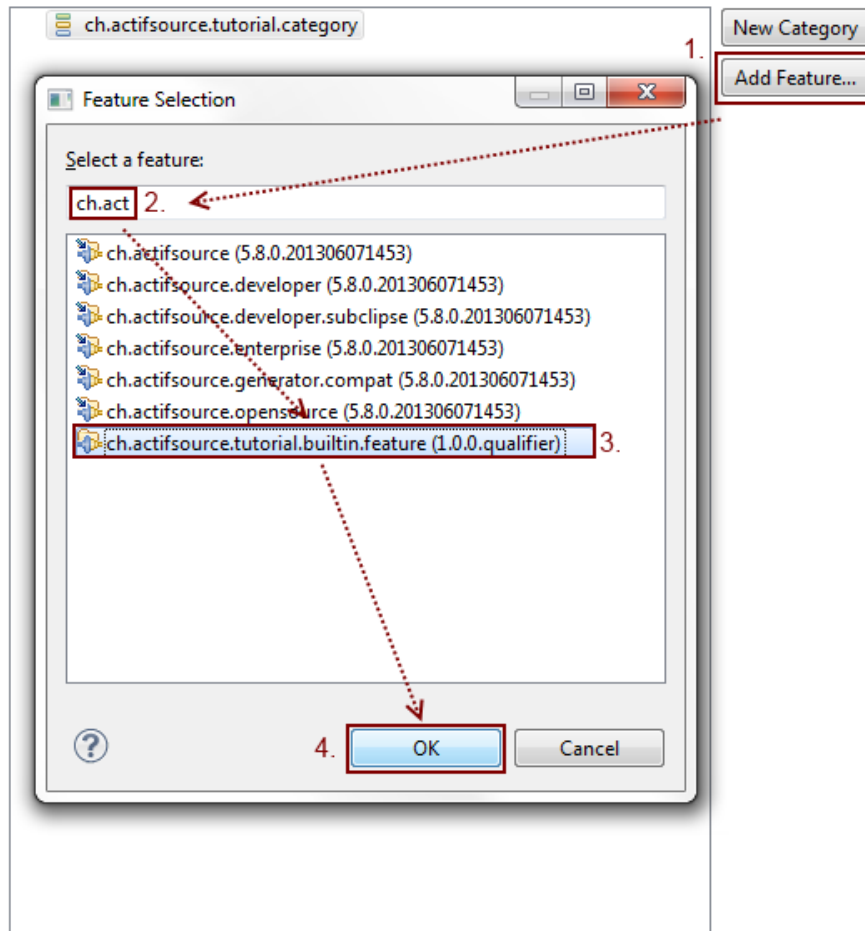
2. ID*:
 Name*:
Description:

- ↩ Press **New Category**
- ↩ Enter the id `ch.actifsource.tutorial.category`
- ↩ Enter the name `Tutorial Category`

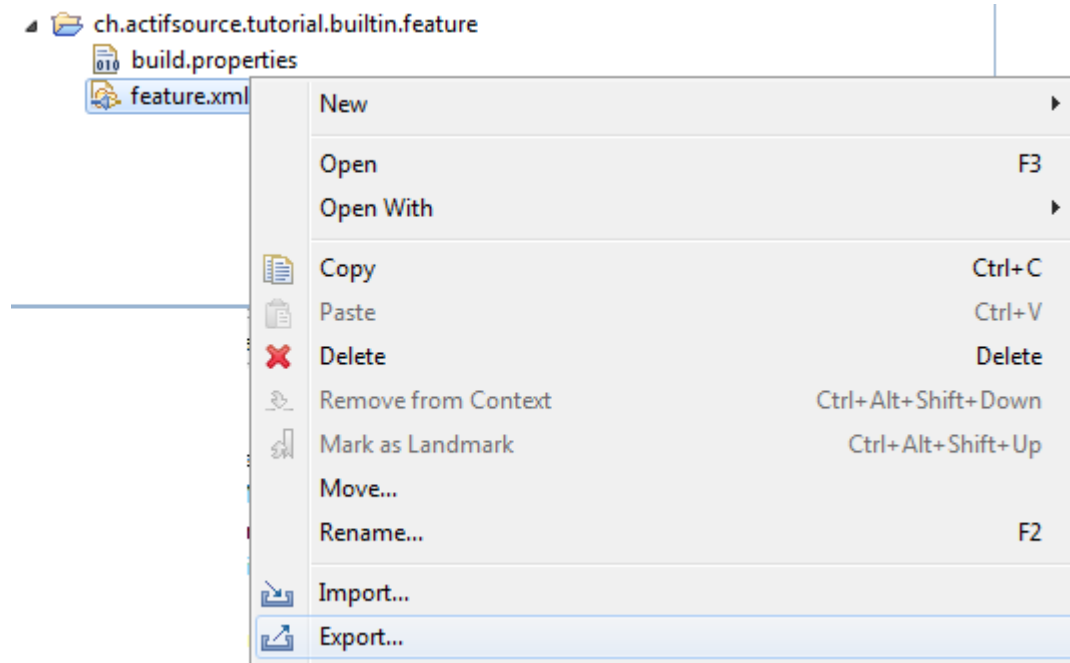
Category Definition

Managing the Categories

1. Add the categories to be published into the repository.
2. For easier browsing of the repository, add the features to the desired categories.

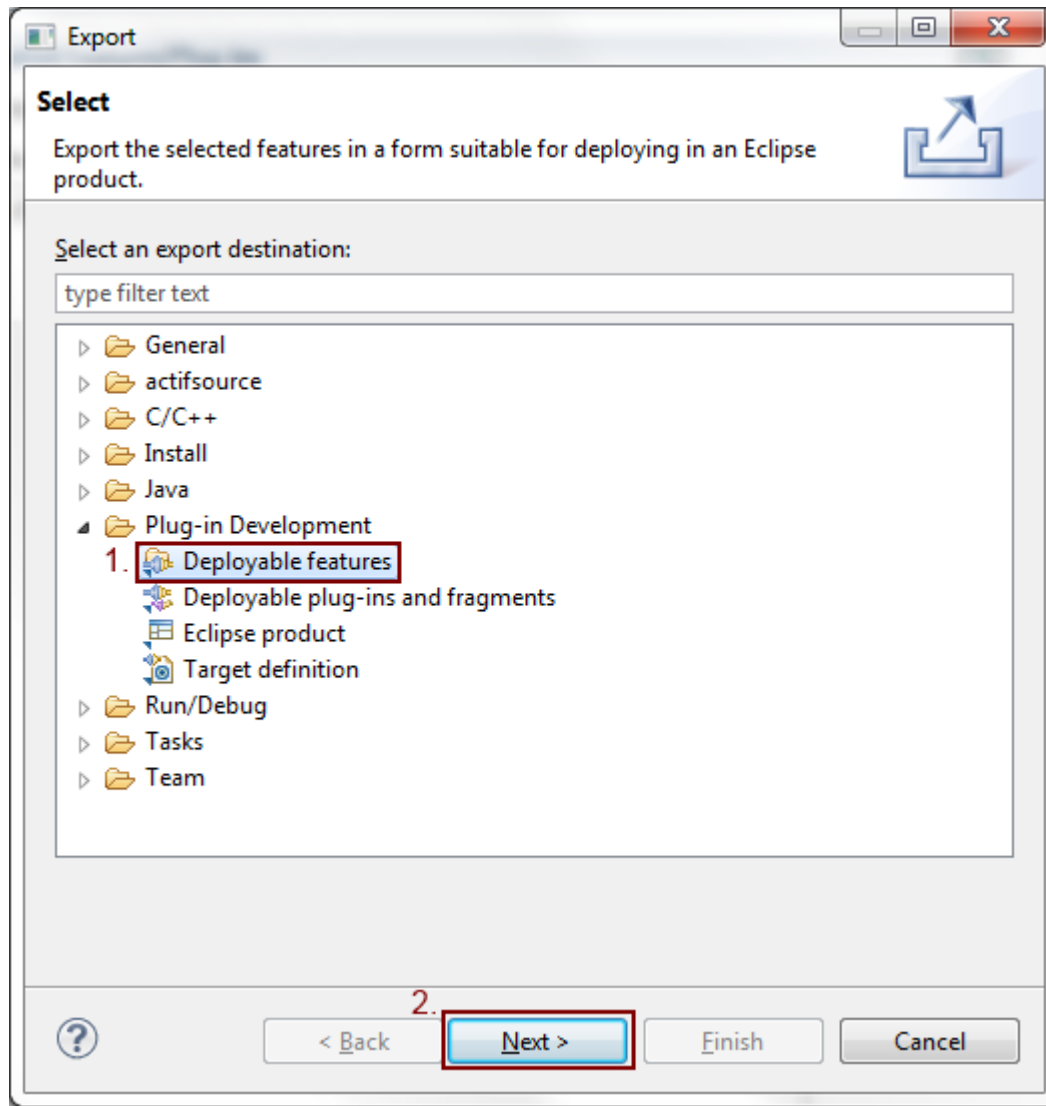


- ↖ Select the category
- ↖ Press **Add Feature...**
- ↖ Select the your feature ch.actifsource.tutorial.builtin.feature
- ⓘ At this point we are ready to export the feature to an updatesite



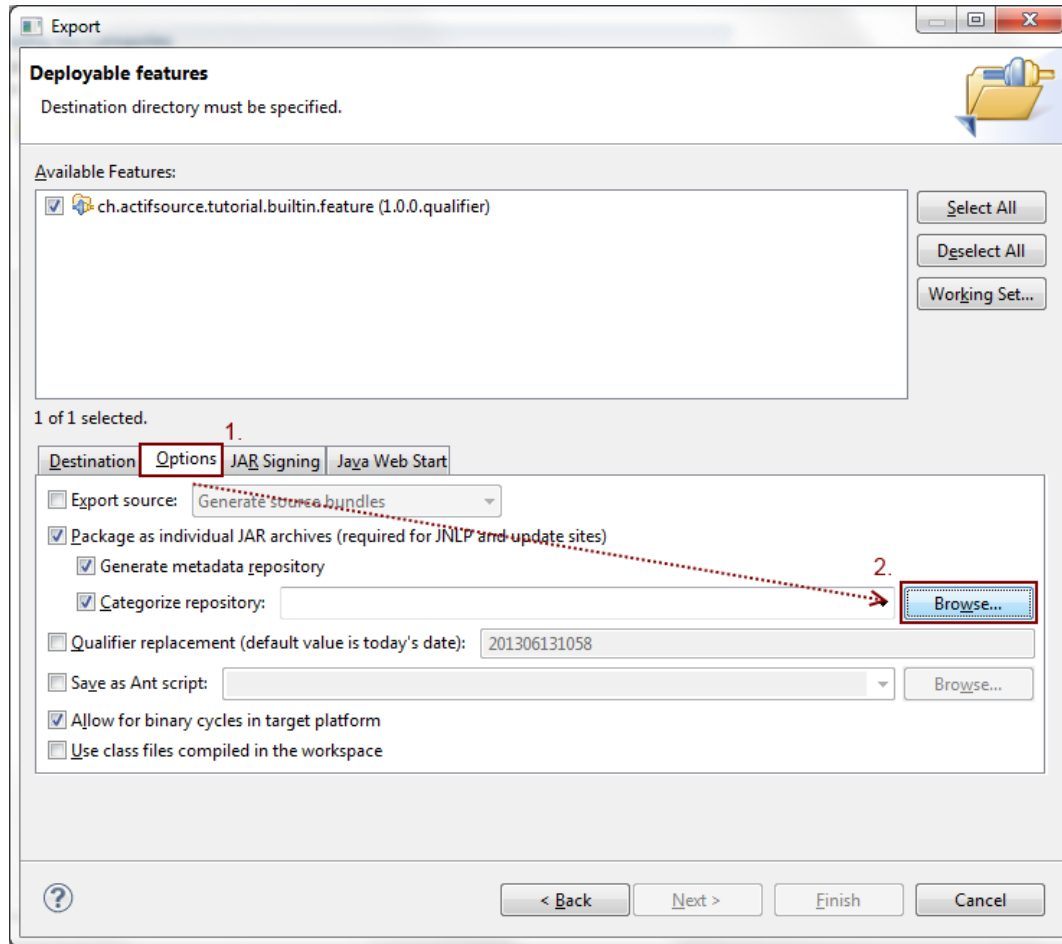
↵ Right click on the feature.xml

↵ Select **Export...**

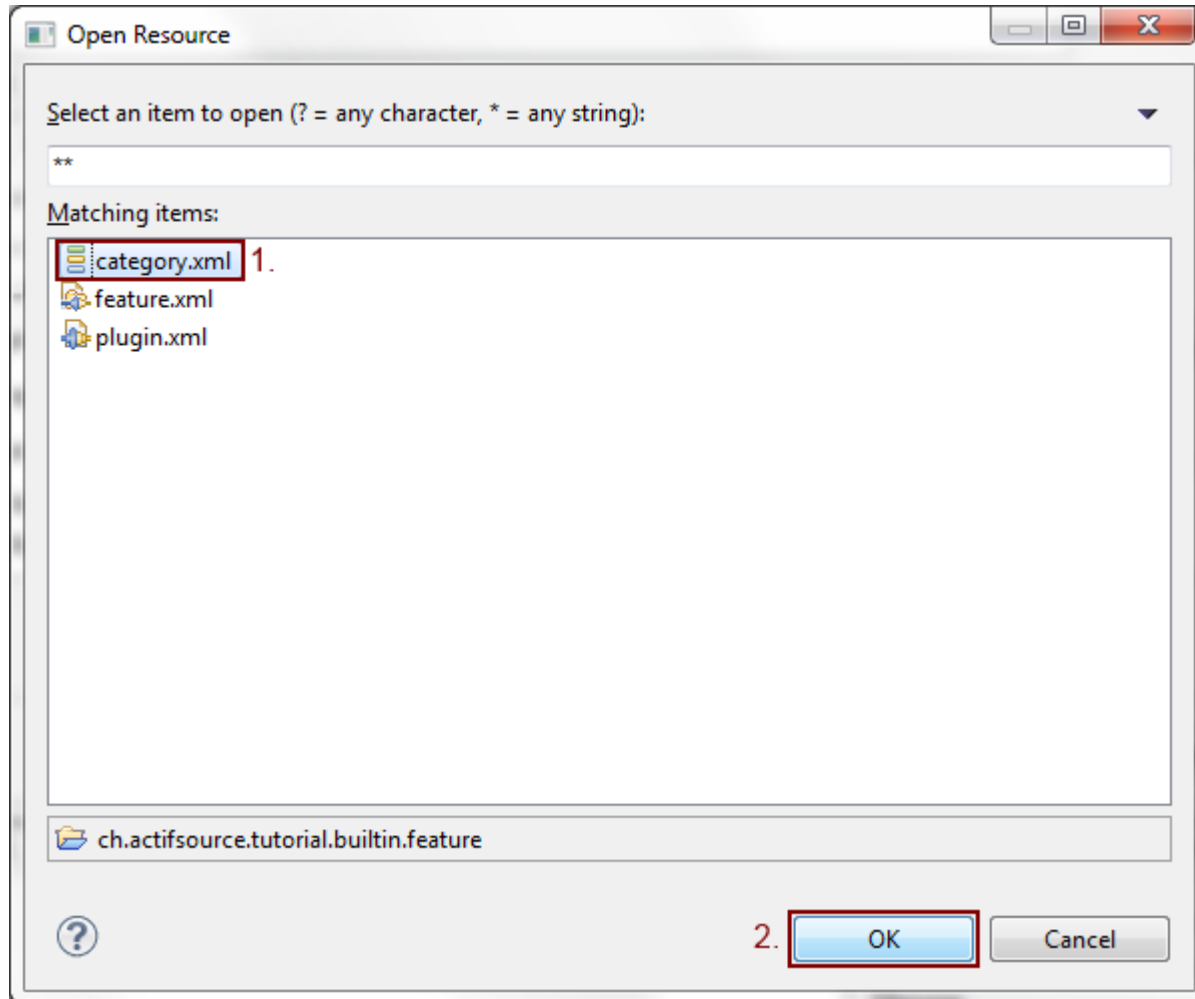


↖ Select **Deployable features**

↖ Click on **Next**

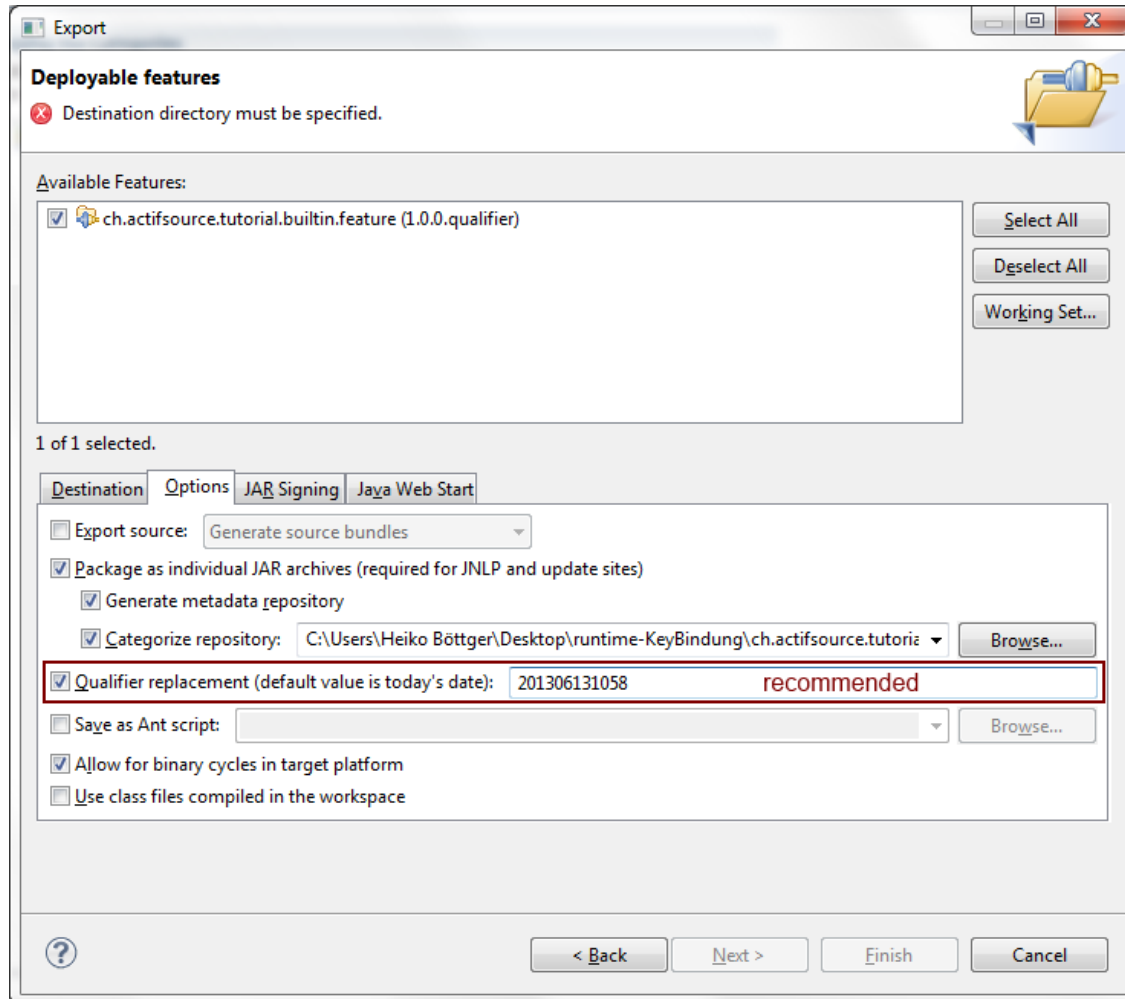


- ↖ Select **Options**
- ↖ Click on **Browse...**

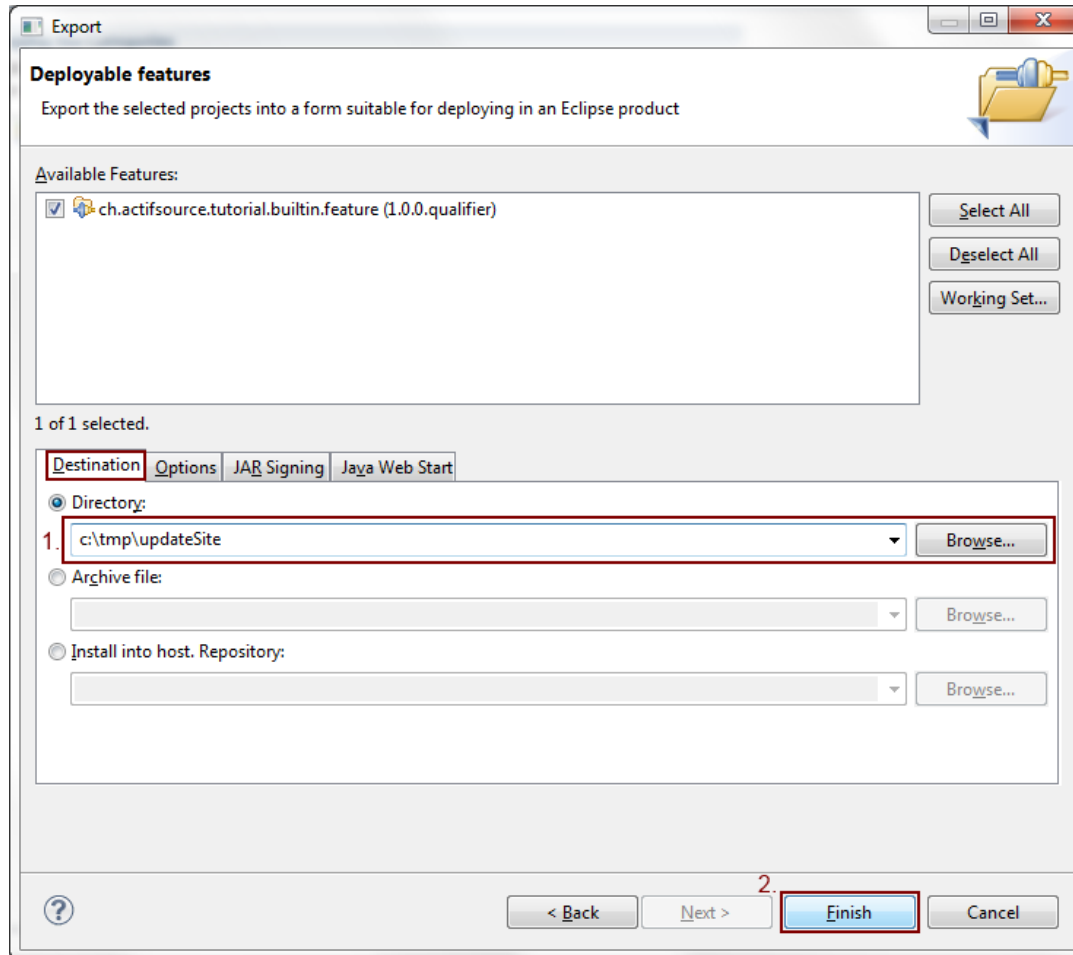


↵ Select category.xml

↵ Close with **OK**



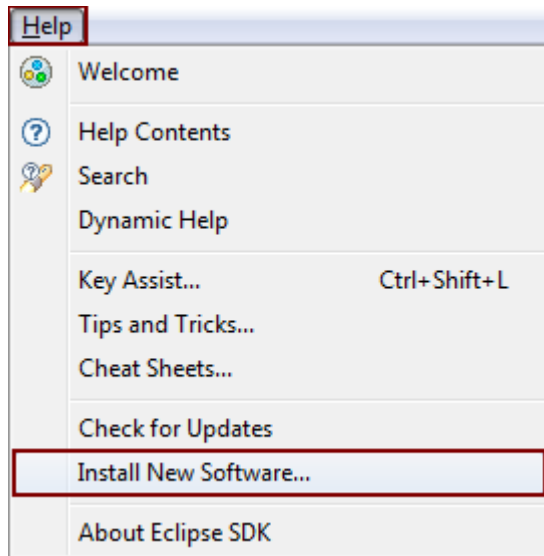
- ① If you don't want to increase the micro version each time you export, it is recommended to set a qualifier. Failing to change the version or qualifier will cause eclipse to not overwrite an existing on the update site.
- ↶ Check the **Qualifier replacement** checkbox
- ↶ Enter the current date and time in reverse order
- ① Be careful if the major, minor and micro version of two versions are the same, a text comparison between the qualifiers is done. This means write the date the other way around will cause eclipse to select the wrong latest version.



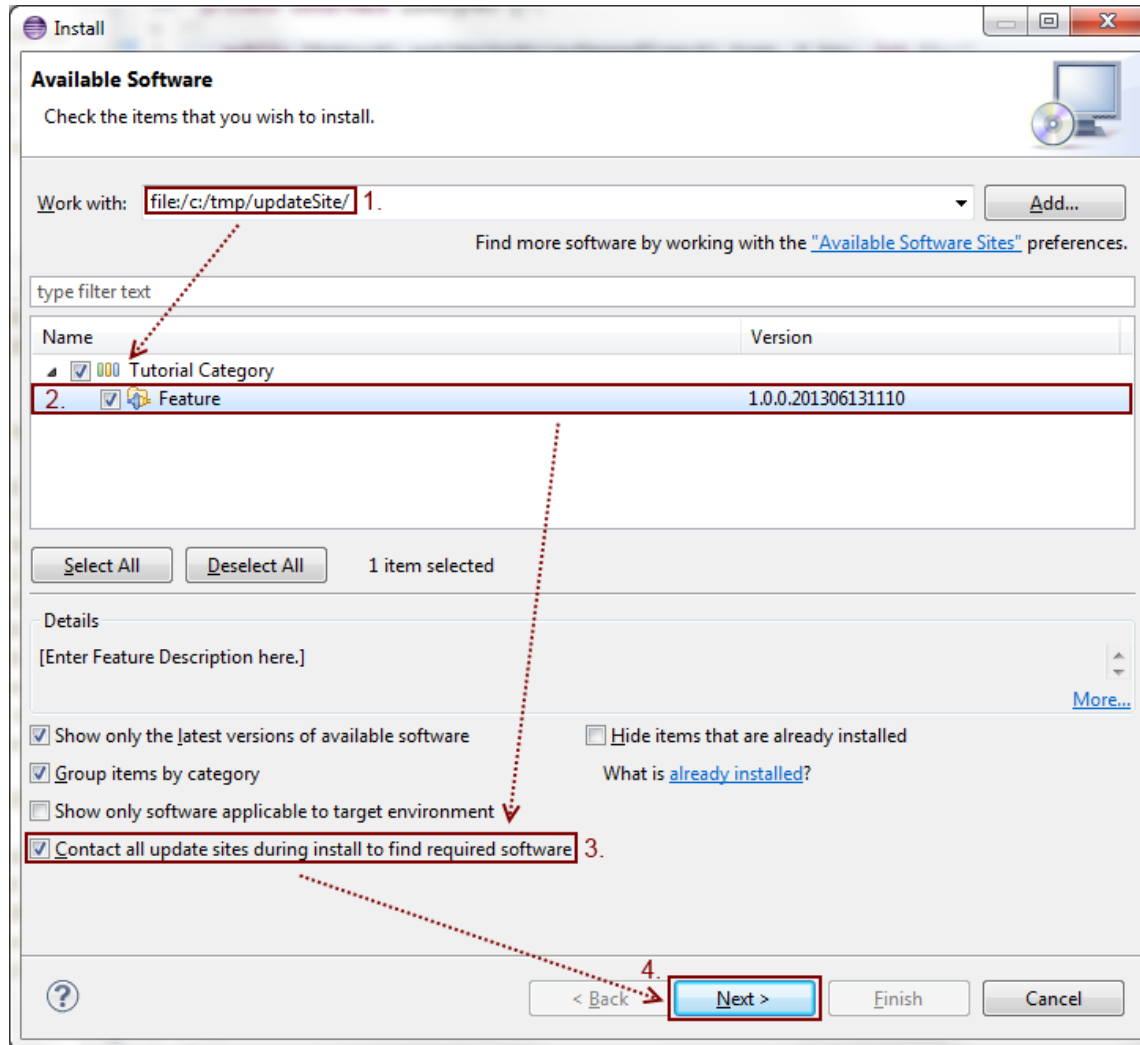
- ↩ Go back to the **Destination** tab
- ↩ Enter the location of the updatesite, we use `c:\tmp\updateSite`
- ⓘ Depending on your installation you may have to choose a different directory, where you have write access.
- ↩ Press **Finish**

Install the example feature

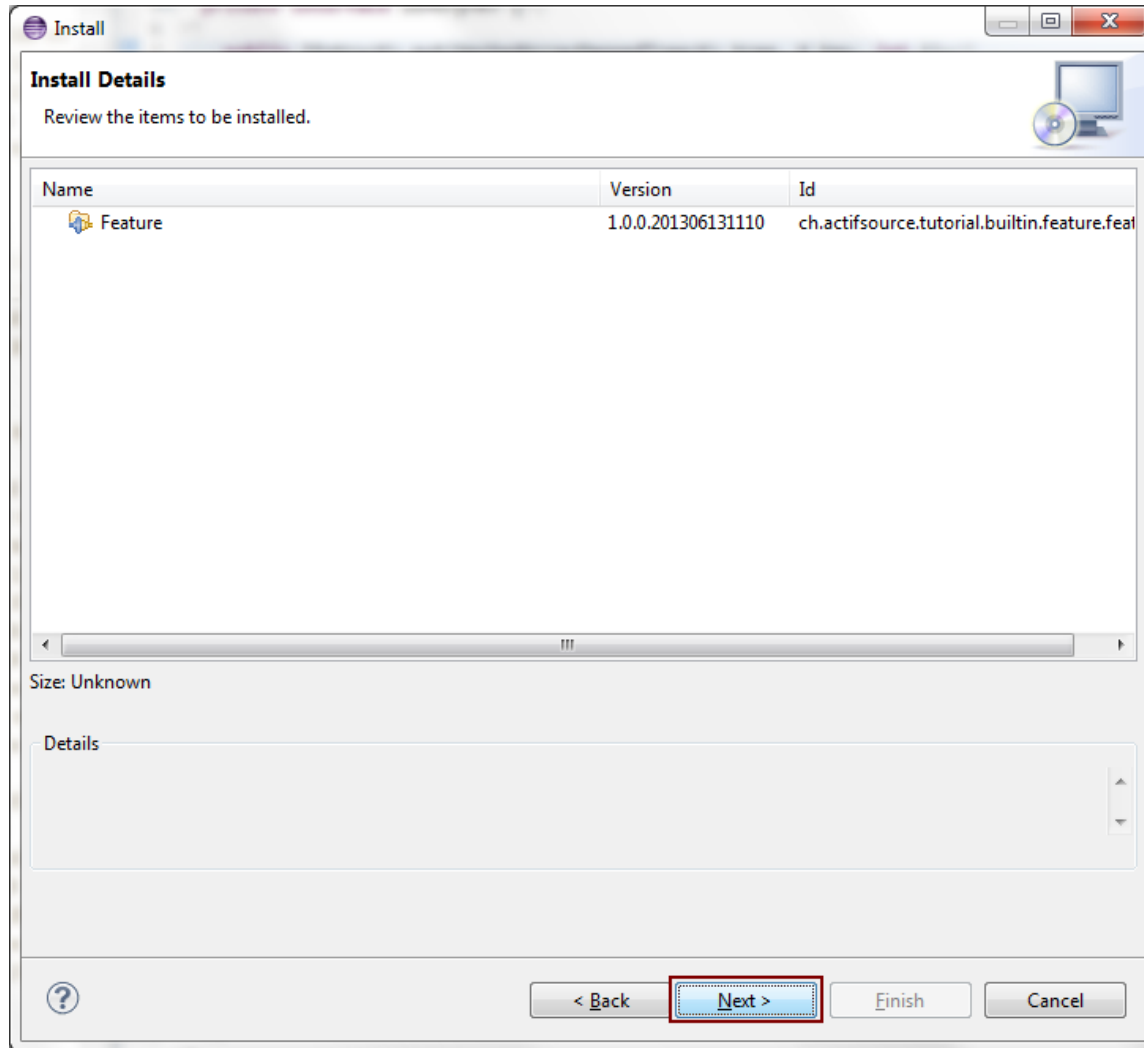
- ① For this step you need to have installation rights on the running eclipse. To get around this, you may download a new eclipse from www.eclipse.org and test it on a writable location.



- ↵ Go **Help**
- ↵ Select **Install New Software....**

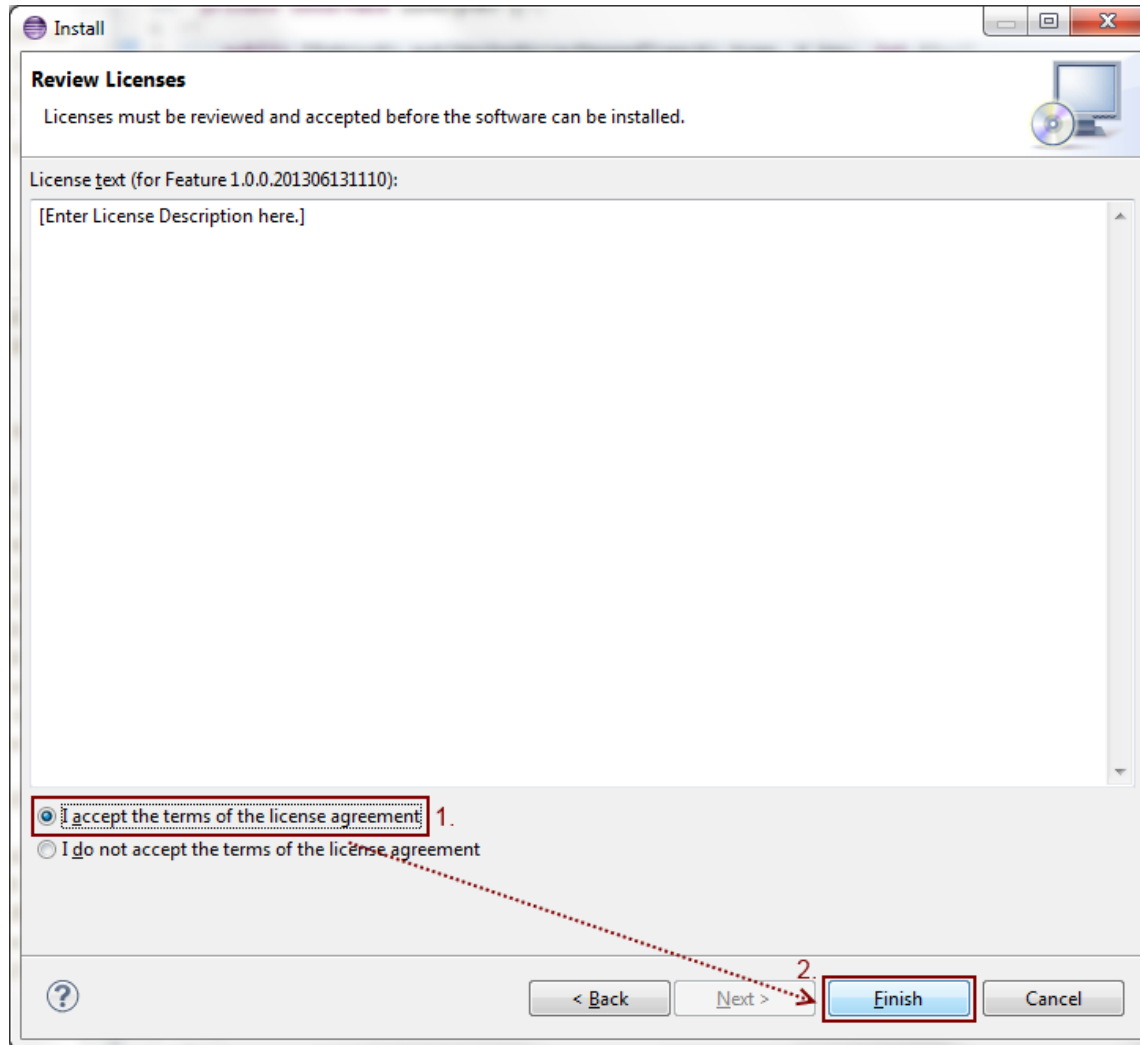


- ↪ Enter the location of the updatesite, we used `file:/c:/tmp/updateSite`
- ① Eclipse requires the use of the file-URL. In order to distribute the updateSite on the web, just upload the content of the directory to your webserver.
- ↪ Check **Contact all update sites during install to find required software**
- ① This option uses the **required feature** defined in the feature where we added the `ch.actifsource.enterprise`.
- ↪ Press **Next**

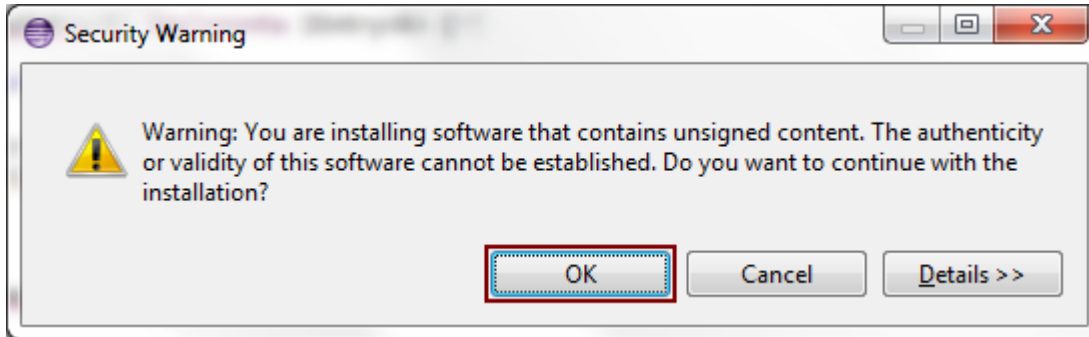


- ① This dialog will show addition features if eclipse found missing feature required for installation. However you will see an error message if any feature is missing that wasn't found. This happens when eclipse doesn't know an updatesite containing the feature. Another reason for installation failures are conflicts between installed versions, you may need to uninstall conflicting plugins first or try to update all at once using the **Check For Update** dialog

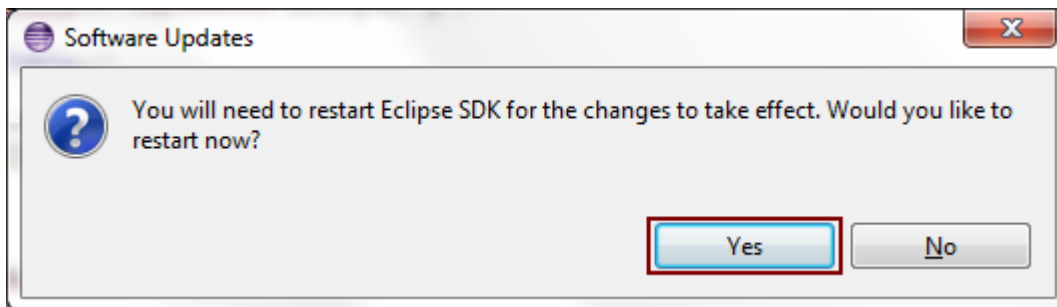
↩ Press **Next**



- ↵ Read the license(s)
- ↵ Accept the license
- ↵ Press **Finish**
- ⓘ Eclipse will start downloading the required features and plugins from the updatesites



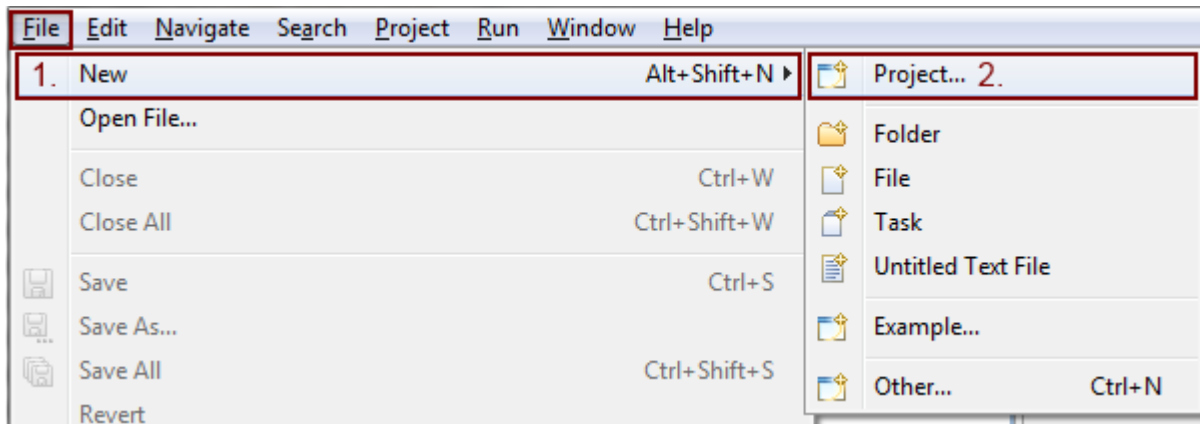
- ① Before starting the installation of a feature, eclipse will check if the feature and the plugins are signed. This option can be activate in the export wizard, however this goes beyond this tutorial.
- ↪ Since we know from the verification dialog, that only our feature is going to be installed, safely click **OK**



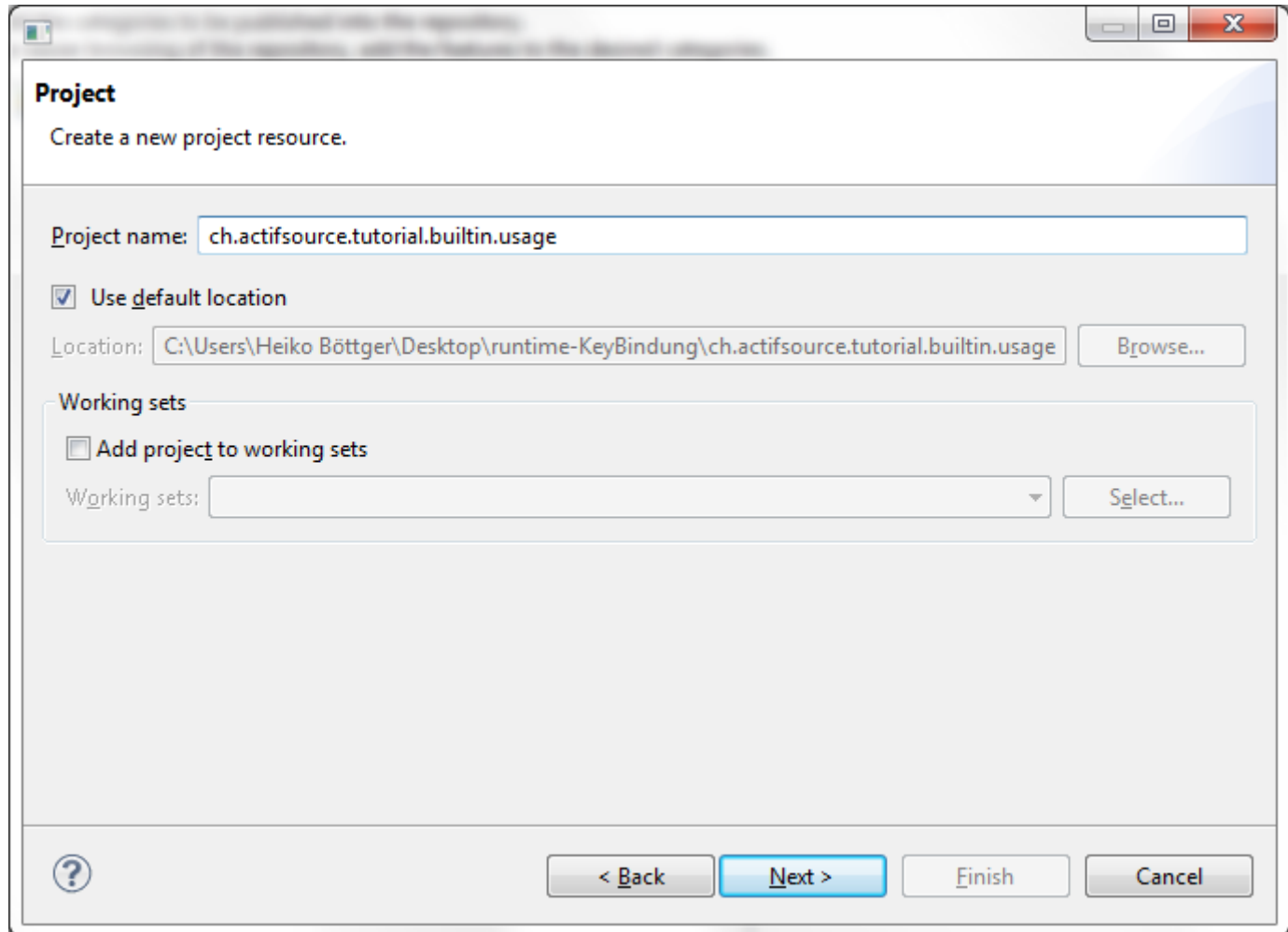
- ↪ Press **Yes** to restart

Using the builtin

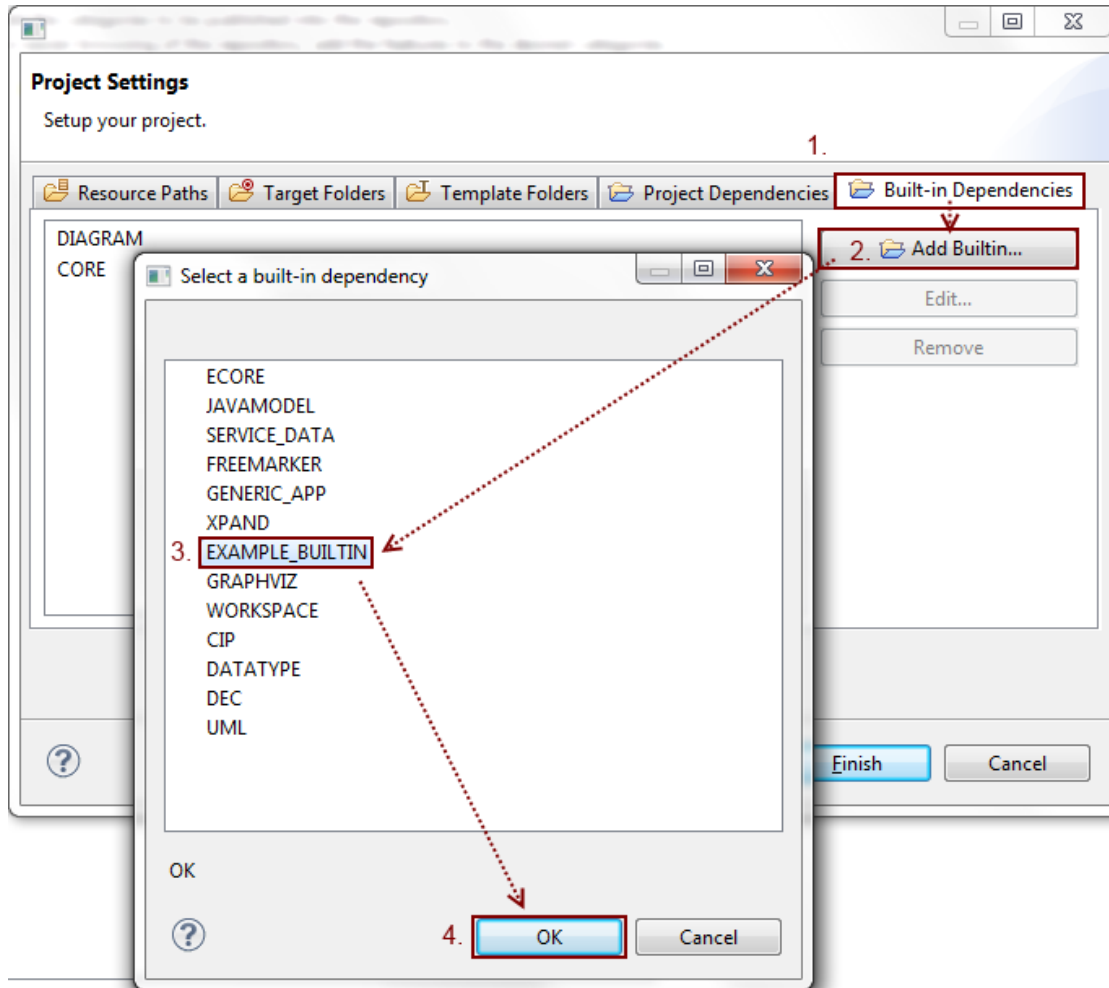
- ① Now we can use the builtin. You have two options, creating a plugin dependency or adding the builtin in the actifsource preferences



- ↩ Select **File**
- ↩ Go to **New**
- ↩ Click on **Project...**



- ↩ Enter a project name for the usage, we use `ch.actifsource.tutorial.builtin.usage`
- ↩ Switch to the **Next** page



- ↗ Activate the **Built-in Dependencies** tab
- ↗ Click on **Add Builtin...**
- ↗ Select EXAMPLE_BUILTIN
- ↗ Hit **OK**
- ↗ **Finish** the wizard and you are ready to go
- ⓘ As mentioned when defining the builtin this step can be omitted if you have defined builtin with the **defaultBuiltin**-attribute set to **true**.

