Eclipse UOMo Tutorial

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME
1.0	2013-06-17		C

Contents

This is a quick start guide for users of Eclipse UOMo.

Chapter 1

Eclipse Plugin Tutorial

1.1 Introduction

Eclipse UOMo is an API for working with physical quantities and units. We are going to create a demonstration application that provides an API to work with Newton's Second Law of Motion:

> The acceleration of a body is directly proportional to, and in the same direction as, the net force acting on the body, and inversely proportional to its mass.

Otherwise written as:

`F =M×A`

Where:

- `F` is Force in Newtons
- `M` is Mass in Kilograms
- `A` is Acceleration in Meters per second

Our example API will provide methods for computing any of the above, given the other two.

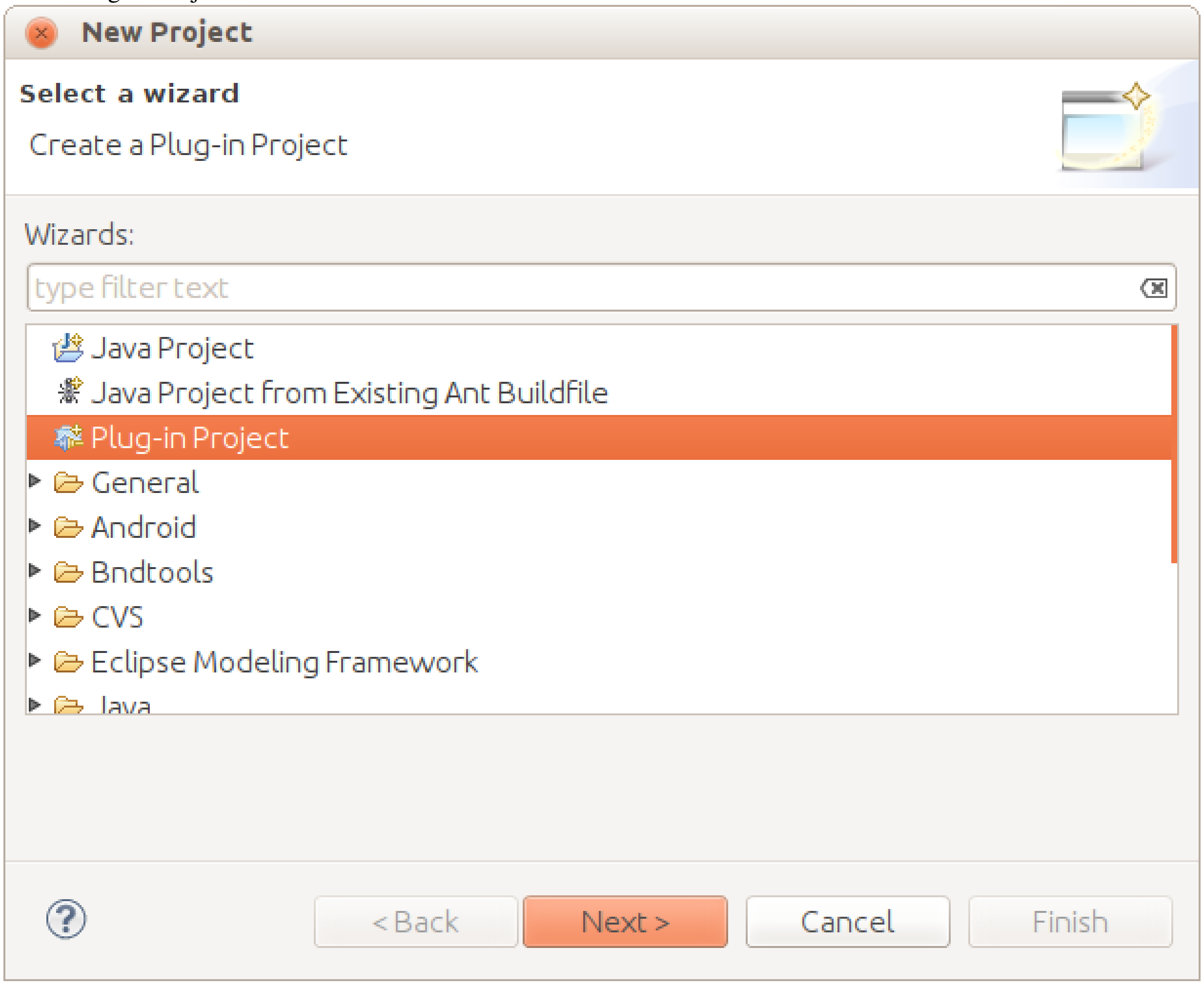
Example code for the following is available at GitHub:

https://github.com/duckAsteroid/uomo-example

1.2 New Eclipse Plugin Project

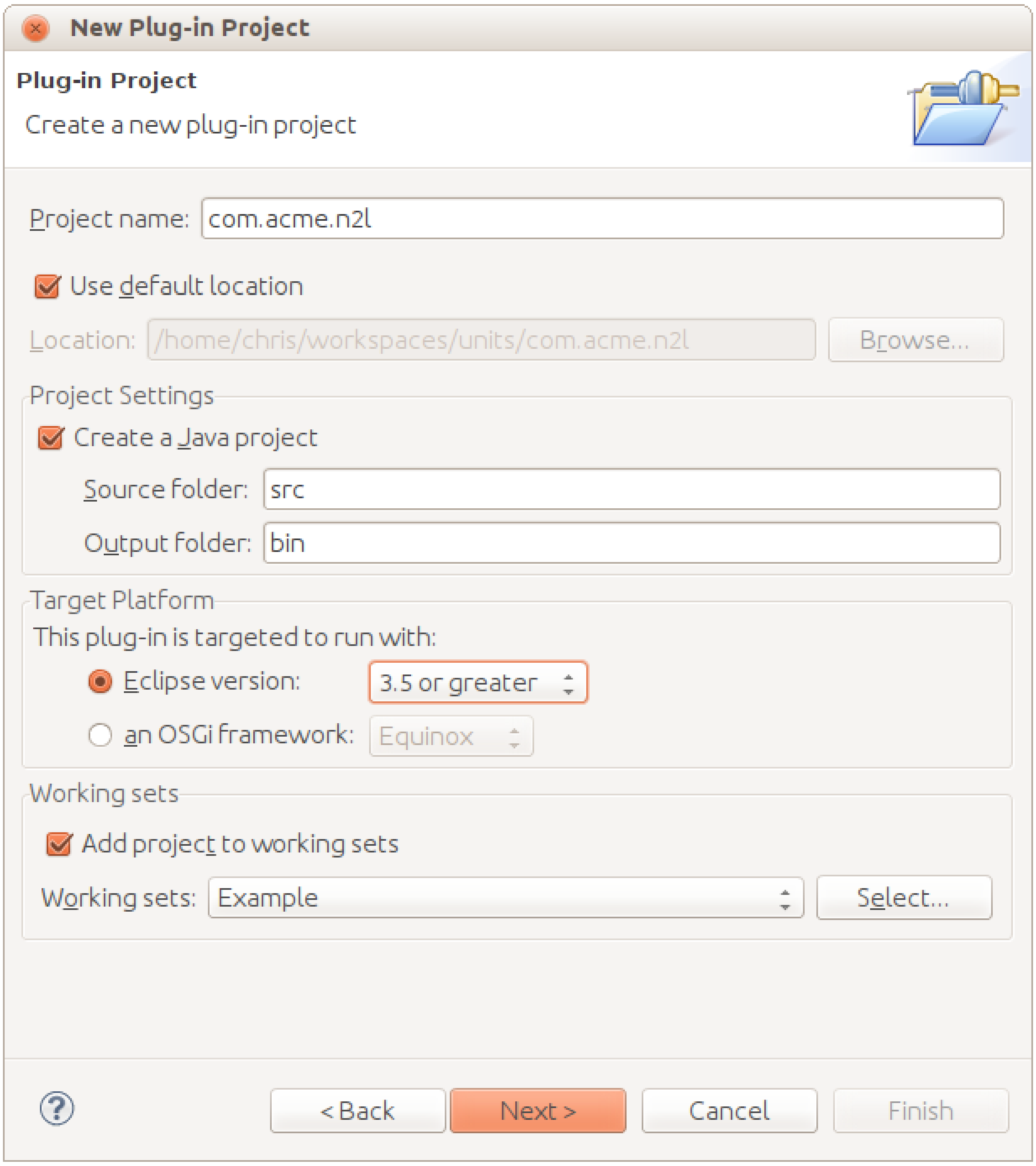
Most of verbosity in the steps that follows are related to complexities of creating Eclipse plug-ins rather than anything hard about using UOMo. . .

1. In Eclipse click the `File` → `New` → `Project...` menu
2. Select Plug-in Project:



   Click “Next”

3. Enter a name for the plugin project (e.g. `com.acme.n21`):


New Plug-in Project

Plug-in Project

Create a new plug-in project

Project name: com.acme.n2l

[✓] Use default location

Location: /home/chris/workspaces/units/com.acme.n2l

Browse...

Project Settings

[✓] Create a Java project

Source folder: src

Output folder: bin

Target Platform

This plug-in is targeted to run with:

[✓ Eclipse version:] 3.5 or greater

an OSGi framework: Equinox

Working sets

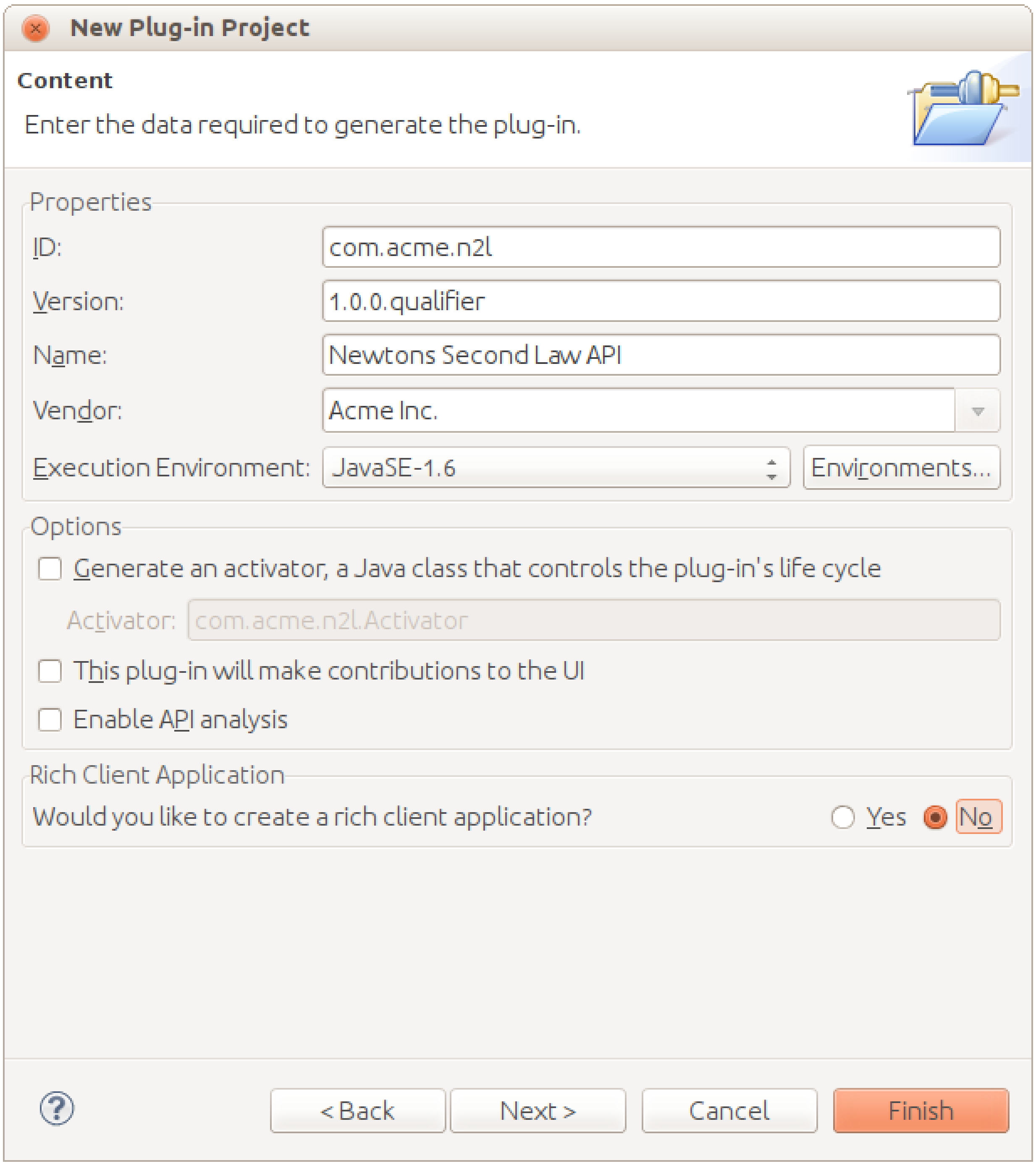
[✓] Add project to working sets

Working sets: Example

Select...

Click “Next”

4. You can now customise the plugin details as follows:



Properties

ID:	com.acme.n2l
Version:	1.0.0.qualifier
Name:	Newtons Second Law API
Vendor:	Acme Inc.
Execution Environment:	JavaSE-1.6

Options

- [ ] Generate an activator, a Java class that controls the plug-in's life cycle
  - Activator: com.acme.n2l.Activator
- [ ] This plug-in will make contributions to the UI
- [ ] Enable API analysis

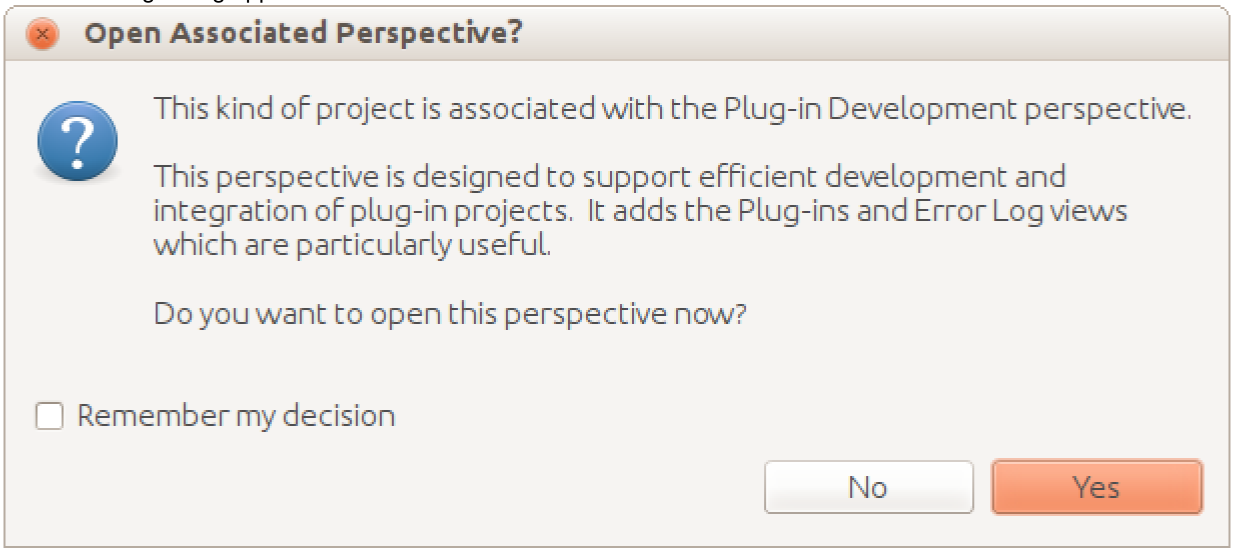
Rich Client Application

Would you like to create a rich client application? Yes [✓ No]

Ensure that all checkboxes under “Options” are de-selected.
Then click “Finish”.

5. Eclipse will create your new project.

Note
If the following dialog appears:



Open Associated Perspective?

This kind of project is associated with the Plug-in Development perspective.

This perspective is designed to support efficient development and integration of plug-in projects. It adds the Plug-ins and Error Log views which are particularly useful.

Do you want to open this perspective now?

☐ Remember my decision

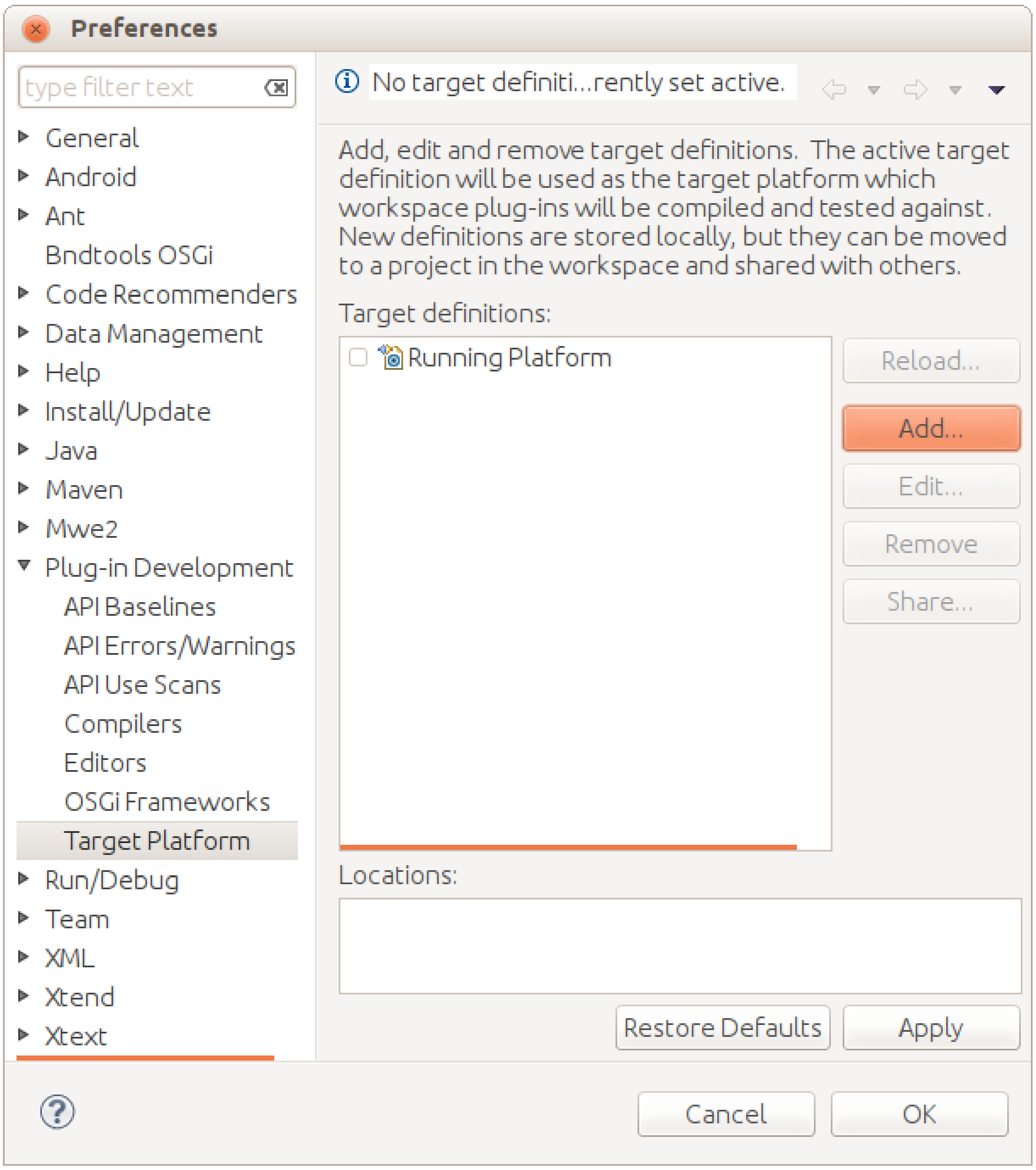
click “Yes” to open the PDE perspective.

1.3 Eclipse Build Platform Configuration

Now we need to configure Eclipse so that it knows about UOMo and where it lives so it can build against it.

Note
The default behavour of Eclipse when building plug-ins is to build against "itself"; that is to use the currently running plug-ins of the IDE as the build target.

1. Click the menu “Window → Preferences”
2. Navigate to the “Plug-in Development → Target Platform” section of the preferences dialog:



No target definiti...rently set active.

Add, edit and remove target definitions. The active target definition will be used as the target platform which workspace plug-ins will be compiled and tested against. New definitions are stored locally, but they can be moved to a project in the workspace and shared with others.

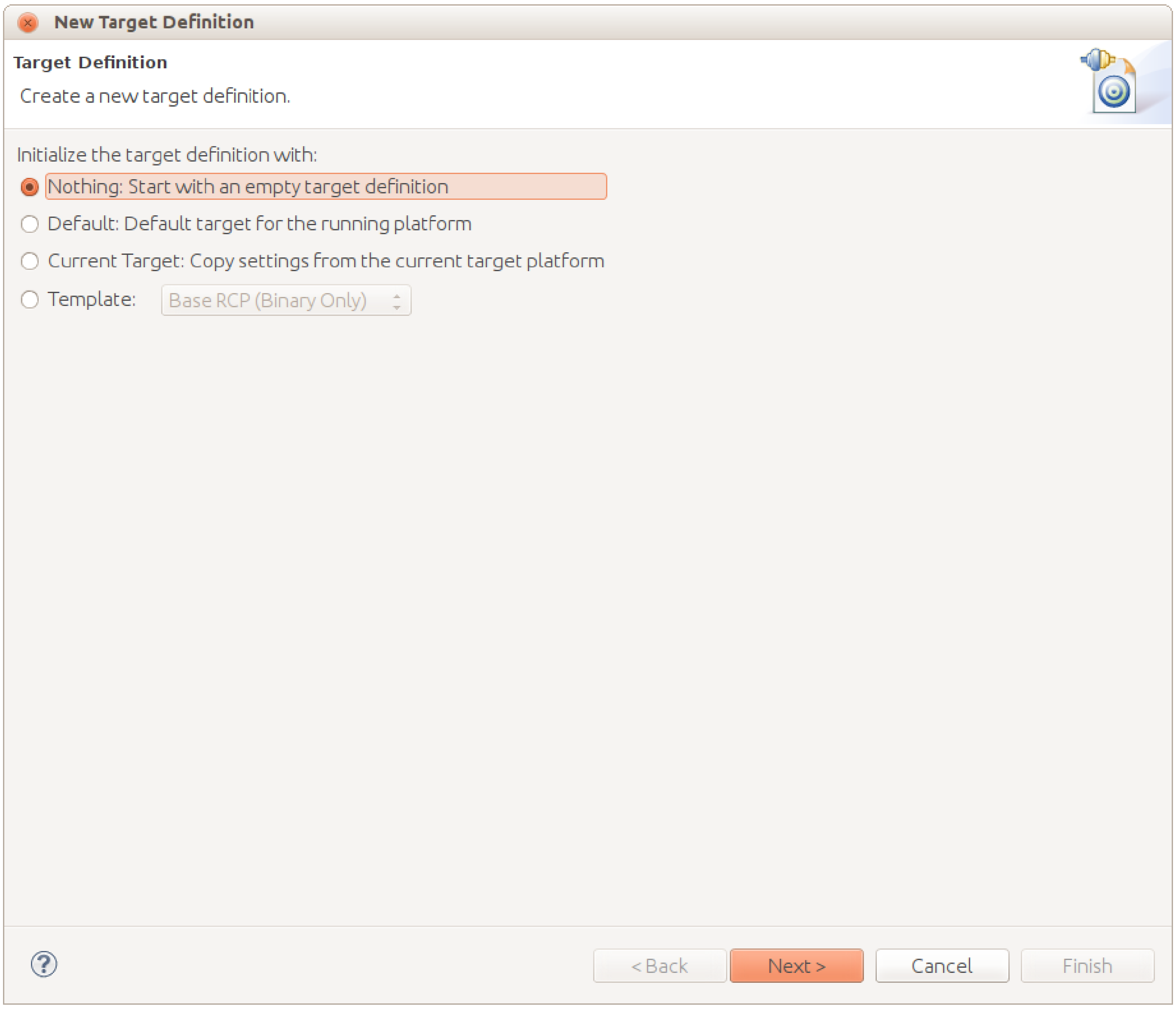
Target definitions:

Running Platform

Locations:

Click “Add…”

3. Select “Nothing: Start with an empty target definition” in the “New target definition” dialog:



Target Definition

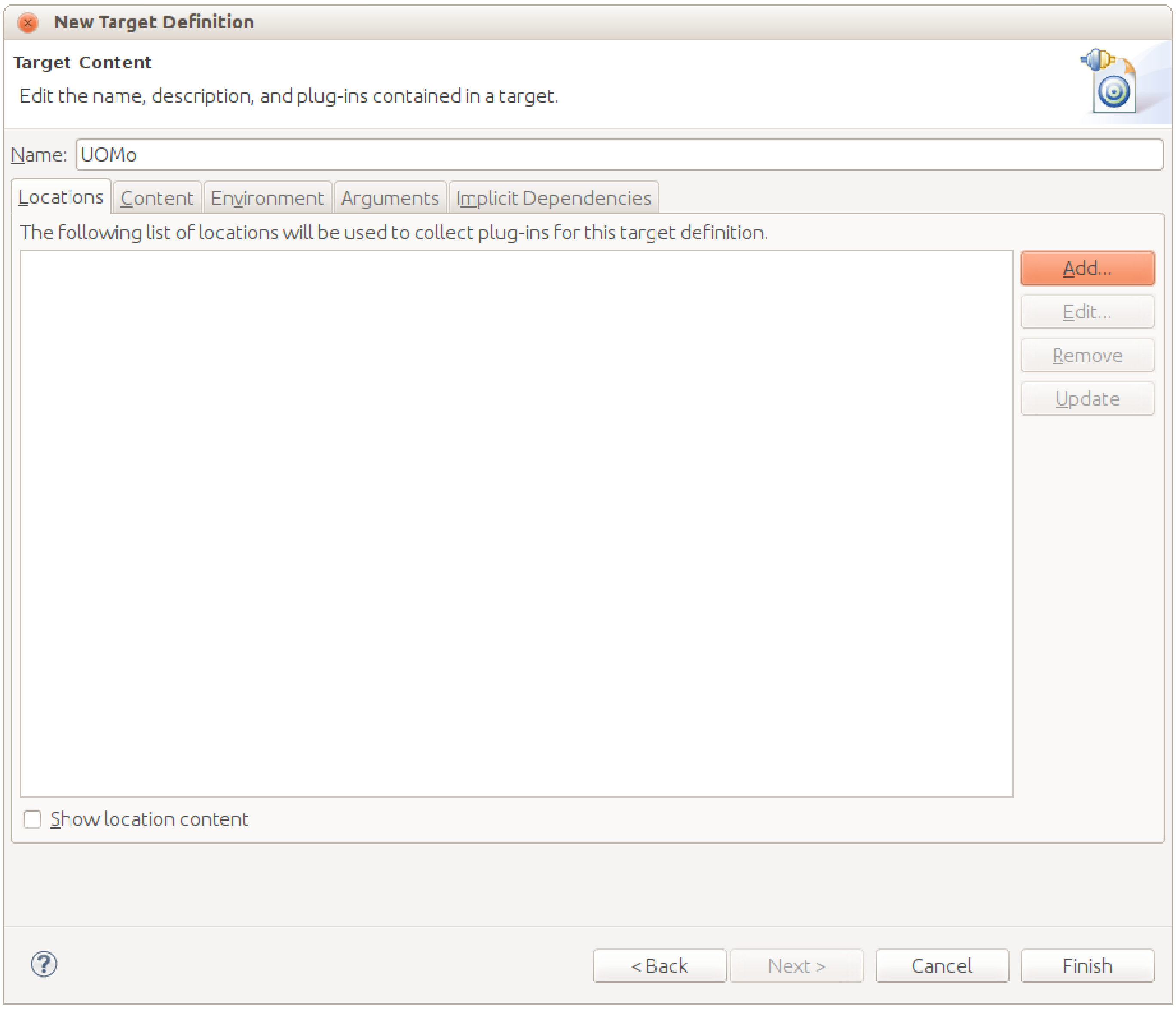
Create a new target definition.

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)

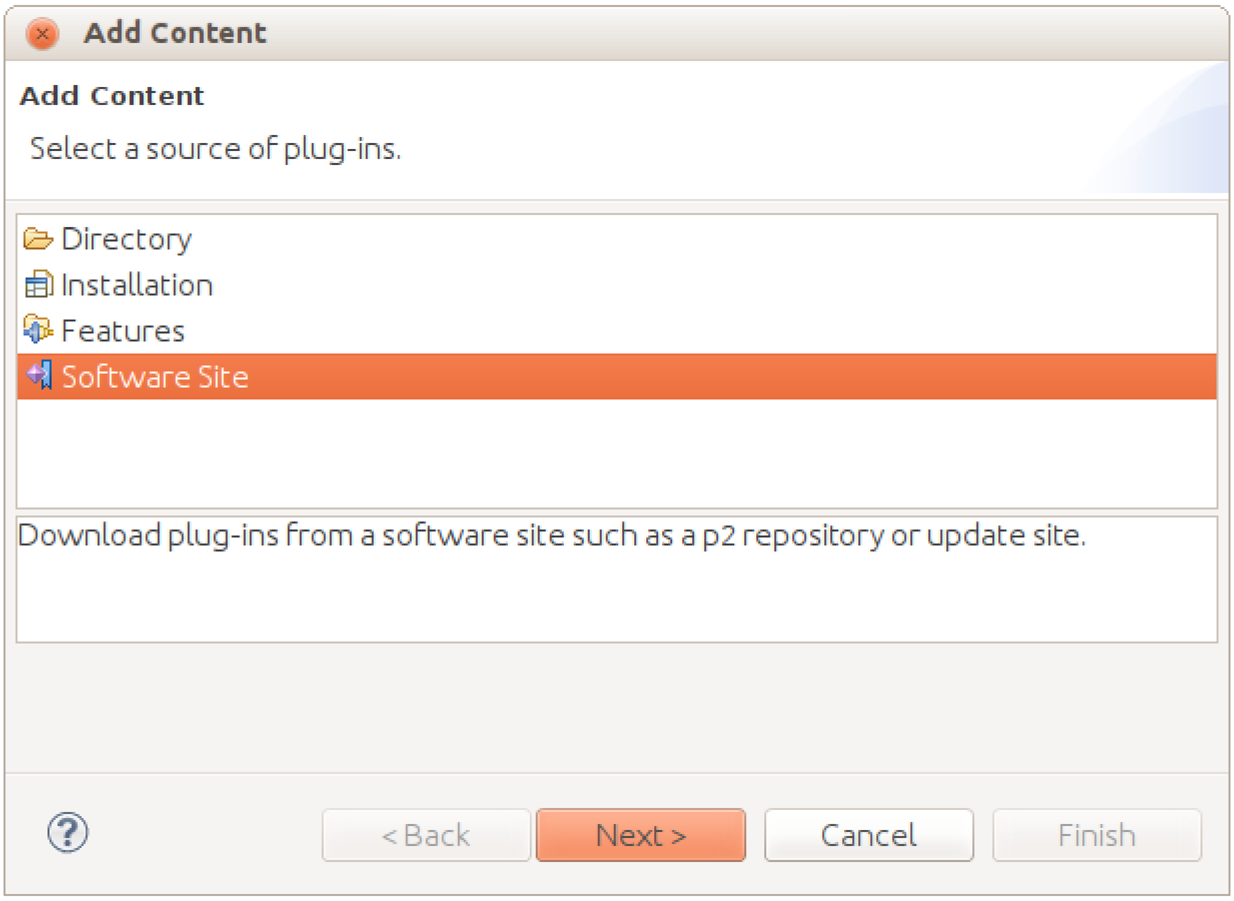
Click "Next"

4. Give the new target a name, such as "UOMo":



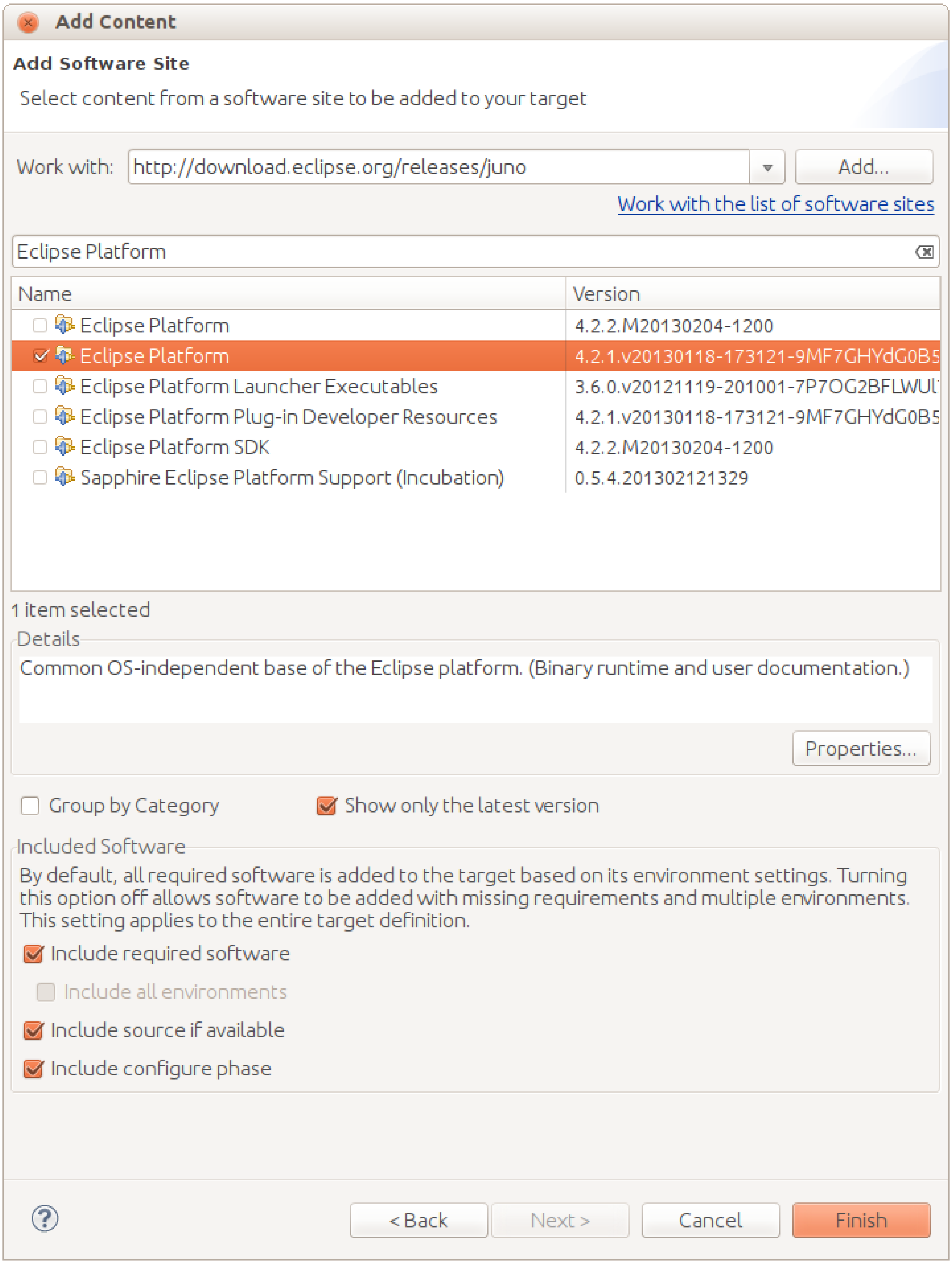
Click “Add. . . ”

5. Select “Software Site” from the plug-in source options:


Add Content
Add Content
Select a source of plug-ins.
Directory
Installation
Features
Software Site
Download plug-ins from a software site such as a p2 repository or update site.
< Back
Next >
Cancel
Finish

Click “Next”

6. You should now be in the "Add Content" dialog. From here we can add plugins to our target platform:.



Add Content

Add Software Site

Select content from a software site to be added to your target

Work with: http://download.eclipse.org/releases/juno

Work with the list of software sites

Eclipse Platform

Name	Version
☐ Eclipse Platform	4.2.2.M20130204-1200
✓ Eclipse Platform	4.2.1.v20130118-173121-9MF7GHYdG0B5
☐ Eclipse Platform Launcher Executables	3.6.0.v20121119-201001-7P7OG2BFLWUl
☐ Eclipse Platform Plug-in Developer Resources	4.2.1.v20130118-173121-9MF7GHYdG0B5
☐ Eclipse Platform SDK	4.2.2.M20130204-1200
☐ Sapphire Eclipse Platform Support (Incubation)	0.5.4.201302121329

1 item selected

Details

Common OS-independent base of the Eclipse platform. (Binary runtime and user documentation.)

☐ Group by Category ✓ Show only the latest version

Included Software

By default, all required software is added to the target based on its environment settings. Turning this option off allows software to be added with missing requirements and multiple environments. This setting applies to the entire target definition.

✓ Include required software

☐ Include all environments

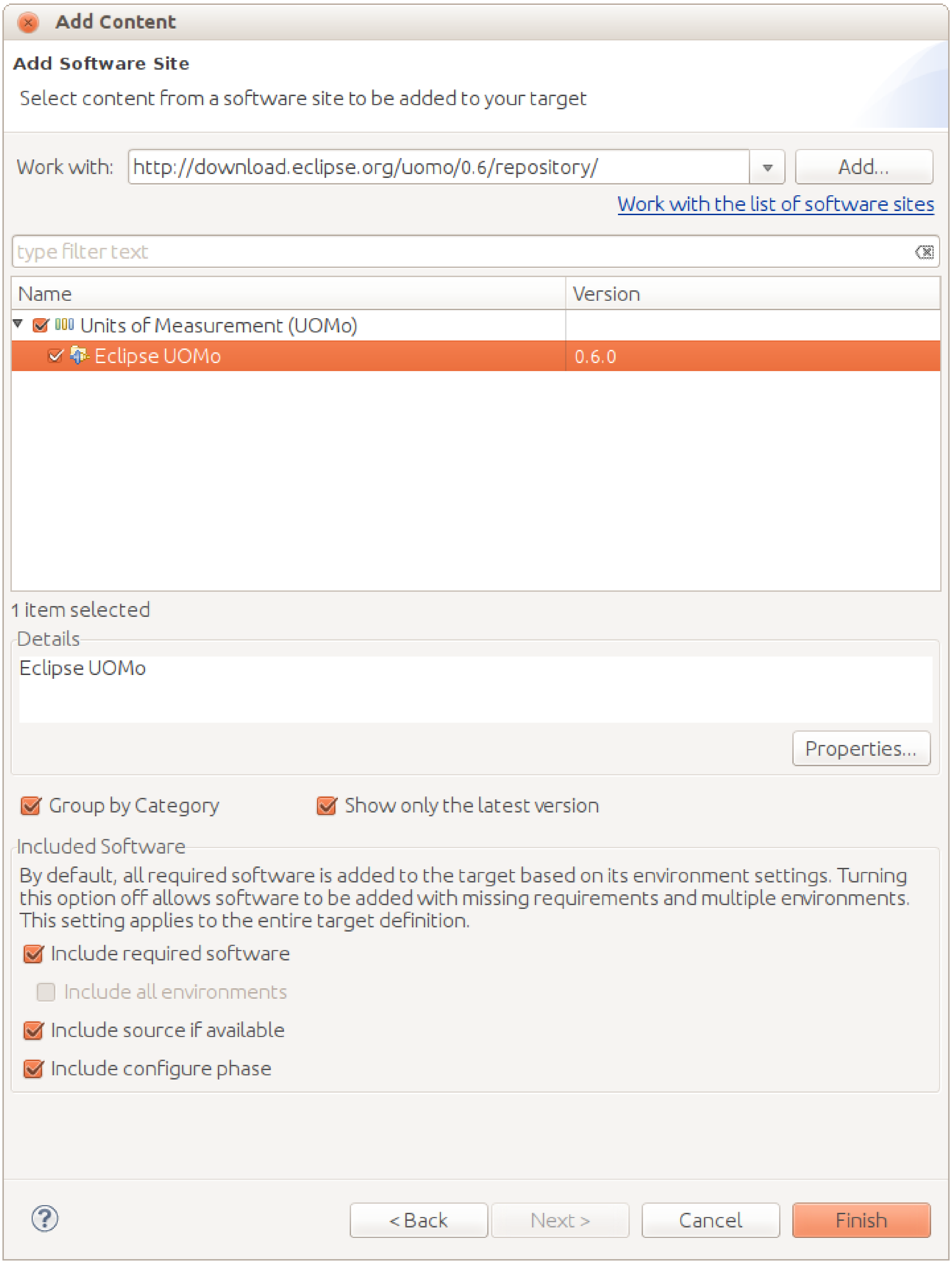
✓ Include source if available

✓ Include configure phase

Firstly, we need to give our target definition a place to get Eclipse plug-ins from. Enter the URL http://download.eclipse.org/-releases/juno into the “Work With” field. Ensure that the “Group by Category” field is un-checked.
Type “Eclipse Platform” into the search field:
Select the Eclipse Platform (without `M*` at the end :- as this is a milestone release)
Click “Finish”

7. Eclipse will do some loading/resolving. . .

8. Now we need to add the UOMo libraries to our target. Click “Add..” again on the target platform editor window. The select “Software Site” again. But this time enter the URL http://download.eclipse.org/uomo/0.6/repository/



Add Content

Add Software Site

Select content from a software site to be added to your target

Work with: http://download.eclipse.org/uomo/0.6/repository/ Add...

Work with the list of software sites

type filter text

Name	Version
✓ Units of Measurement (UOMo)	
✓ Eclipse UOMo	0.6.0

1 item selected

Details

Eclipse UOMo

Properties...

[✓] Group by Category [✓] Show only the latest version

Included Software

By default, all required software is added to the target based on its environment settings. Turning this option off allows software to be added with missing requirements and multiple environments. This setting applies to the entire target definition.

[✓] Include required software

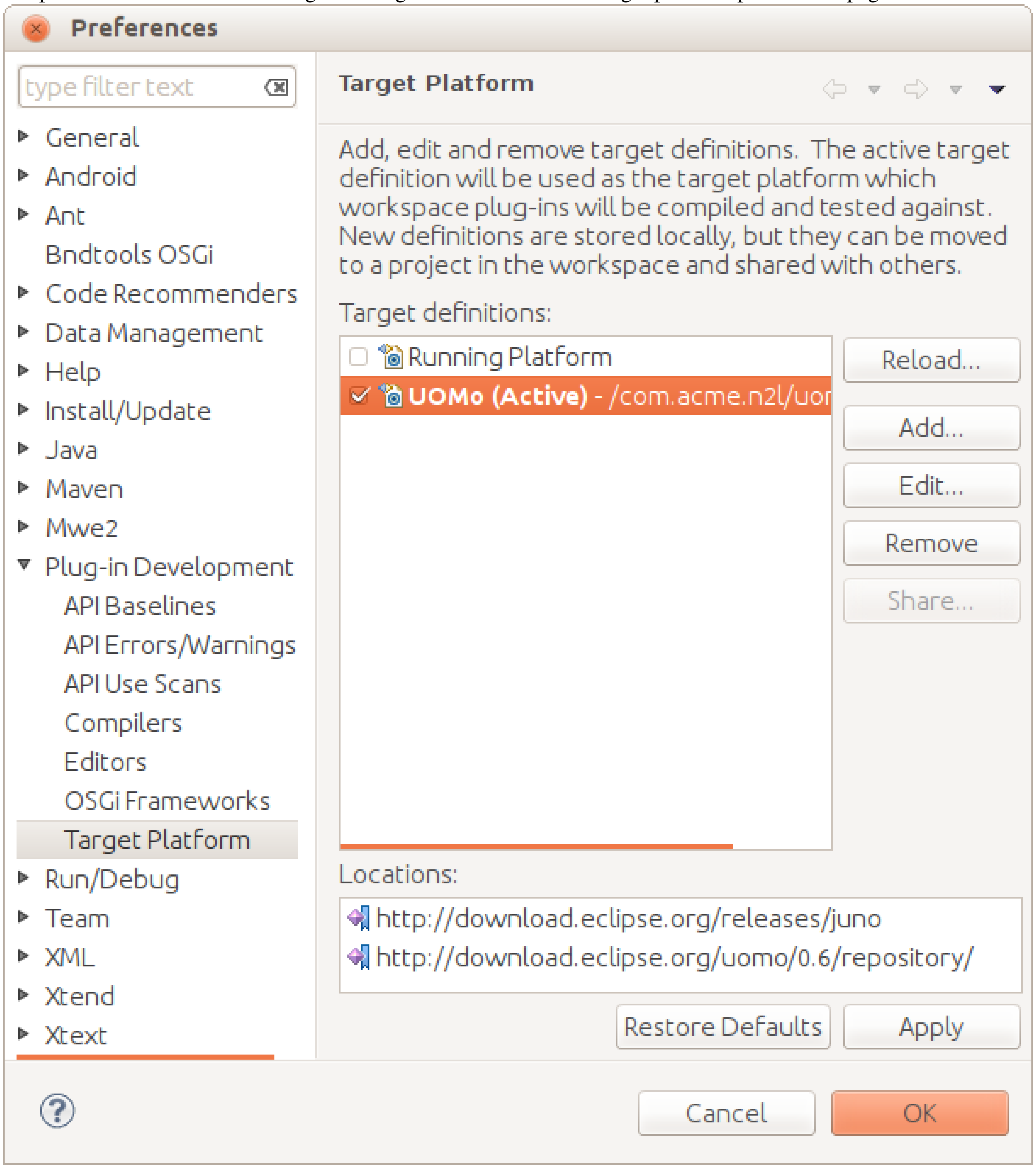
[ ] Include all environments

[✓] Include source if available

[✓] Include configure phase

Select the EclipseUOMo feature. Click “Finish”

9. Eclipse will do some more loading/resolving. You are back at the target platform preferences page:



Target Platform

Add, edit and remove target definitions. The active target definition will be used as the target platform which workspace plug-ins will be compiled and tested against. New definitions are stored locally, but they can be moved to a project in the workspace and shared with others.

Target definitions:

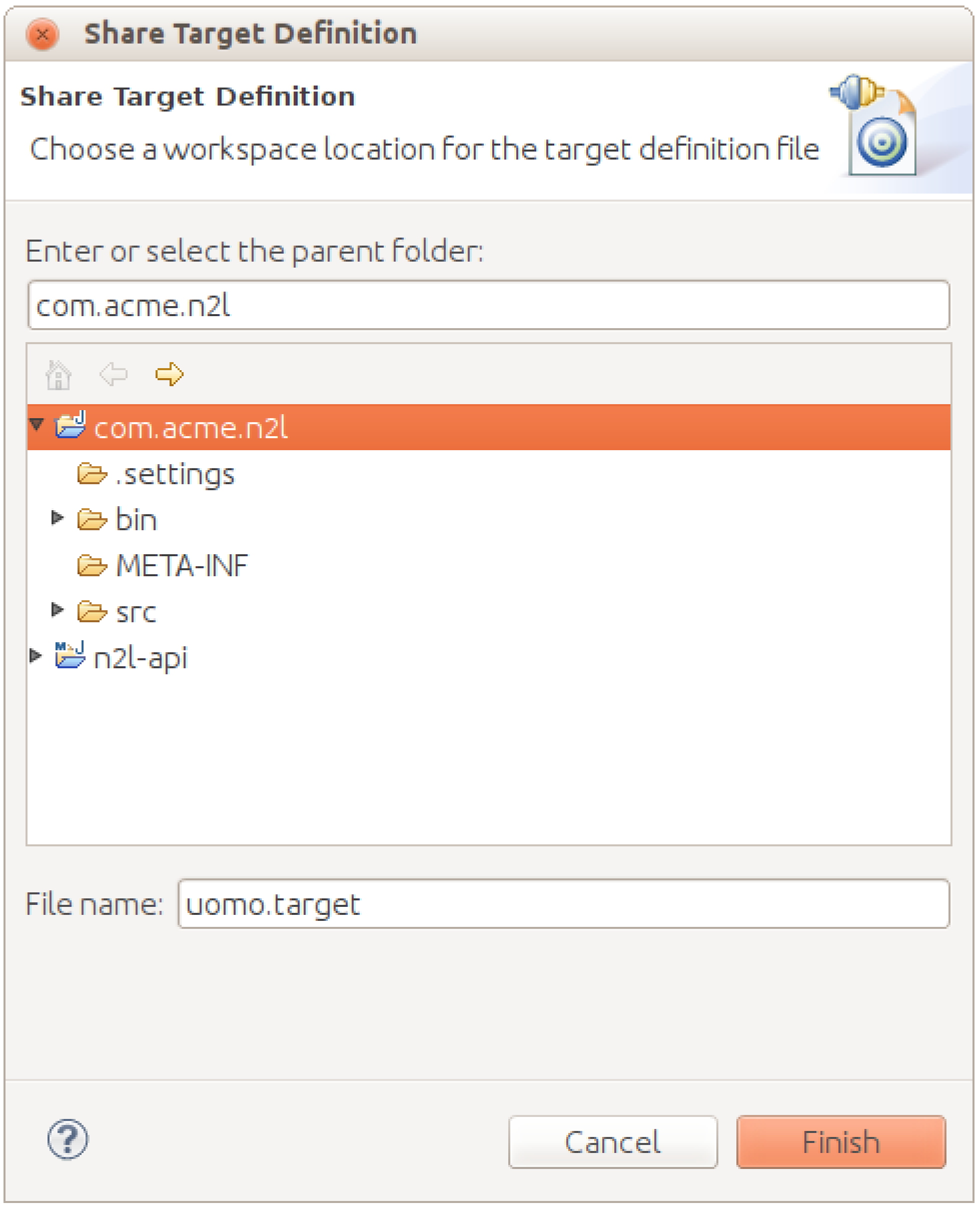
☐ Running Platform
☑ **UOMo (Active)** - /com.acme.n2l/uor

Locations:

http://download.eclipse.org/releases/juno
http://download.eclipse.org/uomo/0.6/repository/

Select UOMo target and click “Apply”

10. **(Optional)** You can click “Share..” if you want to save this target definition as a file for use in the future. Pick a location and a name for the target file in the window as below:



Click “Finish” when you are done.

11. Click “OK” on the preferences page and go back to the main Eclipse IDE.

1.4 Example Code

Now you are ready to start work on our N2L plug-in.

1. Create a new class called `NewtonsSecondLaw` in the package `com.acme.n2l`
2. Now add the following code:

```
package com.acme.n2l;

import org.eclipse.uomo.units.SI;
import org.eclipse.uomo.units.impl.quantity.AccelerationAmount;
import org.eclipse.uomo.units.impl.quantity.ForceAmount;
import org.eclipse.uomo.units.impl.quantity.MassAmount;
```

```
public class NewtonsSecondLaw {

    public static final ForceAmount calculateForce(MassAmount m, AccelerationAmount a)  ↩
        {
        double m_kg = m.doubleValue(SI.KILOGRAM);
        double a_si = a.doubleValue(SI.METRES_PER_SQUARE_SECOND);
        return new ForceAmount(m_kg * a_si, SI.NEWTON);
    }
}
```

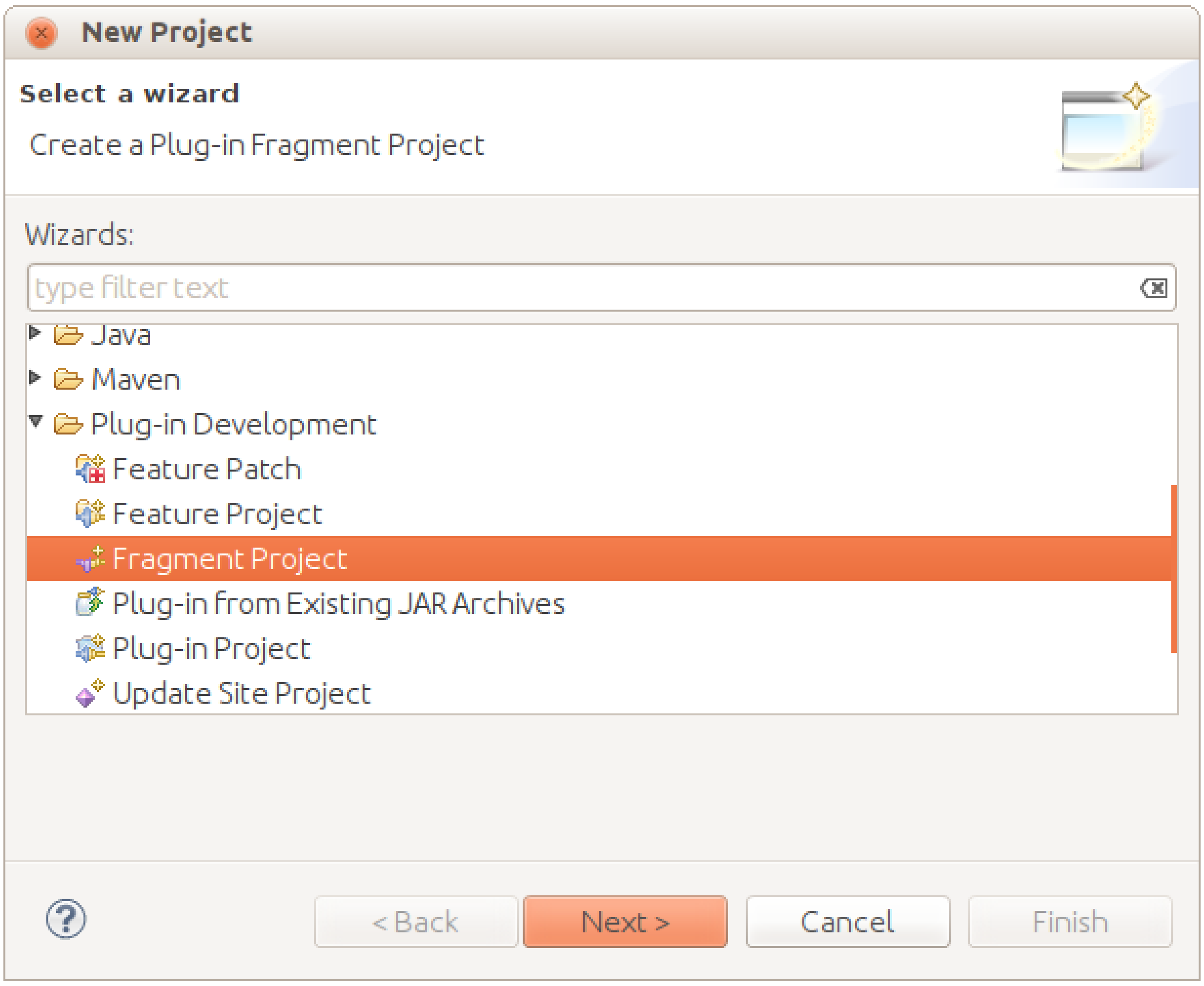
The important part of this code is the `calculateForce` method; it takes as parameters an amount of mass and an amount of acceleration - and returns an amount of force. The units of these parameters are not defined; just that they are of quanity `Mass` and `Acceleration` respectively.

So our code needs to get the absolute value of these in a known unit for calculation - for simplicity we use the SI units Kilogram (kg) and Metres per second per second (m/s2). We then simply perform the multiplication, and create a result using the SI unit for Force - Newtons (N).

3. Now we have our example API implementation - we need to call it to see it do something. Let's create a unit test. (OK purists, we should have created this first to be truly TDD, but this is just a tutorial!)

1.5 Unit Test Plug-in Code

1. Now we are ready to test out our API with a Unit test. *What follows is a little long winded thanks to how Eclipse Plug-ins do unit tests.*
2. Firstly we need to create a New Fragment Project *(File → New → Project...)*:


New Project

Select a wizard

Create a Plug-in Fragment Project

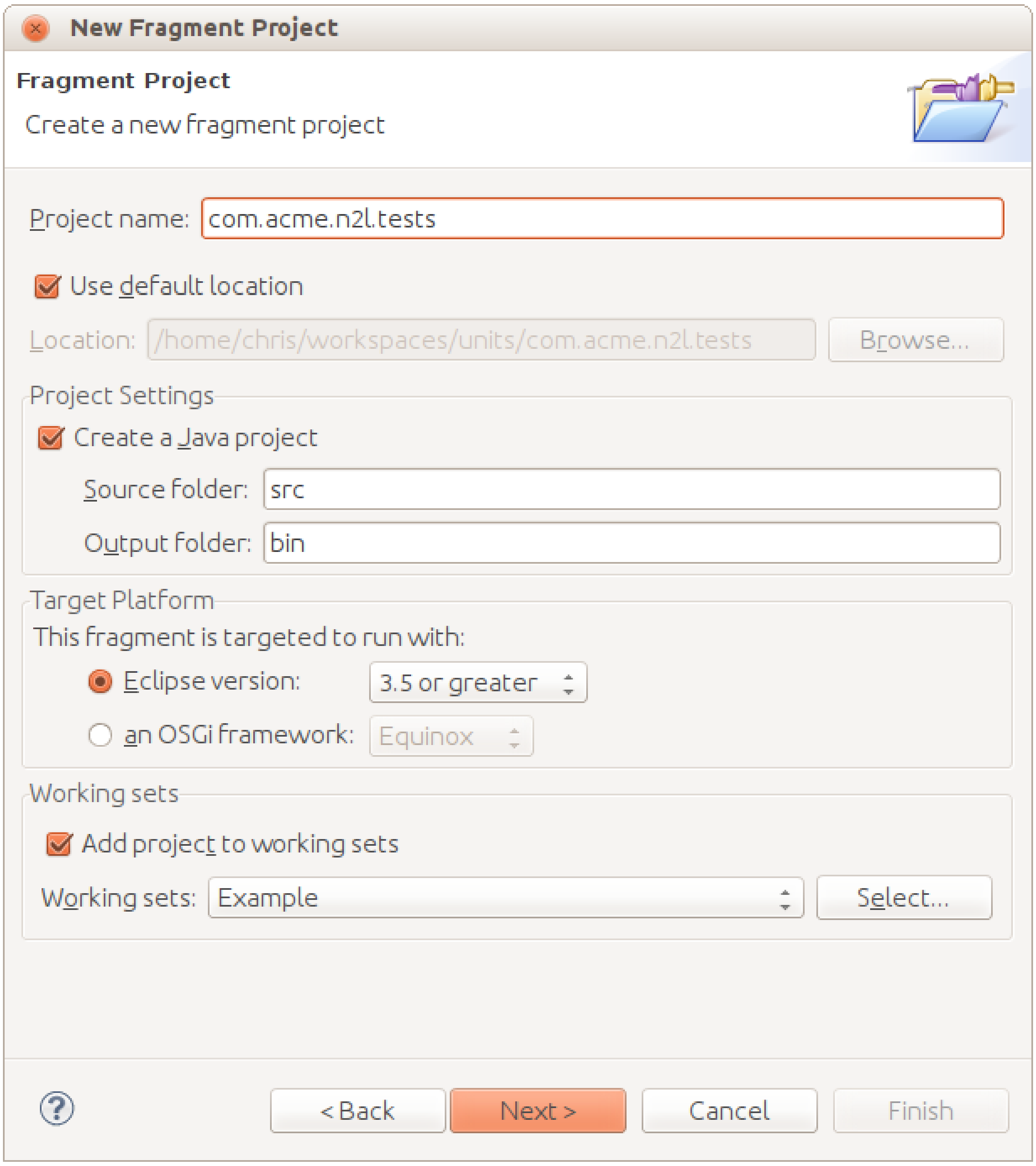
Wizards:

type filter text

- Java
- Maven
- Plug-in Development
  - Feature Patch
  - Feature Project
  - Fragment Project
  - Plug-in from Existing JAR Archives
  - Plug-in Project
  - Update Site Project

Click “Next”

3. Give the fragment the name of our main project with “.test” appended:



New Fragment Project

Fragment Project

Create a new fragment project

Project name: com.acme.n2l.tests

[✓] Use default location

Location: /home/chris/workspaces/units/com.acme.n2l.tests

Project Settings

[✓] Create a Java project

Source folder: src

Output folder: bin

Target Platform

This fragment is targeted to run with:

[✓ Eclipse version:] 3.5 or greater

an OSGi framework: Equinox

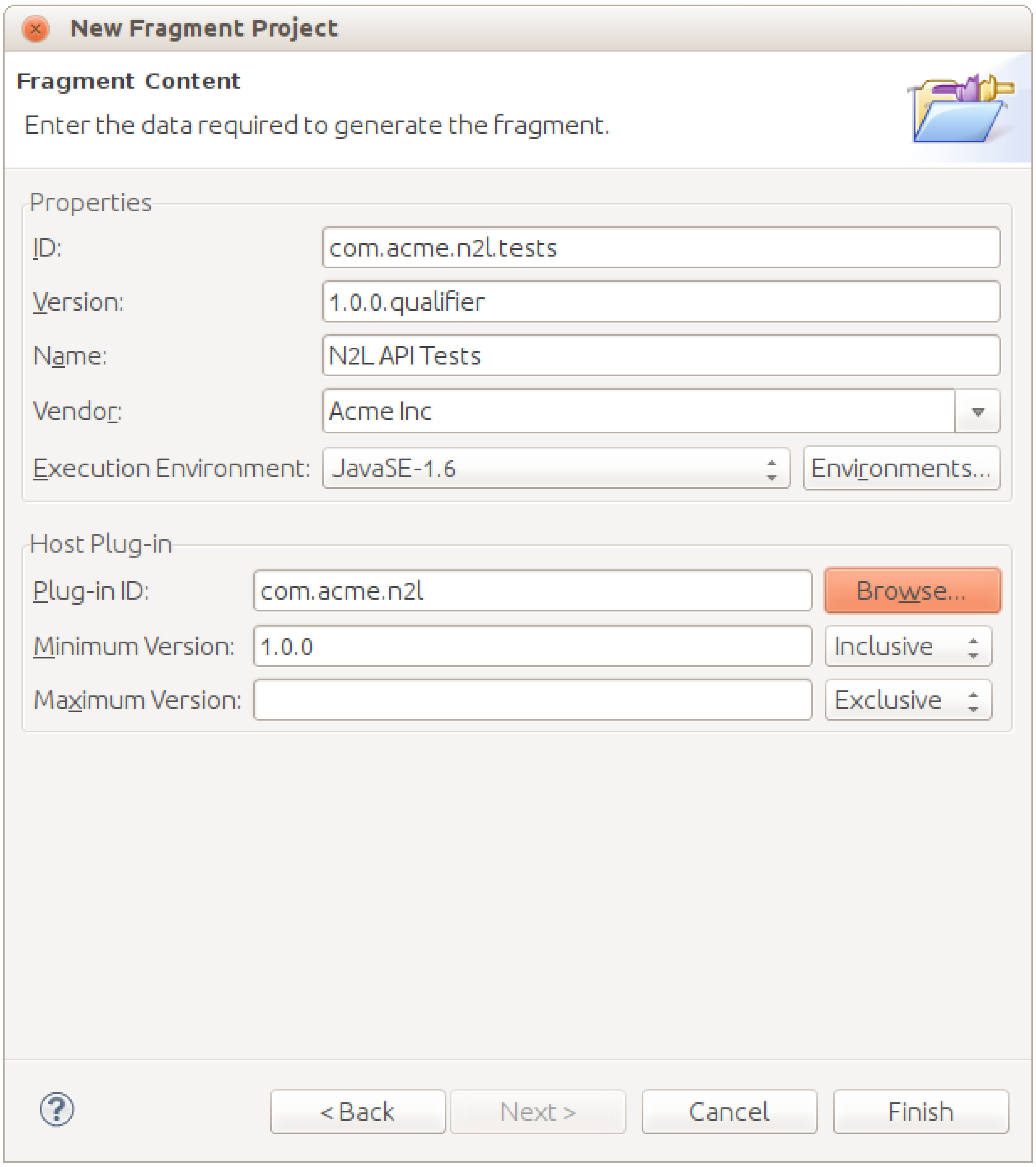
Working sets

[✓] Add project to working sets

Working sets: Example

Click "Next"

4. We need to give the fragment a "host" that is our `com.acme.n2l` plugin:



Properties

ID: com.acme.n2l.tests

Version: 1.0.0.qualifier

Name: N2L API Tests

Vendor: Acme Inc

Execution Environment: JavaSE-1.6 Environments...

Host Plug-in

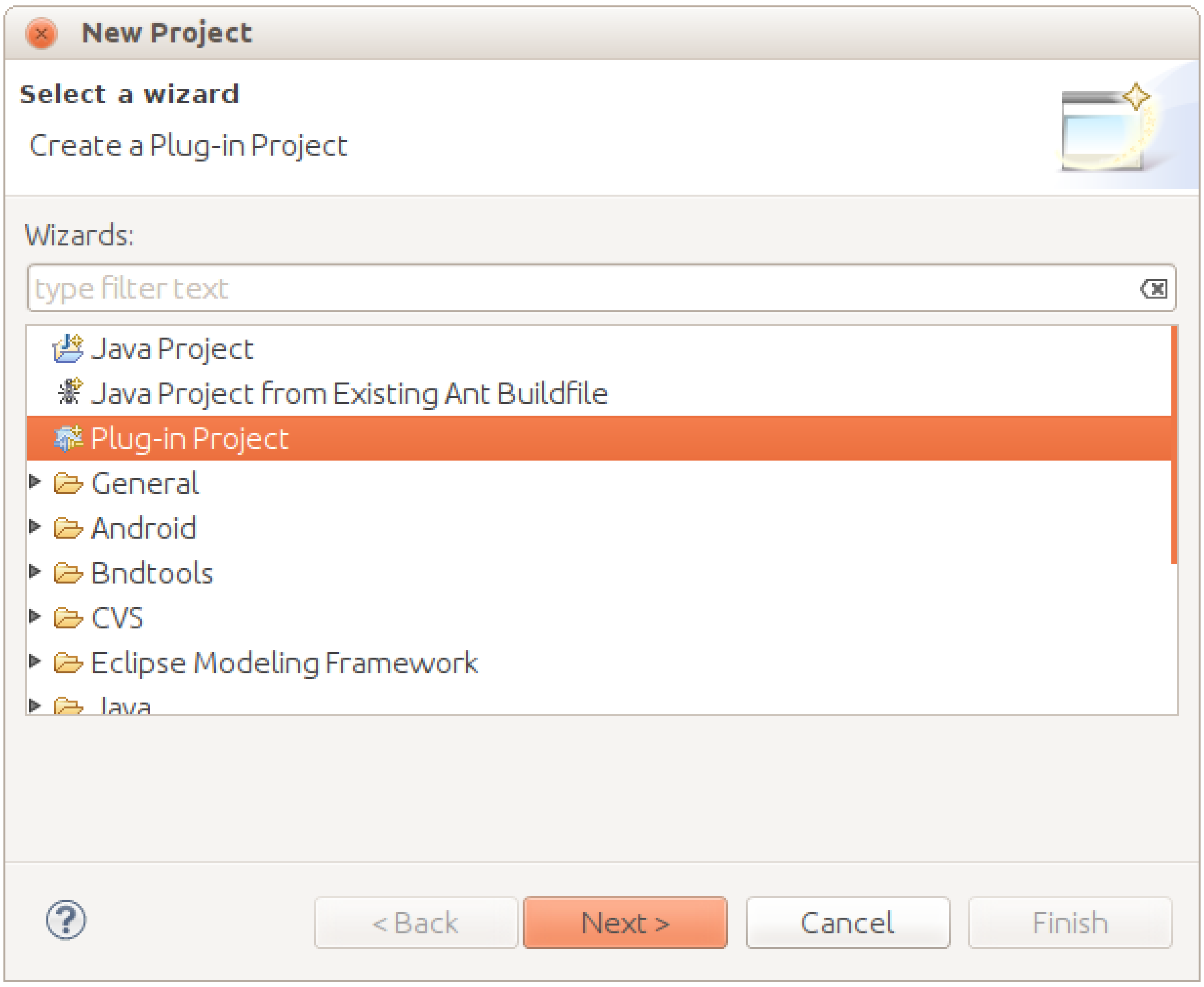
Plug-in ID: com.acme.n2l Browse...

Minimum Version: 1.0.0 Inclusive

Maximum Version: Exclusive

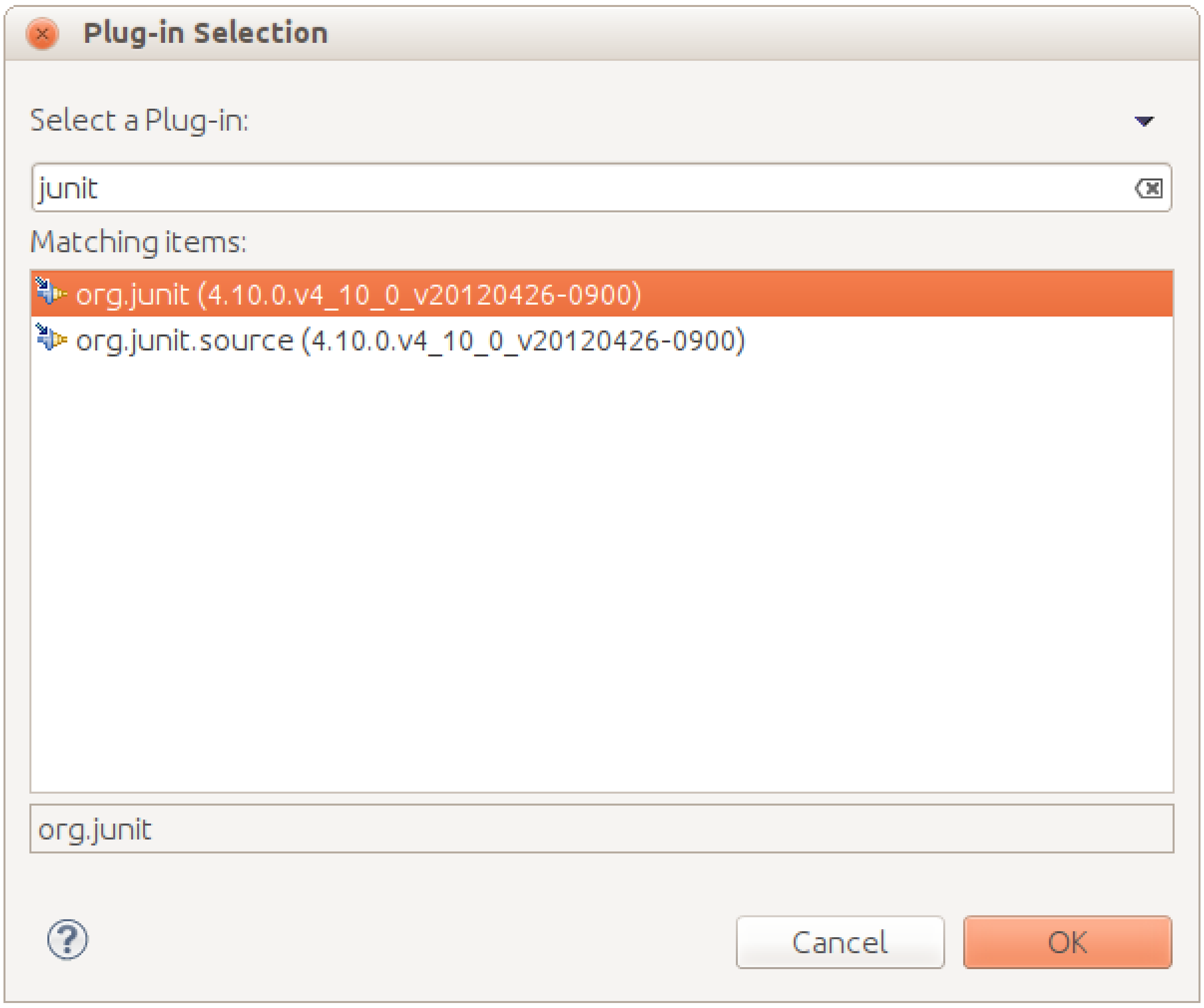
Click “Finish”

5. Now we need to add JUnit to the fragment’s dependencies. Find and double click the `META-INF/MANIFEST.MF` file to open the fragment manifest editor. Select the “Dependencies” tab:



Click “Add. . . ”

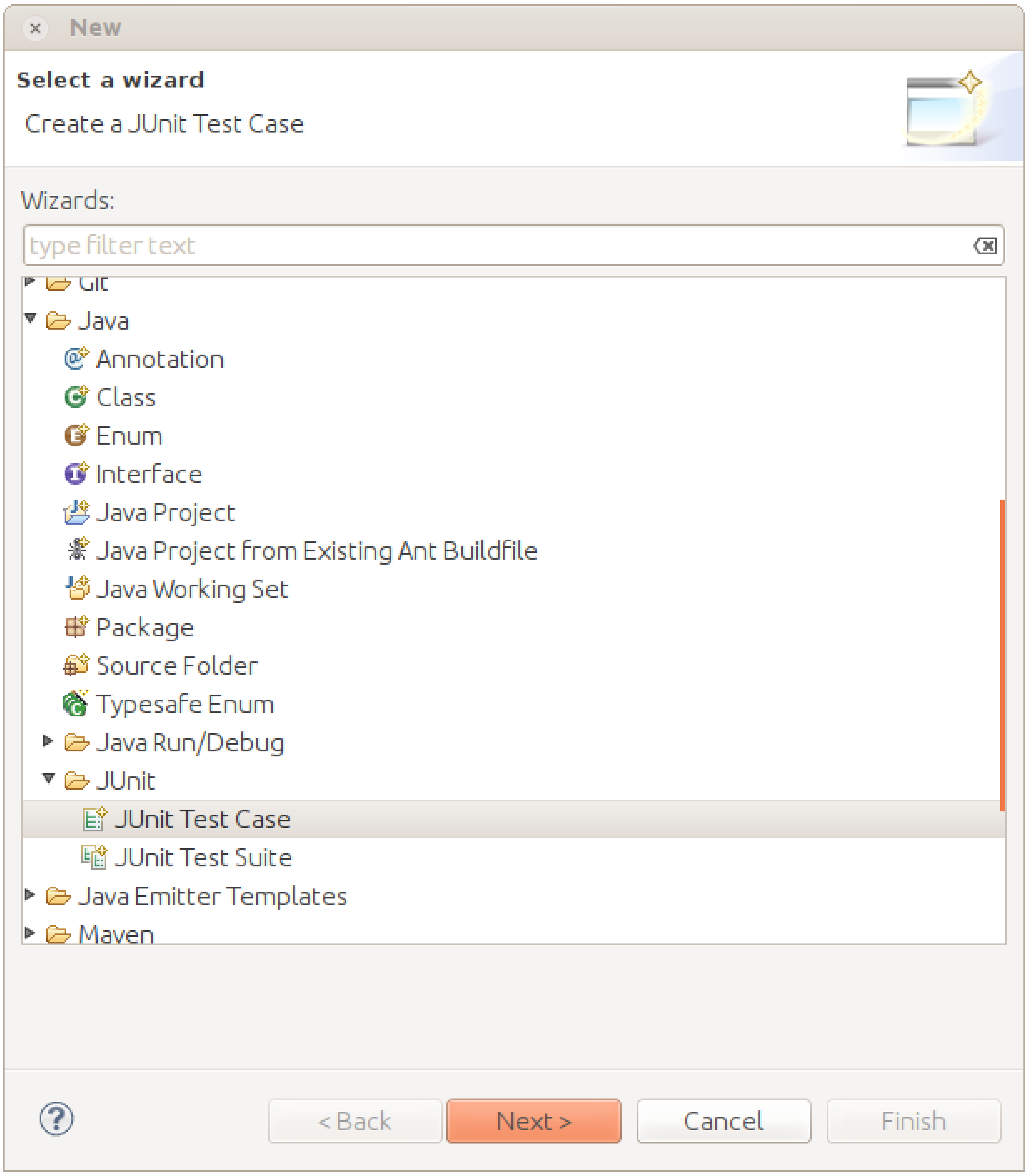
6. Enter `junit` in the search field:


Plug-in Selection
Select a Plug-in:
junit
Matching items:
org.junit (4.10.0.v4_10_0_v20120426-0900)
org.junit.source (4.10.0.v4_10_0_v20120426-0900)
org.junit
Cancel
OK

Select the org.junit plugin and click “OK”

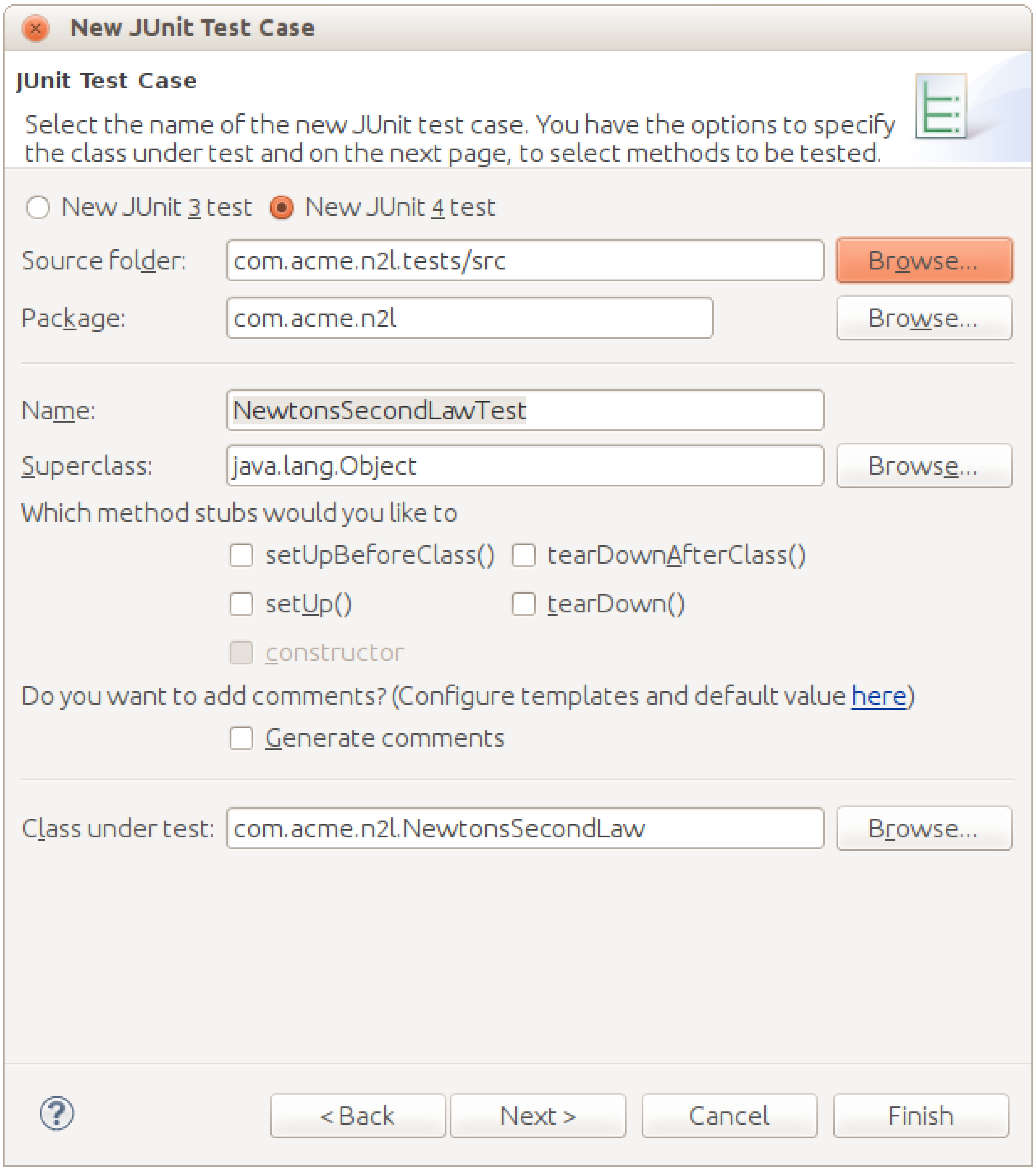
7. Now we create a unit test class in our fragment. Find and right click on our `NewtonsSeconLaw.java` file and choose “New → Other. . . ” from the context menu.

8. Select *Java* → *JUnit* → *JUnit* Test Case from the dialog:



Click “Next”

9. Select the location for the unit test in the src folder of our test fragment. And select the calculateForce method for testing:



New JUnit Test Case

JUnit Test Case

Select the name of the new JUnit test case. You have the options to specify the class under test and on the next page, to select methods to be tested.

○ New JUnit 3 test [✓ ● New JUnit 4 test]

Source folder: com.acme.n2l.tests/src

Package: com.acme.n2l

Name: NewtonsSecondLawTest

Superclass: java.lang.Object

Which method stubs would you like to

☐ setUpBeforeClass() ☐ tearDownAfterClass()

☐ setUp() ☐ tearDown()

☐ constructor

Do you want to add comments? (Configure templates and default value here)

☐ Generate comments

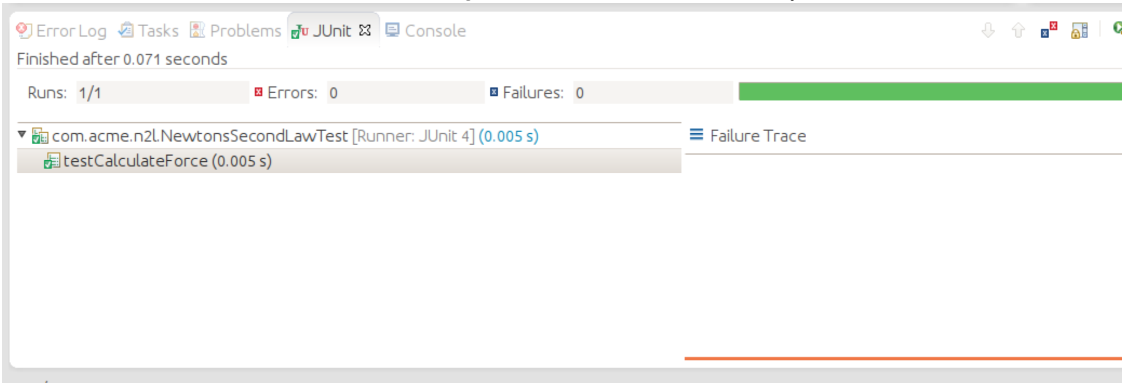
Class under test: com.acme.n2l.NewtonsSecondLaw

Click “Finish”

10. Now we need to replace the template test code with our own. Add the following:

```
@Test
public void testCalculateForce() {
  MassAmount m = new MassAmount(1000, SI.KILOGRAM);
  AccelerationAmount a = new AccelerationAmount(2.5, SI.METRES_PER_SQUARE_SECOND);
  ForceAmount force = NewtonsSecondLaw.calculateForce(m, a);
  assertEquals(2500, force.doubleValue(SI.NEWTON), 0.0001);
}
```

This method simply creates a 1000 kg mass, a 2.5 m/s^2 acceleration and asks our N2L class to calculate the force. We then assert that the calculated force is 2500 Newtons (+/- 0.0001).

11. Save the file and Right click on it - then choose *Run As* → *JUnit Plug-in Test*. The test will run and you should see the JUnit results window appear: 

Error Log Tasks Problems JUnit Console

Finished after 0.071 seconds

Runs: 1/1   Errors: 0   Failures: 0

com.acme.n2l.NewtonsSecondLawTest [Runner: JUnit 4] (0.005 s)
testCalculateForce (0.005 s)

Failure Trace

Now you have now completed and tested your first simple UOMo project.So what. . .

1.6 Conclusions

Now that's the basics out of the way now you can try a few things that demonstrate the power of UOMo and the Units of Measurement API. . .

1. Try changing the units used in your unit test for mass or acceleration to units of other quantities (i.e. seconds, metres, amperes, volts. . .).
   You will see the compiler error created because your unit is not the correct type. . . cool!
2. The same is true when we try to extract a value - change the `force.doubleValue` line in the unit test to use units that are not a force. . .
   same, the compiler error because the unit is not for a force! cool!

This is important, it protects clients of your NewtonsSecondLaw API from making mistakes when they create values to pass to you. They can only pass in legitimate masses or accelerations.

As an example we can add another test method to our test case:

```
@Test
public void testWithOddUnits() {
   // We create a mass in US Pounds!
   MassAmount m = new MassAmount(100, USCustomary.POUND);
   // Now let's create a whacky acceleration unit of our own...
   @SuppressWarnings("unchecked") // we know this creates an acceleration!
   Unit<Acceleration> inch_per_square_second = (Unit<Acceleration>)USCustomary.INCH.divide( ↩
       SI.SECOND).divide(SI.SECOND);
   // Note our N2L API does not know this unit at all!
   AccelerationAmount a = new AccelerationAmount(100, inch_per_square_second);
   ForceAmount force = NewtonsSecondLaw.calculateForce(m, a);
   // Yet our API is able to calculate the corresponding force without change
   // Nice...
   assertEquals(867961.6621451874, force.doubleValue(SI.NEWTON), 0.0000000001);
   // Now let's try to get the result in a weird unit...
   // The "pound-force" is a unit for Force in English engineering units
     // and British gravitational units (http://en.wikipedia.org/wiki/Pound-force)
   Unit<Force> poundForce = SI.NEWTON.multiply(4.448222);
   // and we can now extract the result of our previous calc in that new unit!
   assertEquals(3860886.16071079, force.doubleValue(poundForce), 0.0000000001);
}
```

Nice! Our API can handle weird and wonderful unit values as input and we can extract weird and wonderful unit values as output! And we didn't have to lift a finger.