Eclipse UOMo Tutorial

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:

$$F = M \times A$$

Where:

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

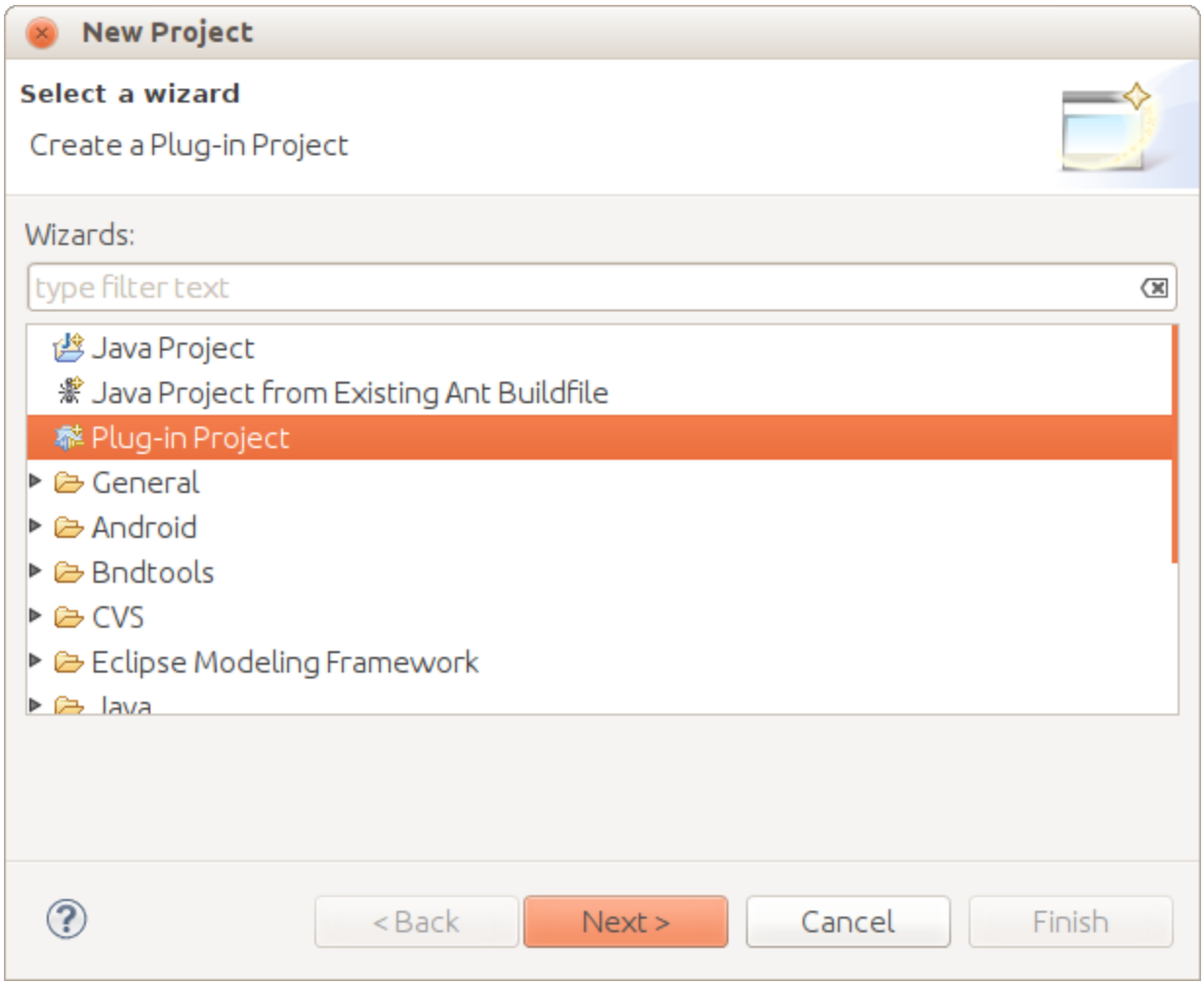
Our example API will provide methods for computing any of the above, given the other two.

Example code for the following is available at GitHub:
https://github.com/duckAsteroid/uomo-example

Getting Started - Eclipse Plugin Project...

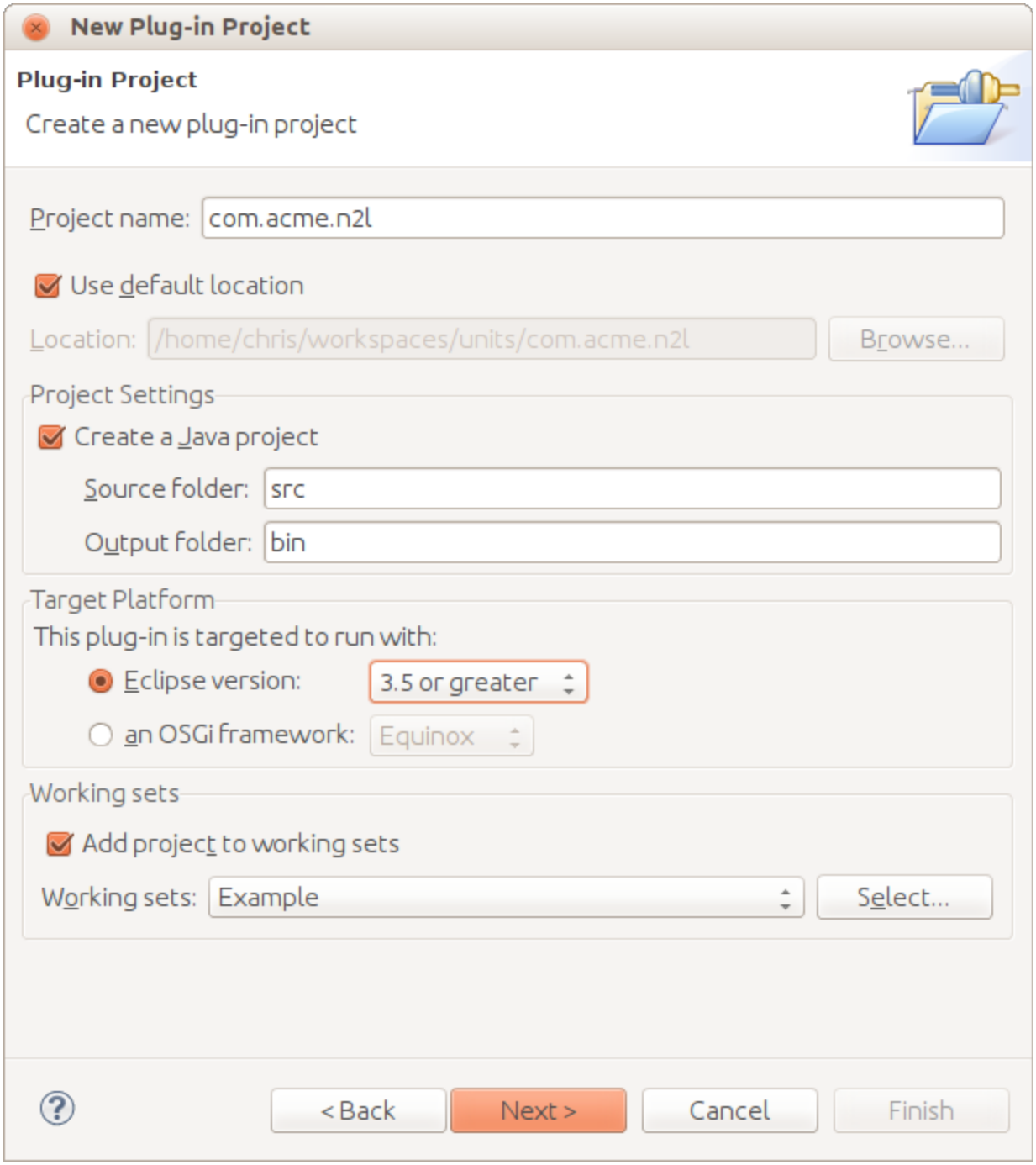
Most of verbosity in the steps that follows are related to complexities of creating Eclipse plug-ins rather than anything hard about using UOMo...

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



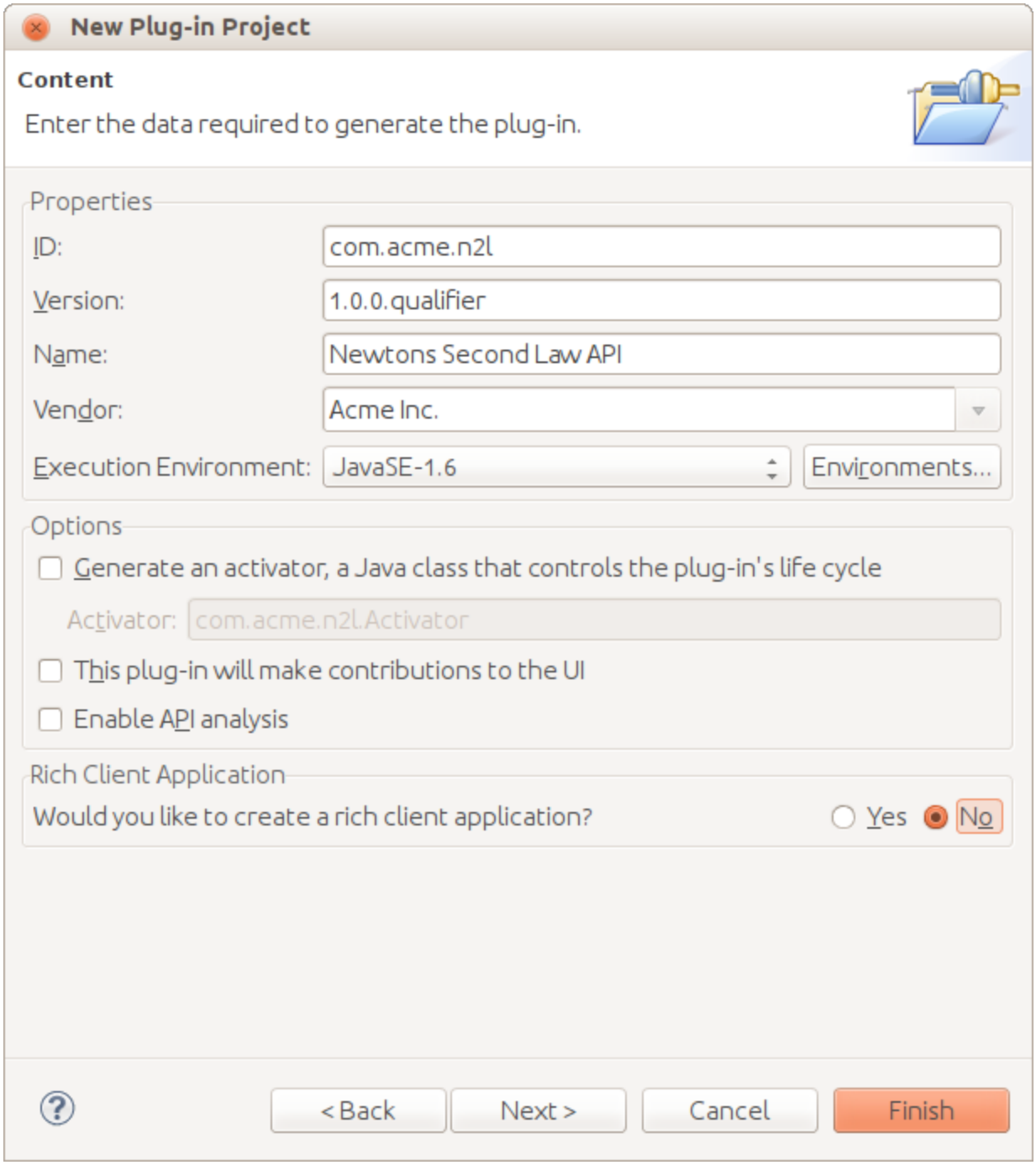
Click “Next”

3. Enter a name for the plugin project (e.g. com.acme.n2l):

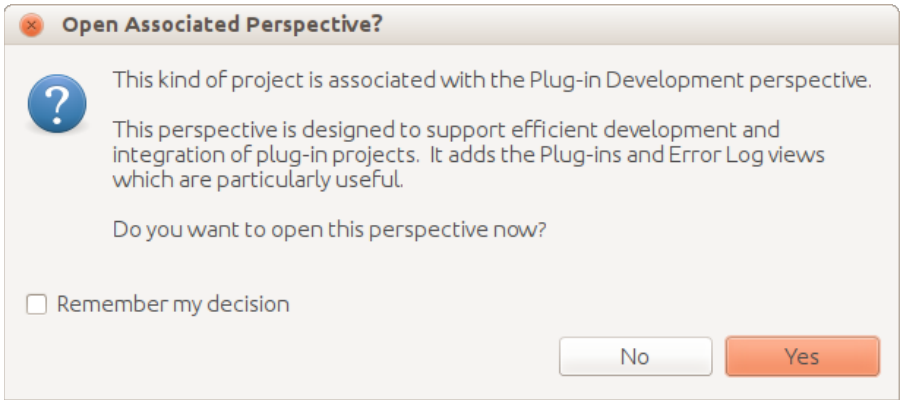


Click “Next”

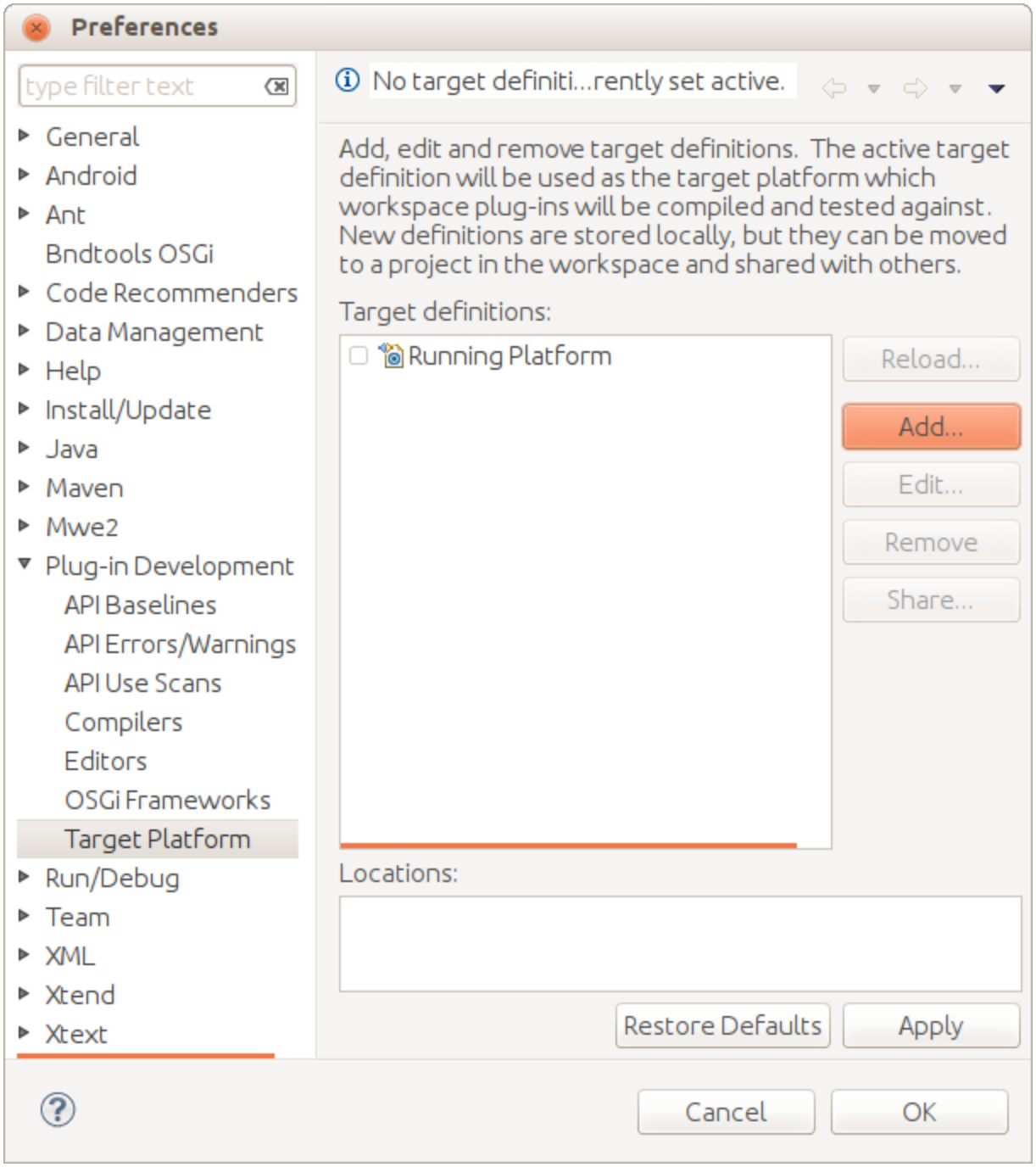
4. You can now customise the plugin details as follows:



Ensure that all checkboxes under “Options” are de-selected.
Click “Finish”
If the following dialog appears click “Yes” to open the PDE perspective:

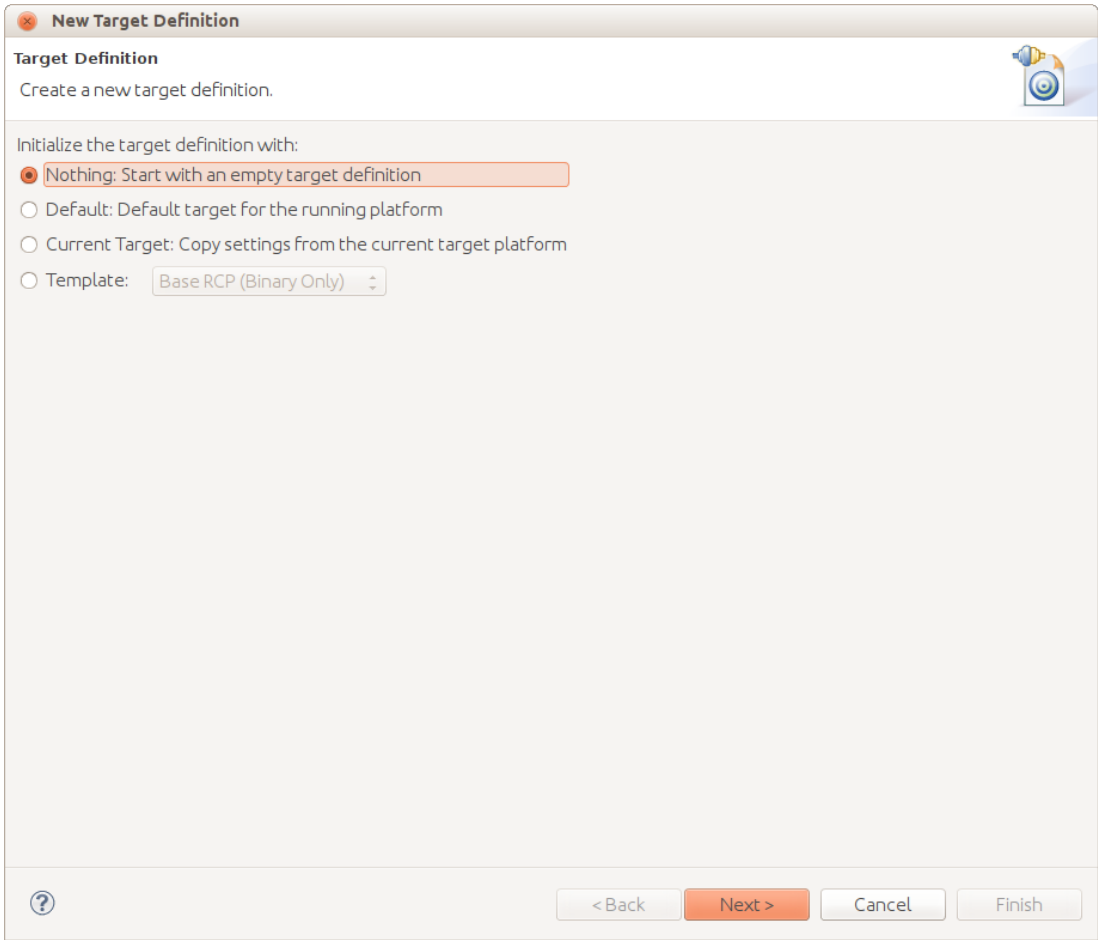


5. Eclipse will create your new project. Now we need to configure Eclipse so that it knows about UOMo and where it lives to build against.
   Click the menu “Window -> Preferences”
6. Navigate to the “Plug-in Development -> Target Platform” section of the preferences dialog:



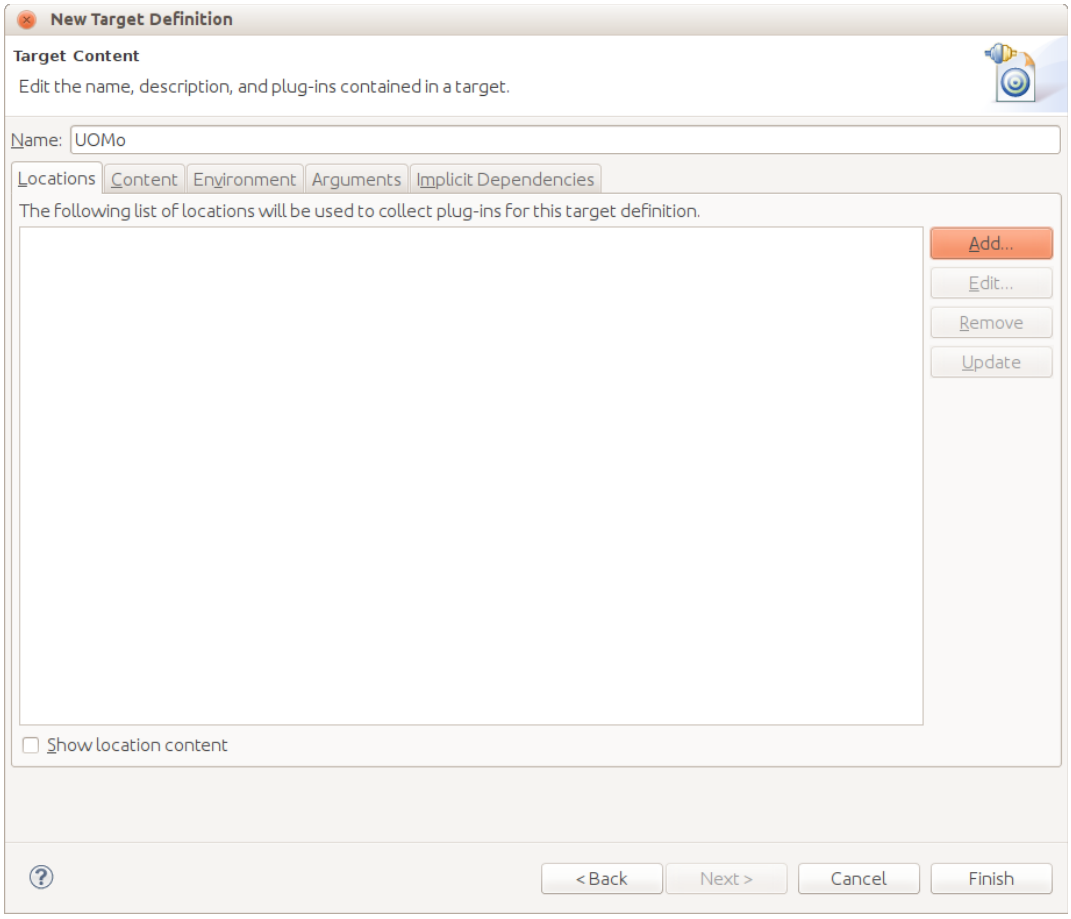
Click “Add...”

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



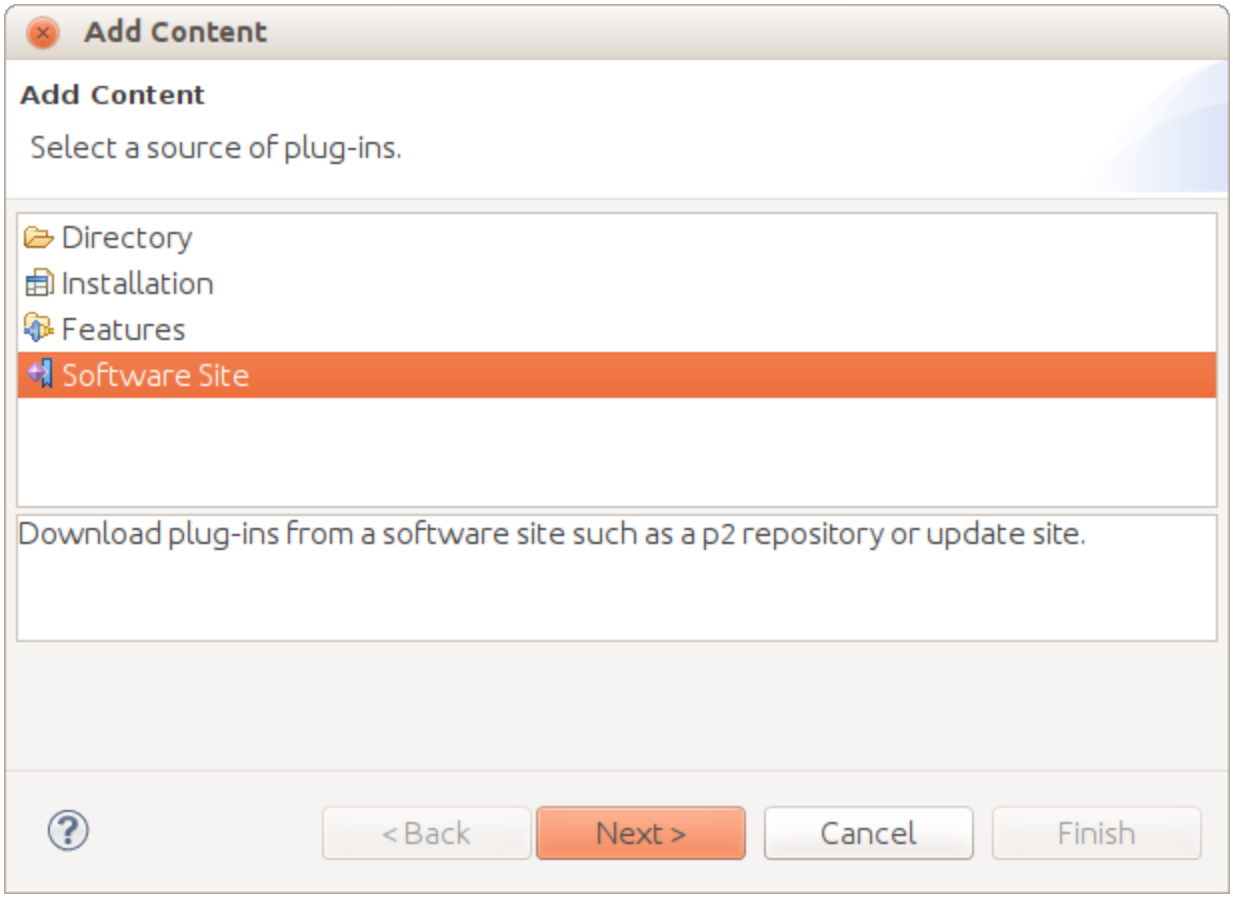
Click “Next”

8. Give the new target a name, such as “UOMo”:



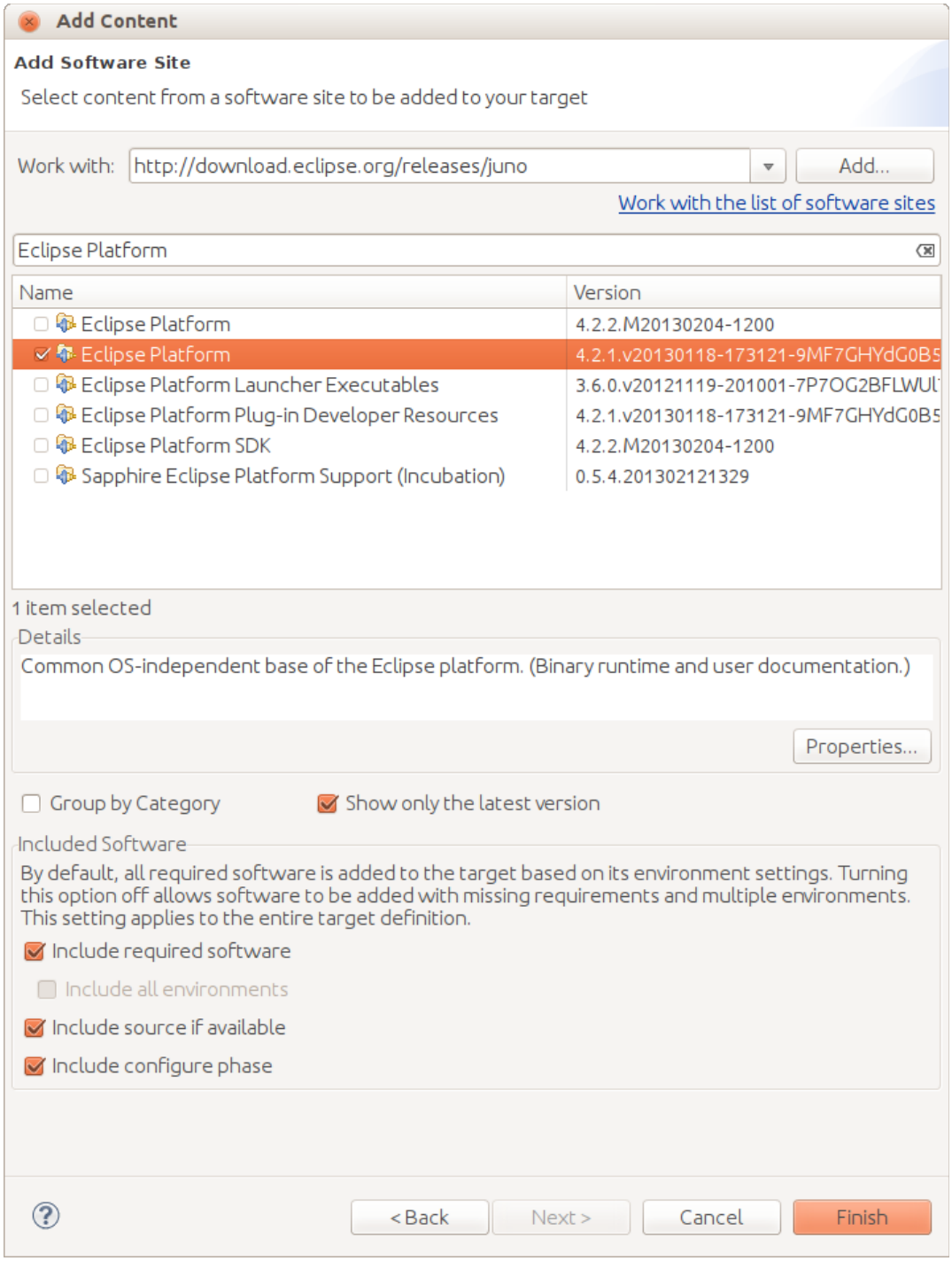
Click “Add...”

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



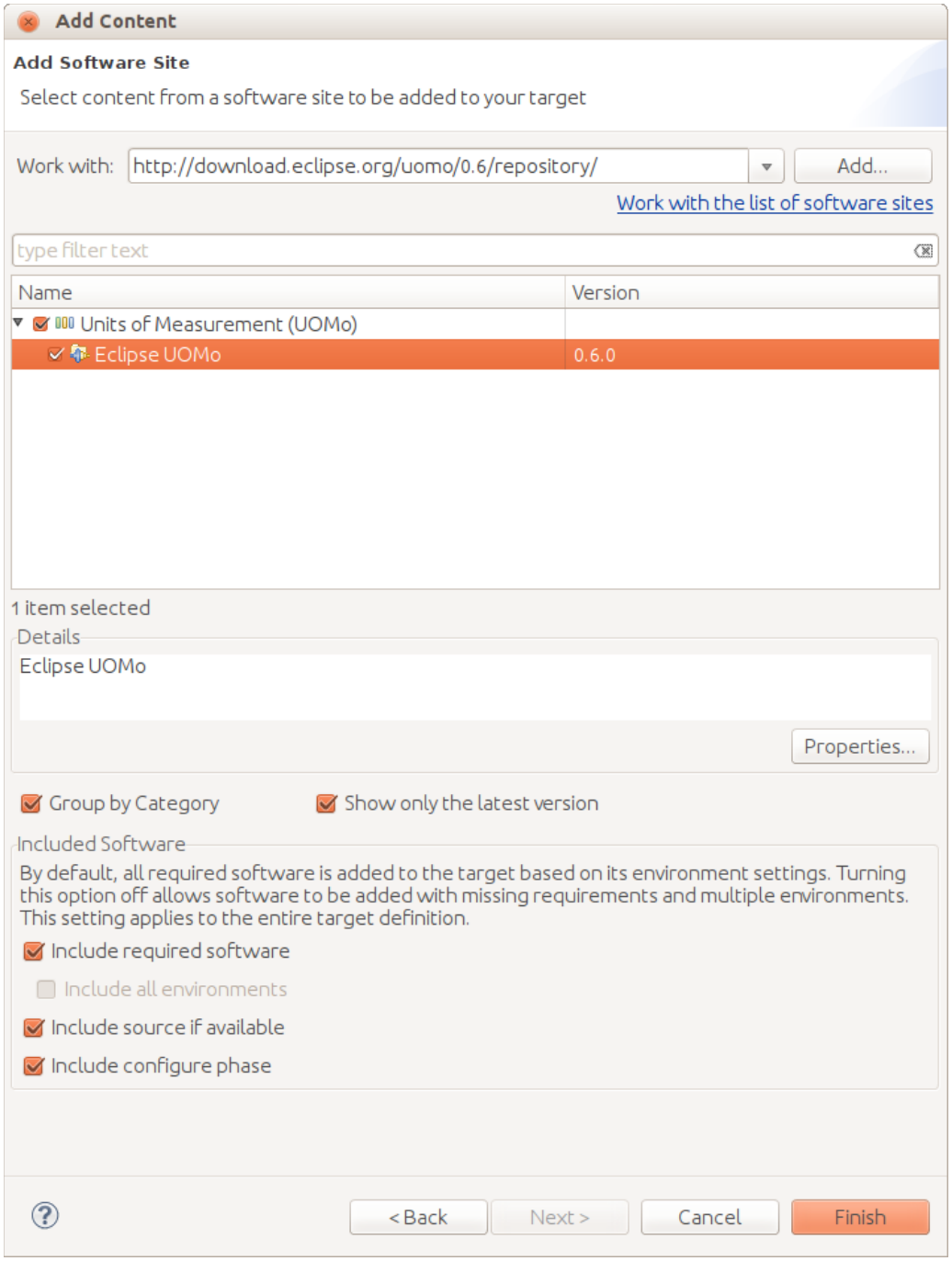
Click “Next”

10. 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:



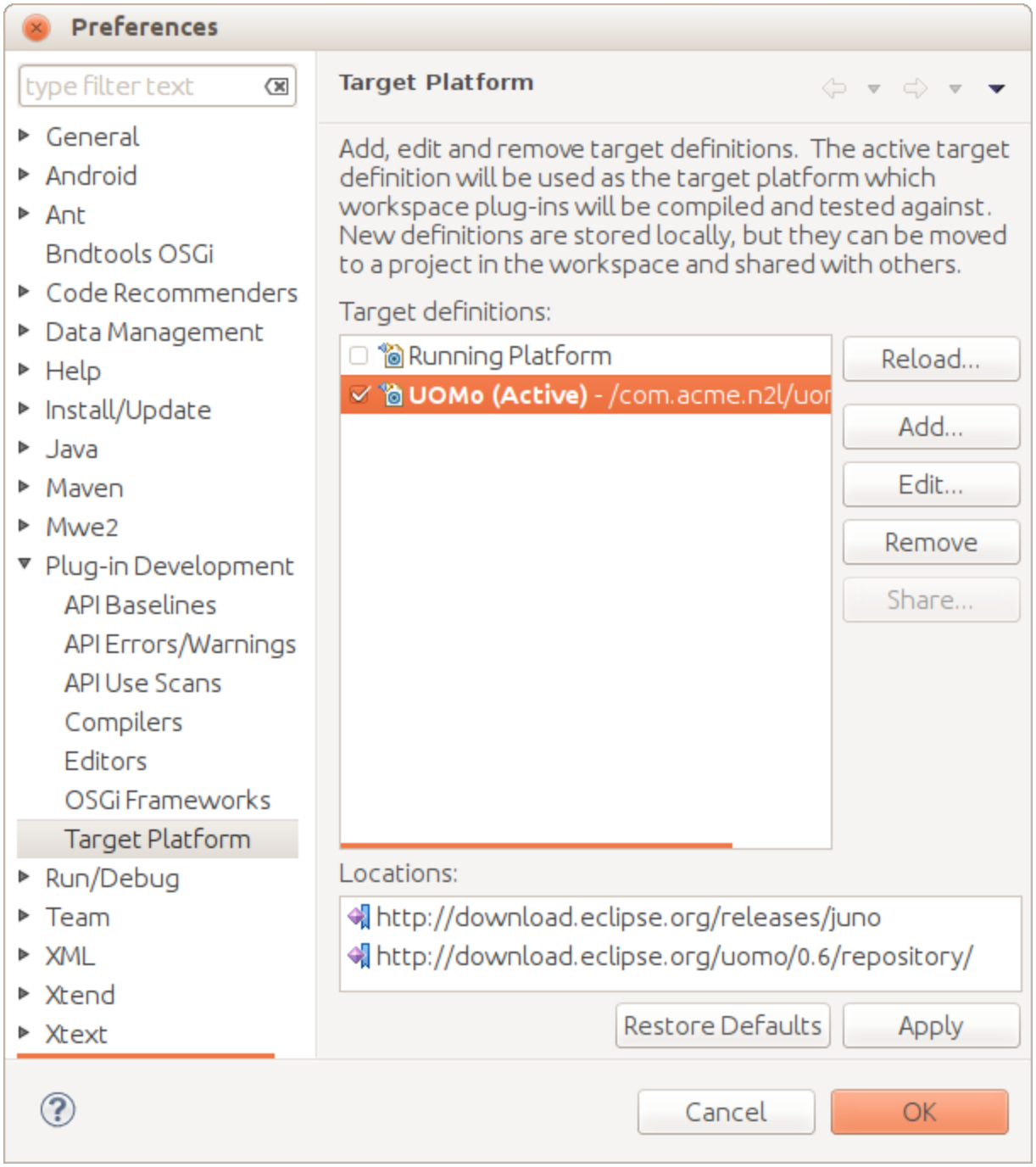
Select the Eclipse Platform (with out M* at the end :- as this is a milestone release)
Click “Finish”

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



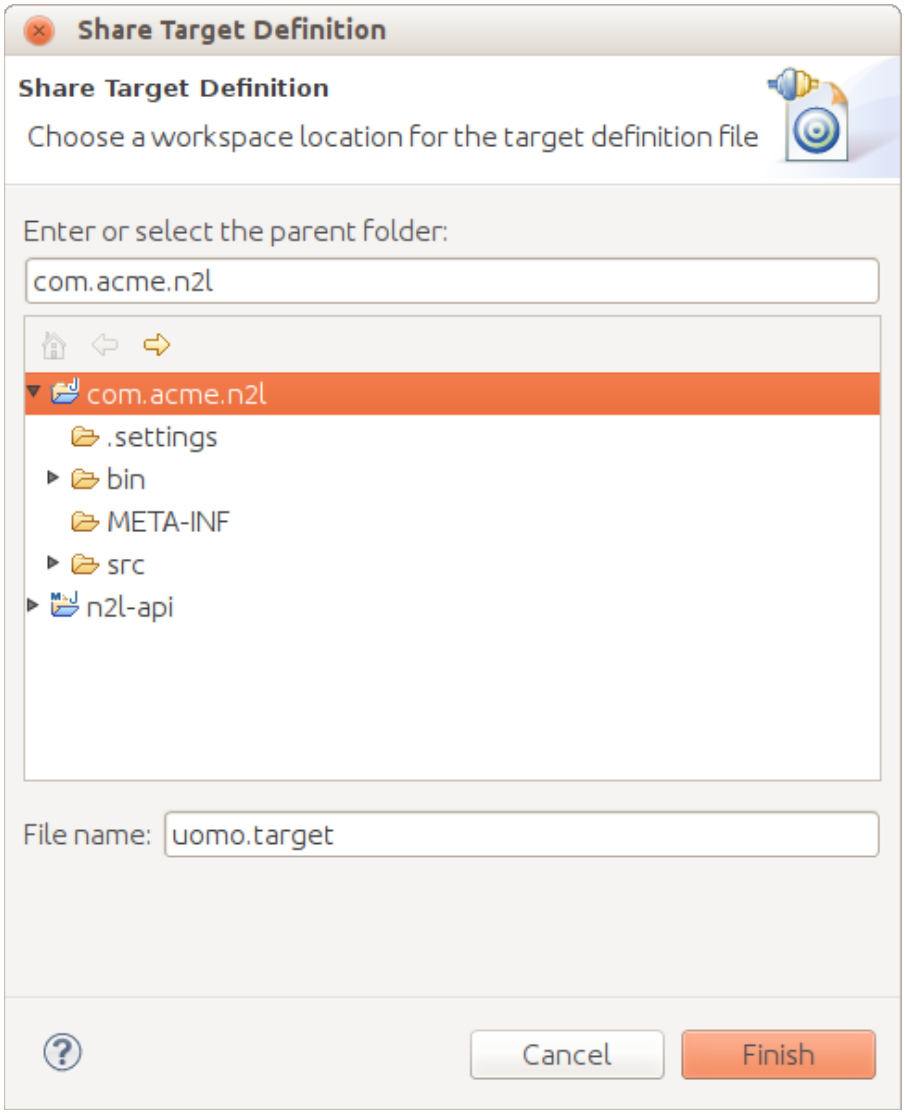
Select the EclipseUOMo feature.
Click “Finish”

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



Select UOMo target and click “Apply”

13. (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.

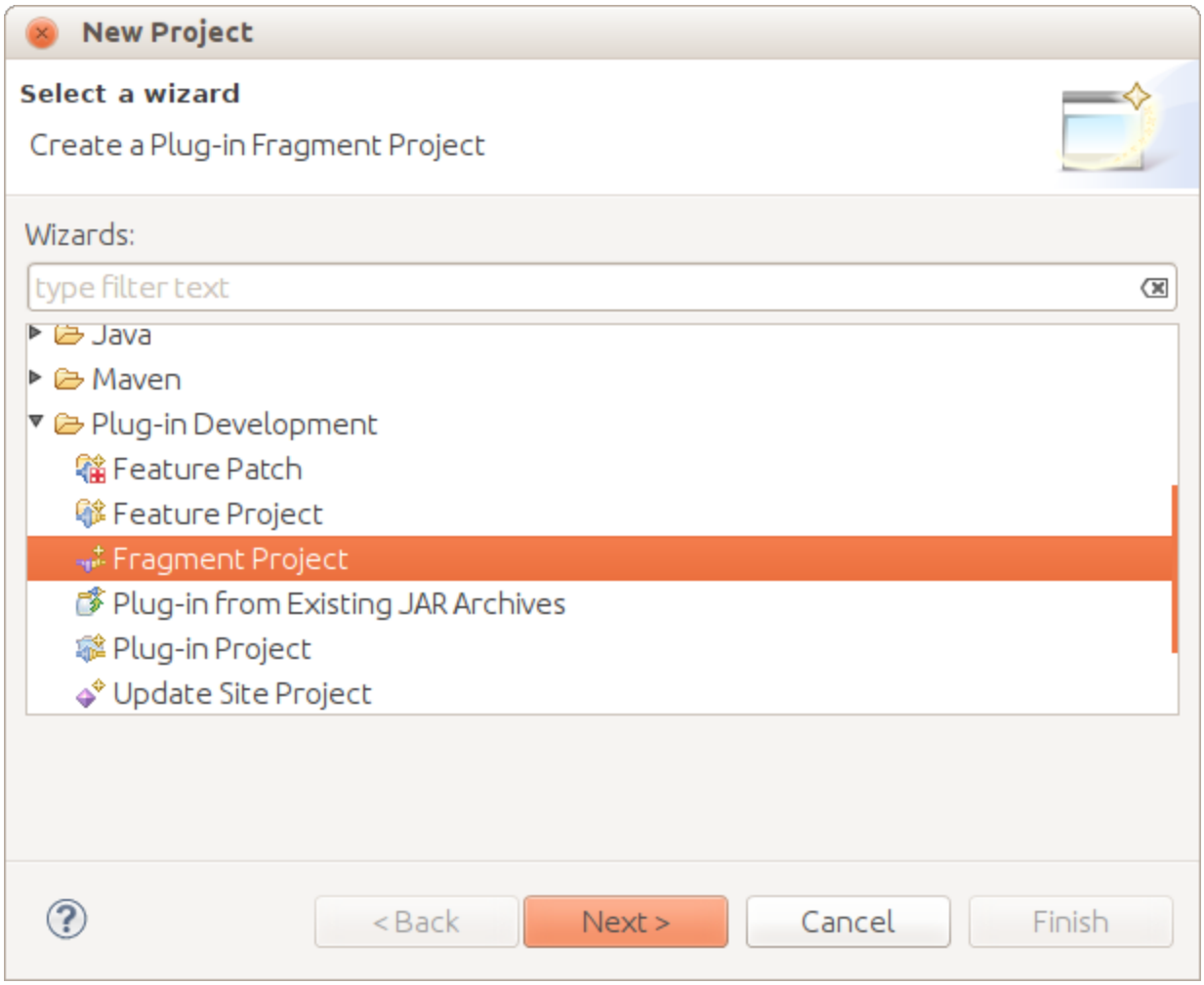
14. Click “OK” on the preferences page
15. Now you are ready to start work on our N2L plug-in.
    Create a new class called `NewtonsSecondLaw` in the package `com.acme.n2l`
    Now add the following code:

```
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);
      }
}
```

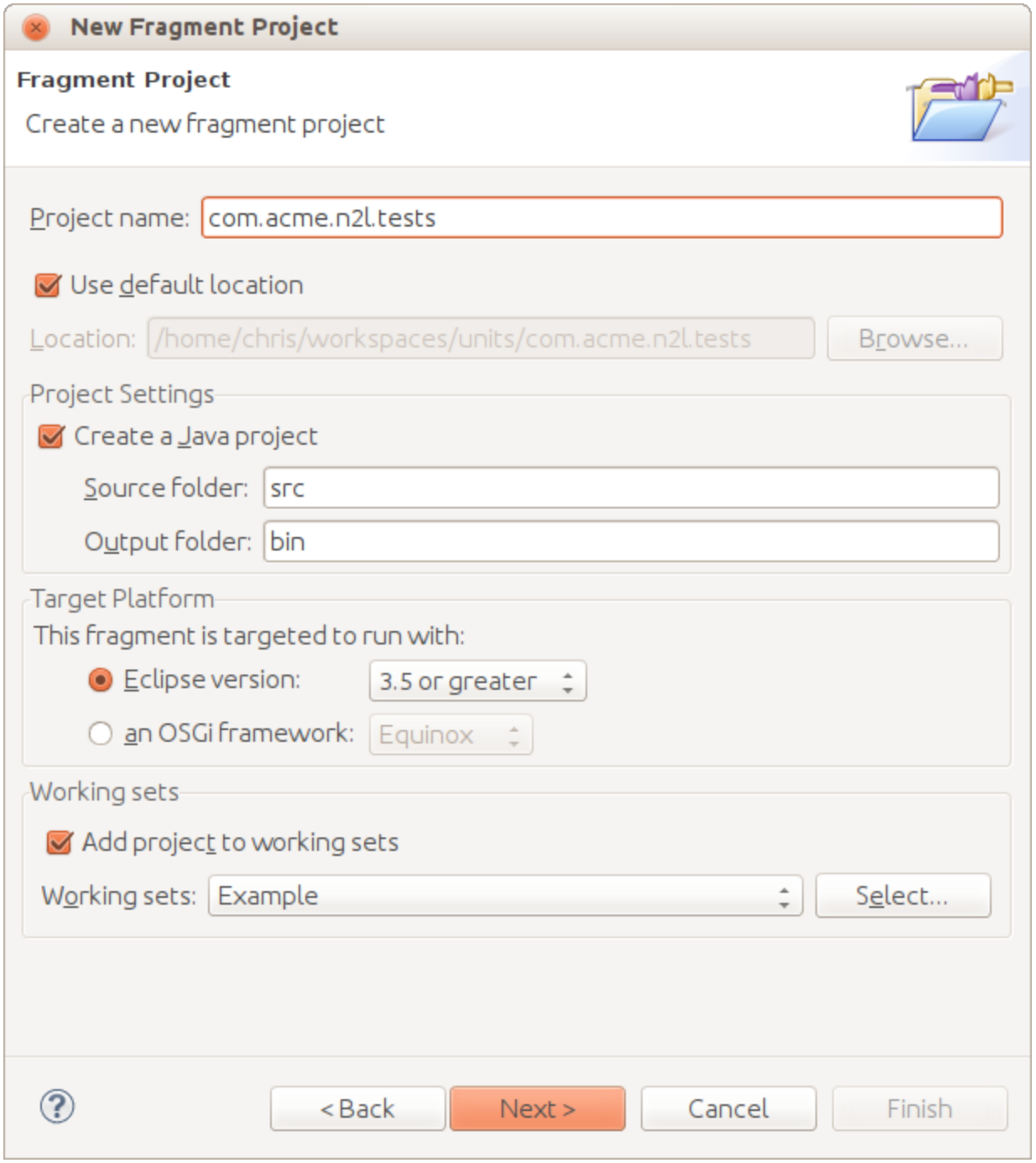
The important part of this code is the calculateForce method; it takes as parameters an amount of mass and an amount of acceleration - and returns an amount of force. The units of these parameters are not defined; just that they are of quanity Mass and Acceleration respectively. So our code needs to 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/s^2).
We then simply perform the multiplication, and create a result using the SI unit for Force - Newtons (N).

16. 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..
17. Firstly we need to create a New Fragment Project (File -> New -> Project...):



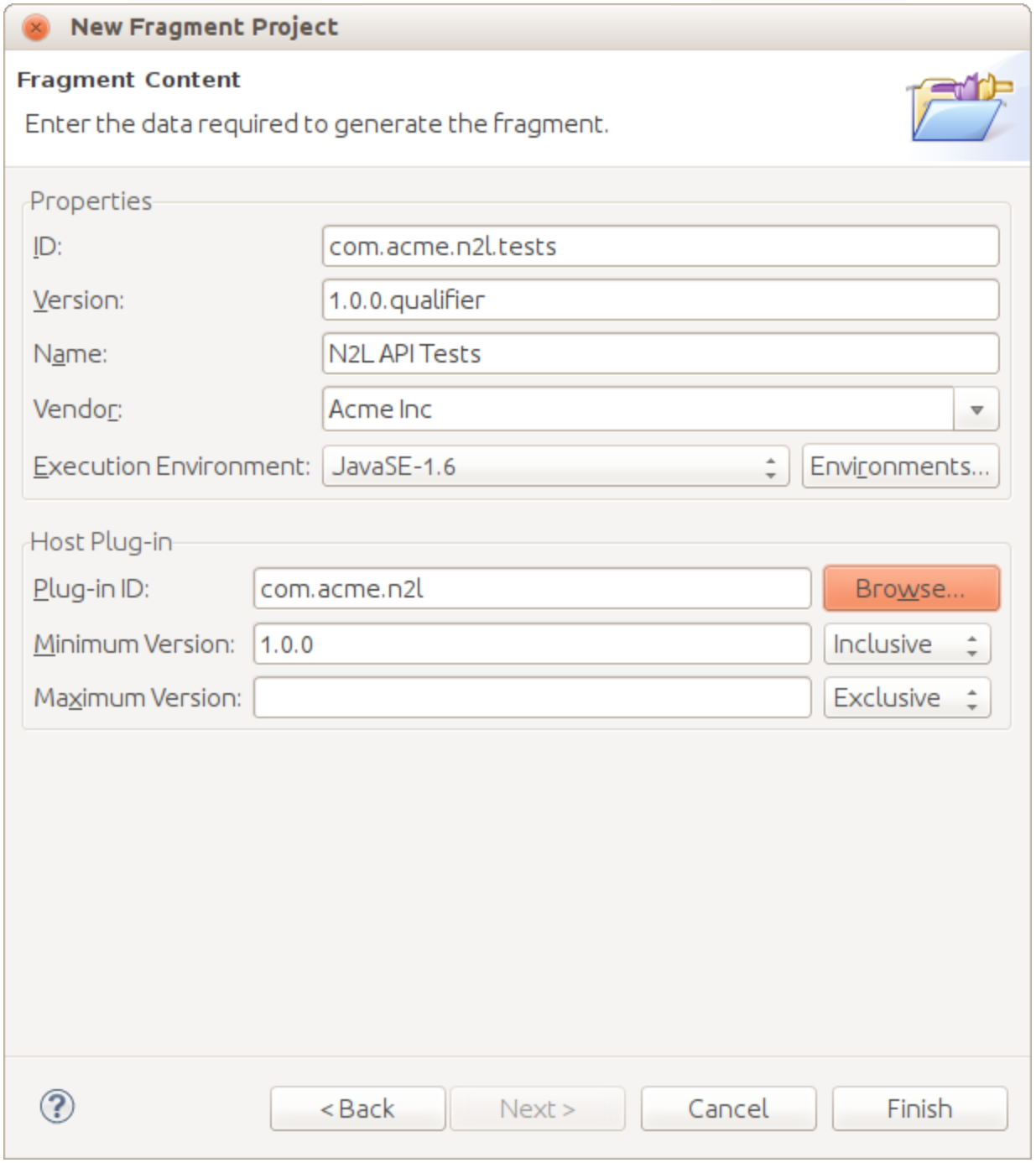
Click “Next”

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



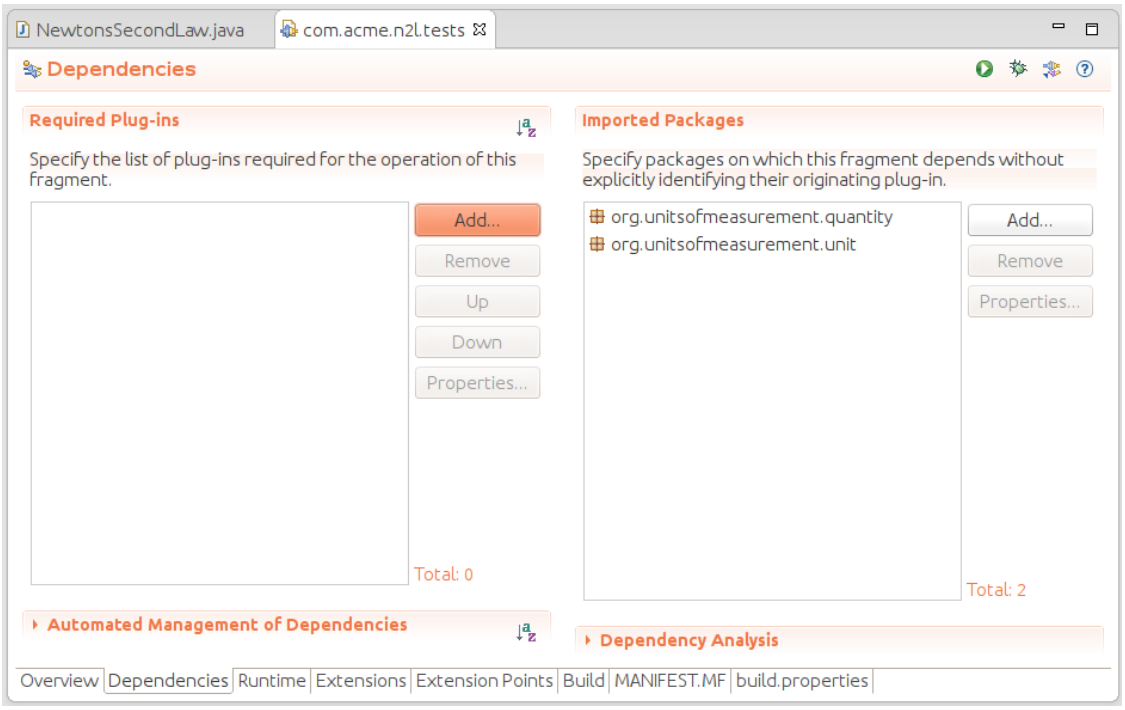
Click “Next”

19. We need to give the fragment a “host” that is our `com.acme.n2l` plugin:



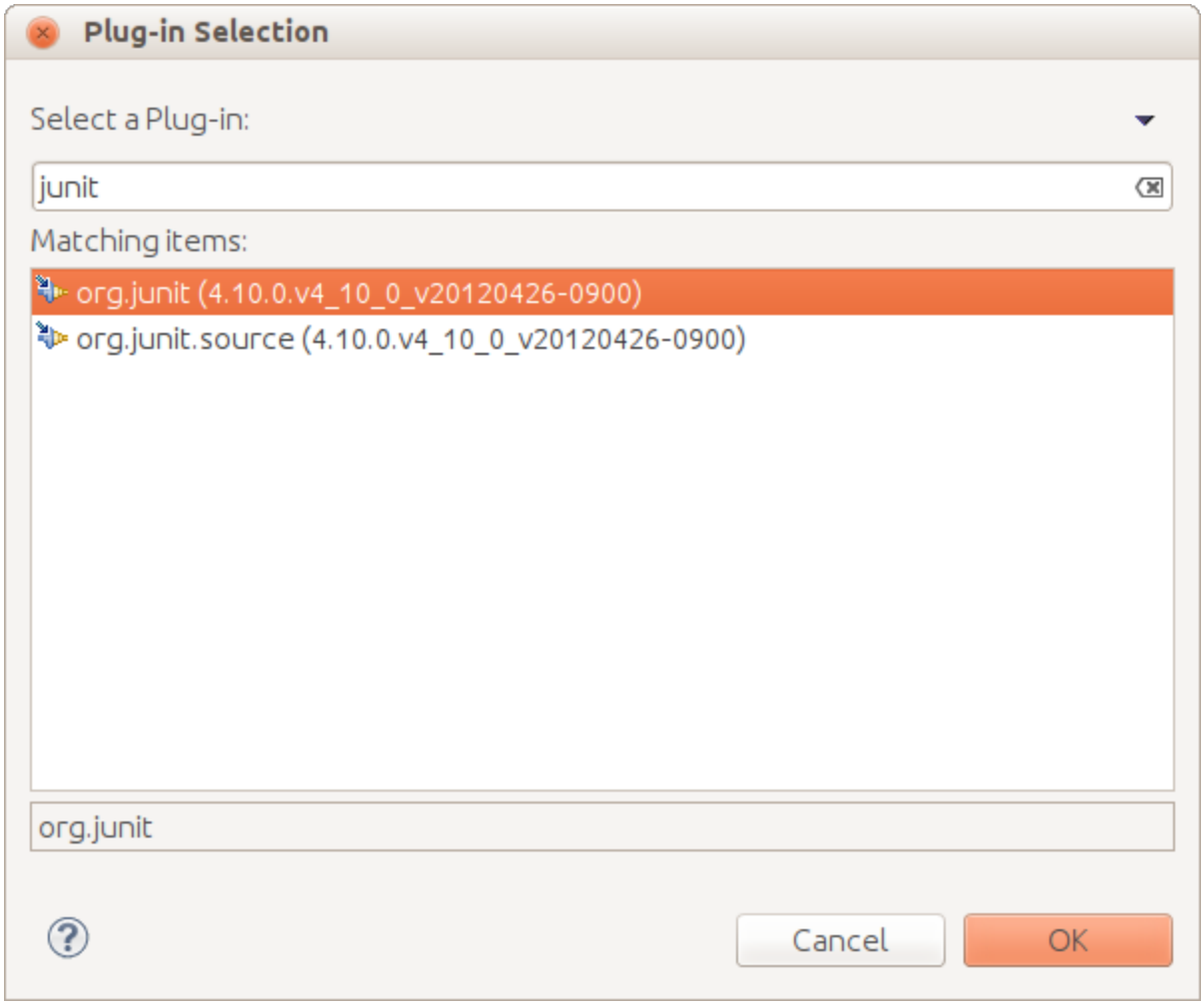
Click “Finish”

20. 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:



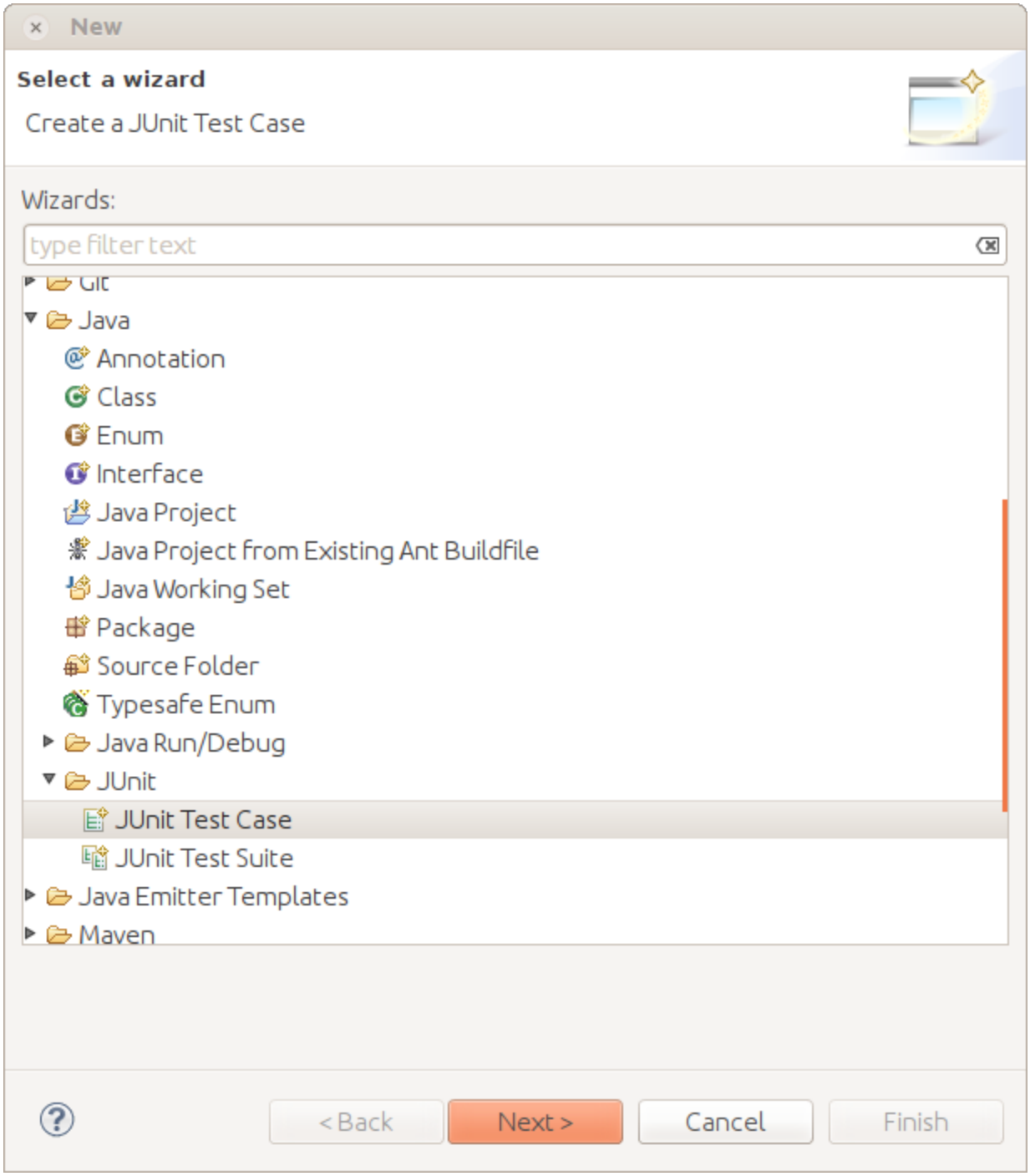
Click “Add...”

21. Enter “junit” in the search field:



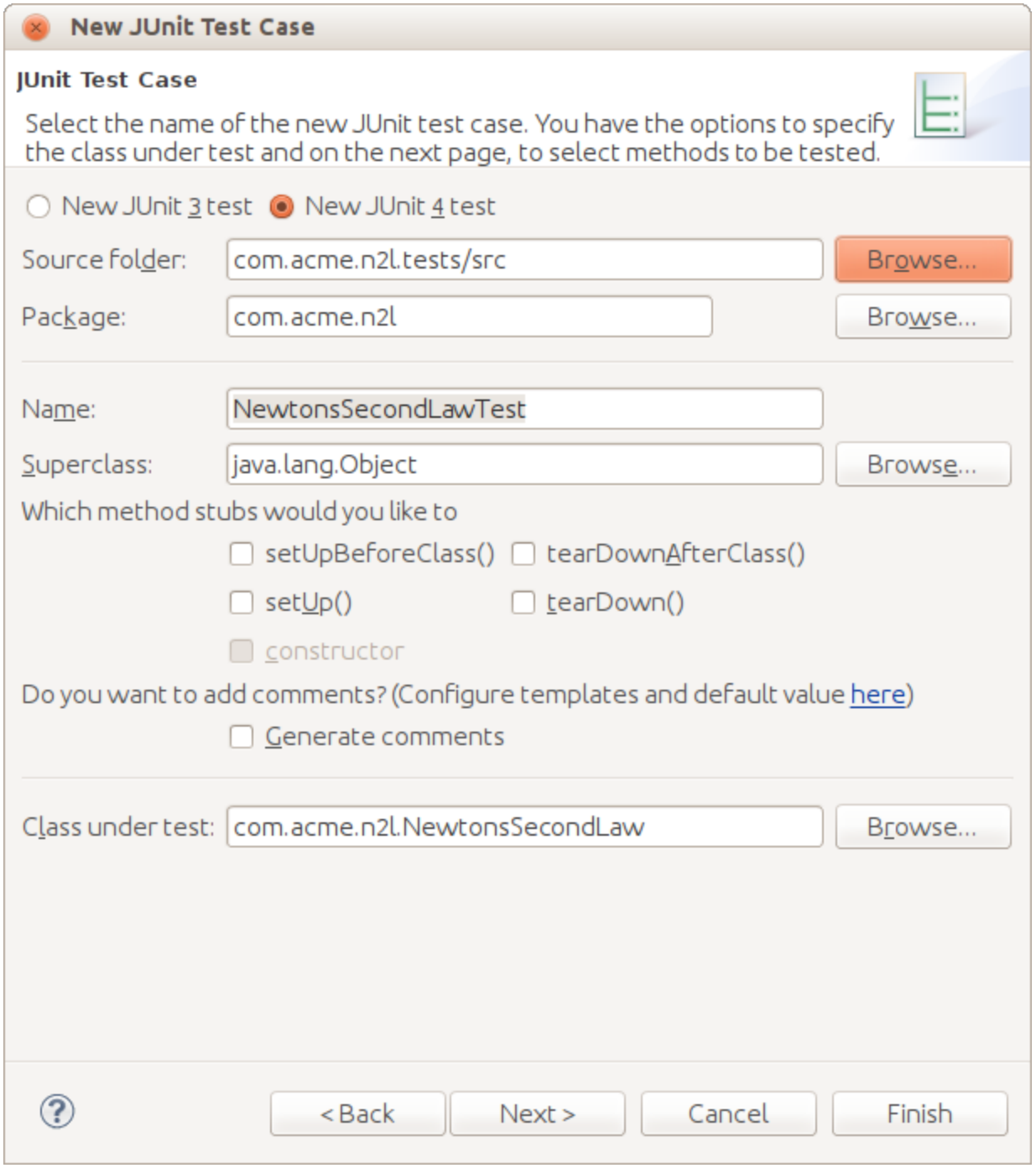
Select the `org.junit` plugin and click “OK”

22. Now we create a unit test class in our fragment. Find and right click on our `NewtonsSecondLaw.java` file and choose “New -> Other...” from the context menu.
23. Select Java -> JUnit -> JUnit Test Case from the dialog:



Click “Next”

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



Click “Finish”

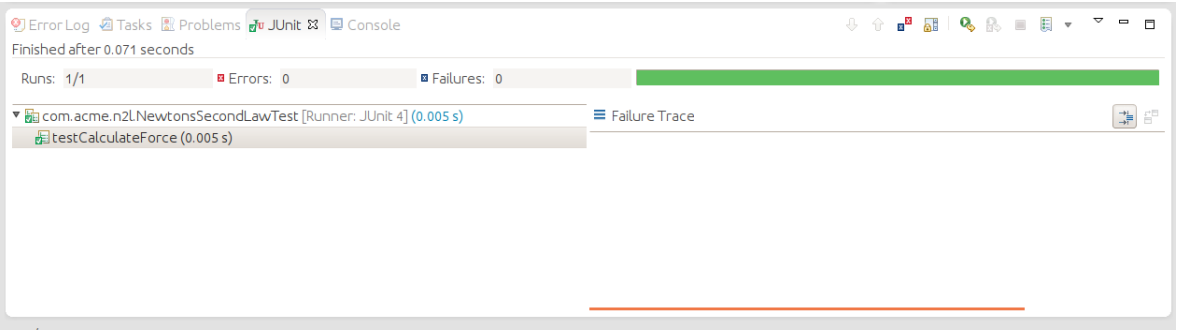
25. Now we need to replace the template test code with our own. Add the following:

```
@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/s2 acceleration and asks our N2L class to calculate the force. We then assert that the calculated force is 2500 Newtons (+/- 0.0001).

26. 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:



27. You have now completed and tested your first simple UOMo project.

OK, so what....

Ok 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. 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! Sweet!