Developers' Guide

Table of Contents

1. Developers' Guide . . . 1
   1.1. Other Guides . . . 1
2. Using an IDE . . . 2
   2.1. Developing using IntelliJ IDEA . . . 2
   2.2. Developing using Eclipse . . . 32
3. Code and File Templates . . . 38
   3.1. Download . . . 38
   3.2. Installation . . . 38
   3.3. Usage . . . 39
4. Building Apache Isis . . . 40
   4.1. Git . . . 40
   4.2. Installing Java . . . 42
   4.3. Installing Maven . . . 43
   4.4. Building all of Apache Isis . . . 44
   4.5. Checking for Vulnerabilities . . . 44
   4.6. Checking for use of internal JDK APIs . . . 44
5. AsciiDoc Documentation . . . 46
   5.1. Where to find the Docs . . . 46
   5.2. Naming Conventions . . . 46
   5.3. Writing the docs . . . 47
   5.4. Build and Review (using Maven) . . . 47
   5.5. Instant Rebuild (using Ruby) . . . 47
   5.6. Publish procedure . . . 48
6. Contributing . . . 50
   6.1. Recommended Workflow (github) . . . 50
   6.2. Alternative Workflow (JIRA patches) . . . 51
   6.3. Setting up your fork/clone . . . 52
   6.4. Commit messages . . . 52
   6.5. Creating the patch file . . . 53
   6.6. Sample Contribution Workflow . . . 53
   6.7. If your pull request is accepted . . . 55
7. Appendix: Git Cookbook . . . 57
   7.1. Modifying existing files . . . 57
   7.2. Adding new files . . . 57
   7.3. Deleting files . . . 57
   7.4. Renaming or moving files . . . 58
   7.5. Common Workflows . . . 58
   7.6. Backing up a local branch . . . 58

7.7. Quick change: stashing changes . . . 58
7.8. Ignoring files . . . 59
7.9. More advanced use cases . . . 59
7.10. If you've accidentally worked on `master` branch . . . 60
7.11. If you've forgotten to prefix your commits (but not pushed) . . . 60
8. Appendix: Asciidoc Templates . . . 62
8.1. Callouts . . . 62
8.2. TODO notes . . . 62
8.3. Xref to Guides . . . 62
8.4. Link to Isis Addons . . . 77
8.5. Source code . . . 78
8.6. Images . . . 78
8.7. YouTube (screencasts) . . . 79
8.8. Tables . . . 79
8.9. Misc. . . . 79
9. Appendix: Project Lombok . . . 81
9.1. Future thoughts . . . 81
10. Appendix: AgileJ . . . 82

Chapter 1. Developers' Guide

This developers' guide is for:

- programmers who want to just use Apache Isis to build applications, and want help setting up their development environment or to build their code from the command line (eg to execute within a continuous integration server such as Jenkins)
- programmers who want to contribute back patches (bug fixes, new features) either to the codebase or the framework's documentation
- committers of Apache Isis itself who want guidance on release process, publishing documents and other related procedures.

1.1. Other Guides

Apache Isis documentation is broken out into a number of user, reference and "supporting procedures" guides.

The user guides available are:

- Fundamentals
- Wicket viewer
- Restful Objects viewer
- Security
- Testing
- Beyond the Basics

The reference guides are:

- Annotations
- Domain Services
- Configuration Properties
- Classes, Methods and Schema
- Apache Isis Maven plugin

The remaining guides are:

- Developers' Guide (this guide)
- Committers' Guide (release procedures and related practices)

Chapter 2. Using an IDE

The vast majority of Java developers use an IDE to assist with developing their code, and we highly recommend that you do like wise as you develop your Apache Isis applications using an IDE. Apache Isis is built with Maven, and all modern IDEs can import Maven projects.

This chapter shows how to setup and use two of the most popular IDEs, IntelliJ IDEA and Eclipse.

2.1. Developing using IntelliJ IDEA

This material does not constitute an endorsement; JetBrains is not affiliated to Apache Software Foundation in any way.

This section describes how to install and setup JetBrains' IntelliJ IDEA, then how to import an application into IntelliJ and run it.

2.1.1. Installing and Setting up

This section covers installation and setup. These notes relates to IntelliJ Community Edition 14.1.x, with screenshots taken for Windows.

Download and Install

Download latest version of IntelliJ Community Edition, and install:

Start the wizard, click through the welcome page:

Figure 1. IntelliJ Installation Wizard - Welcome page

Choose the location to install the IDE:

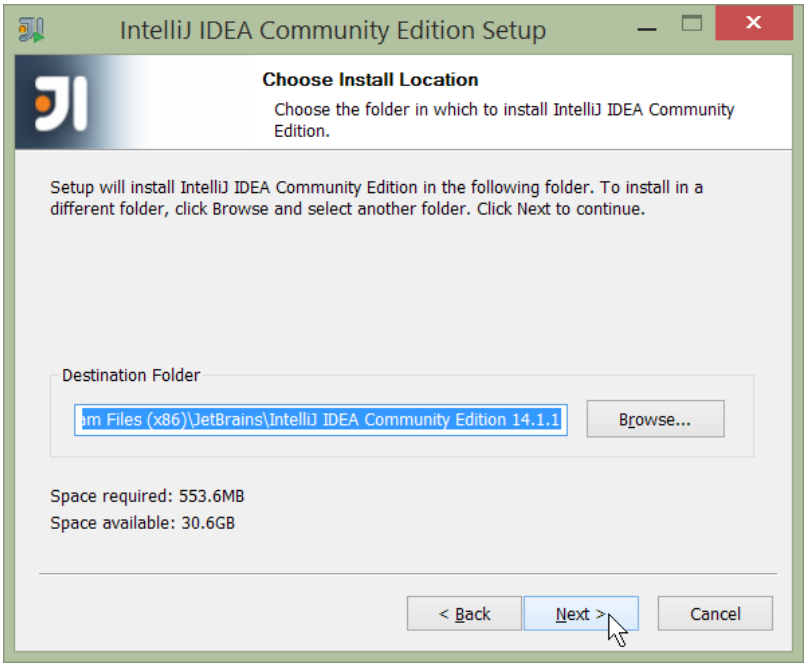


Figure 2. IntelliJ Installation Wizard - Choose Location

Adjust any installation options as you prefer:

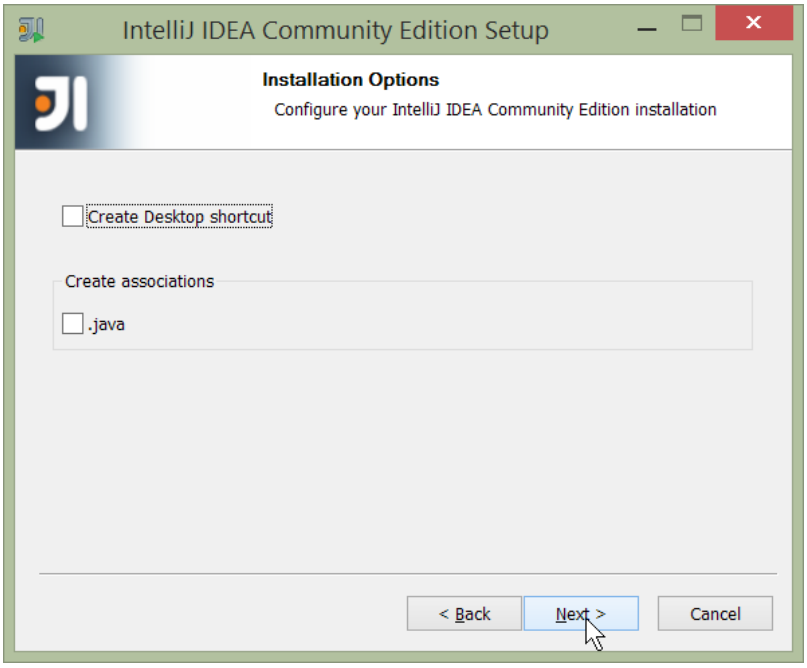


Figure 3. IntelliJ Installation Wizard - Installation Options

and the start menu:

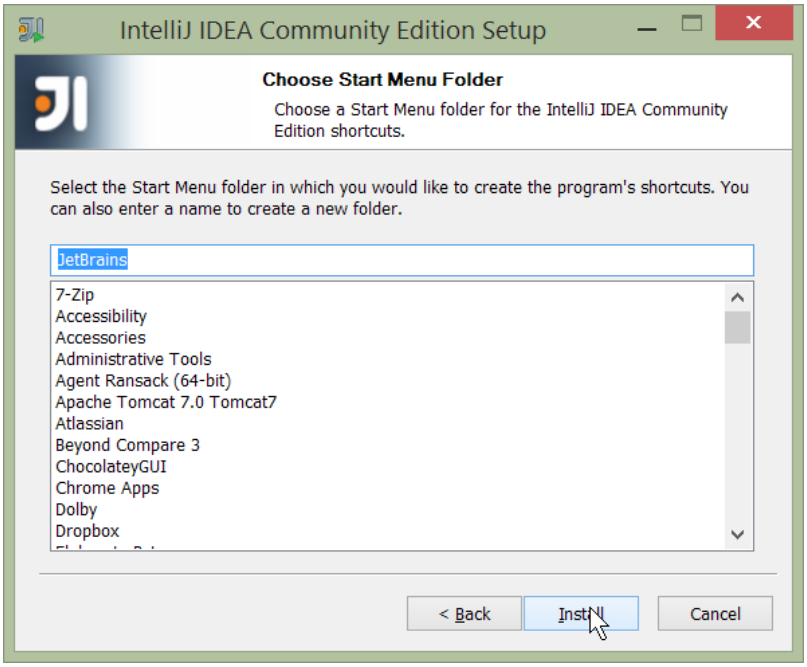


Figure 4. IntelliJ Installation Wizard - Start Menu Folder

and finish up the wizard:

Figure 5. IntelliJ Installation Wizard - Completing the Wizard

Later on we'll specify the Apache Isis/ASF code style settings, so for now select `I do not want to import settings`:

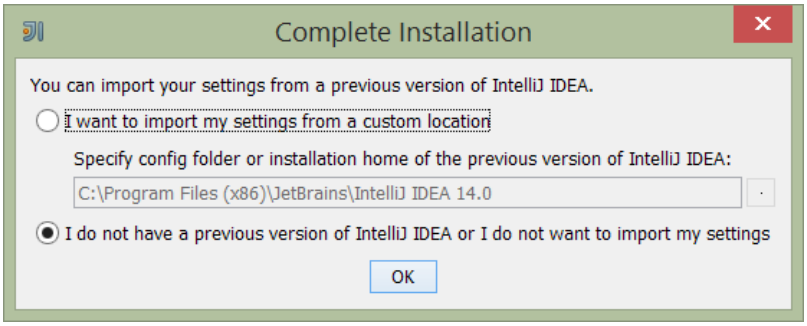


Figure 6. IntelliJ Installation Wizard - Import Settings

Finally, if you are young and trendy, set the UI theme to Darcula:

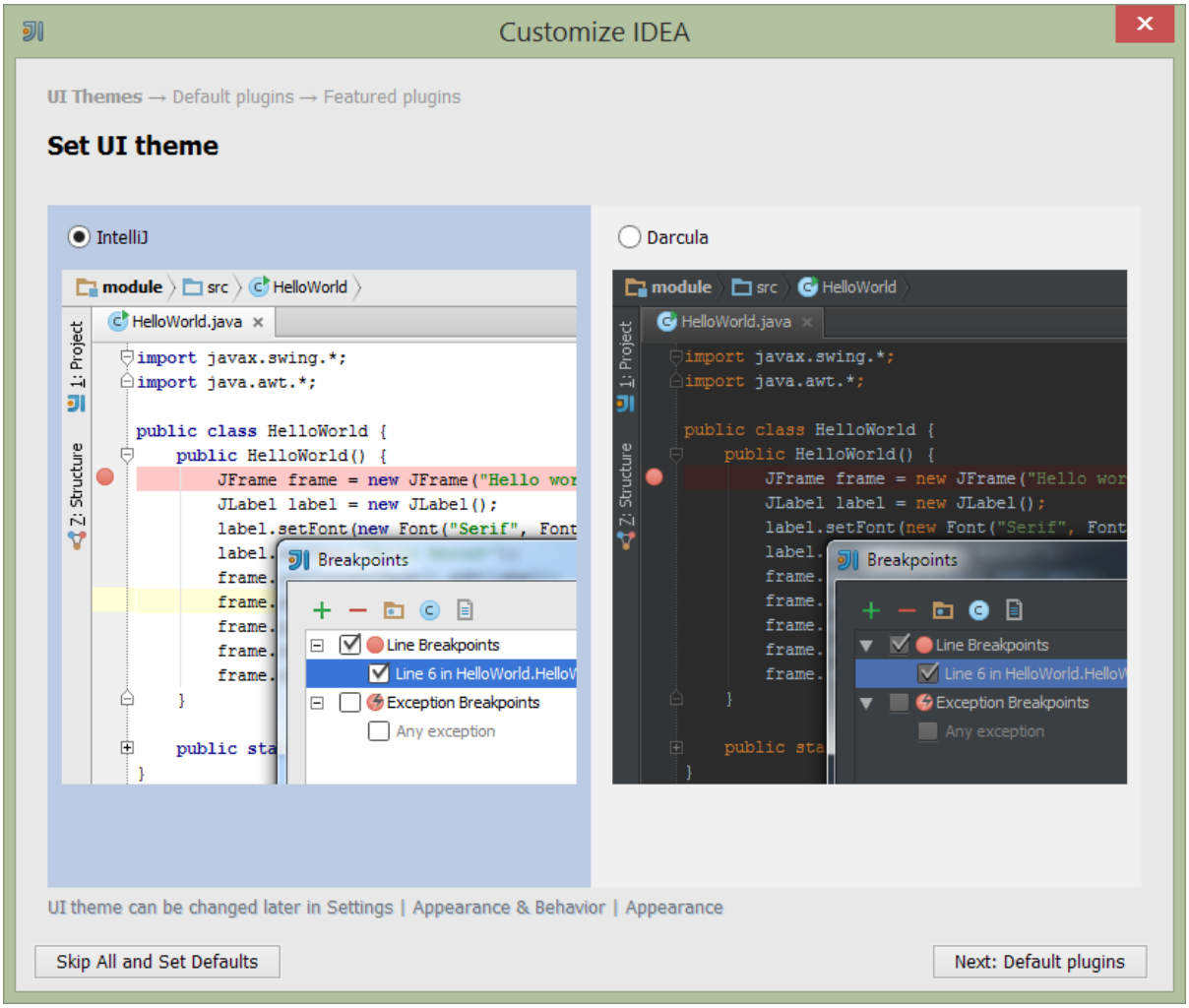


Figure 7. IntelliJ Installation Wizard Set UI Theme

New Project

In IntelliJ a project can contain multiple modules; these need not be physically located together. (If you are previously an Eclipse user, you can think of it as similar to an Eclipse workspace).

Start off by creating a new project:

Figure 8. IntelliJ Create New Project

We want to create a new **Java** project:

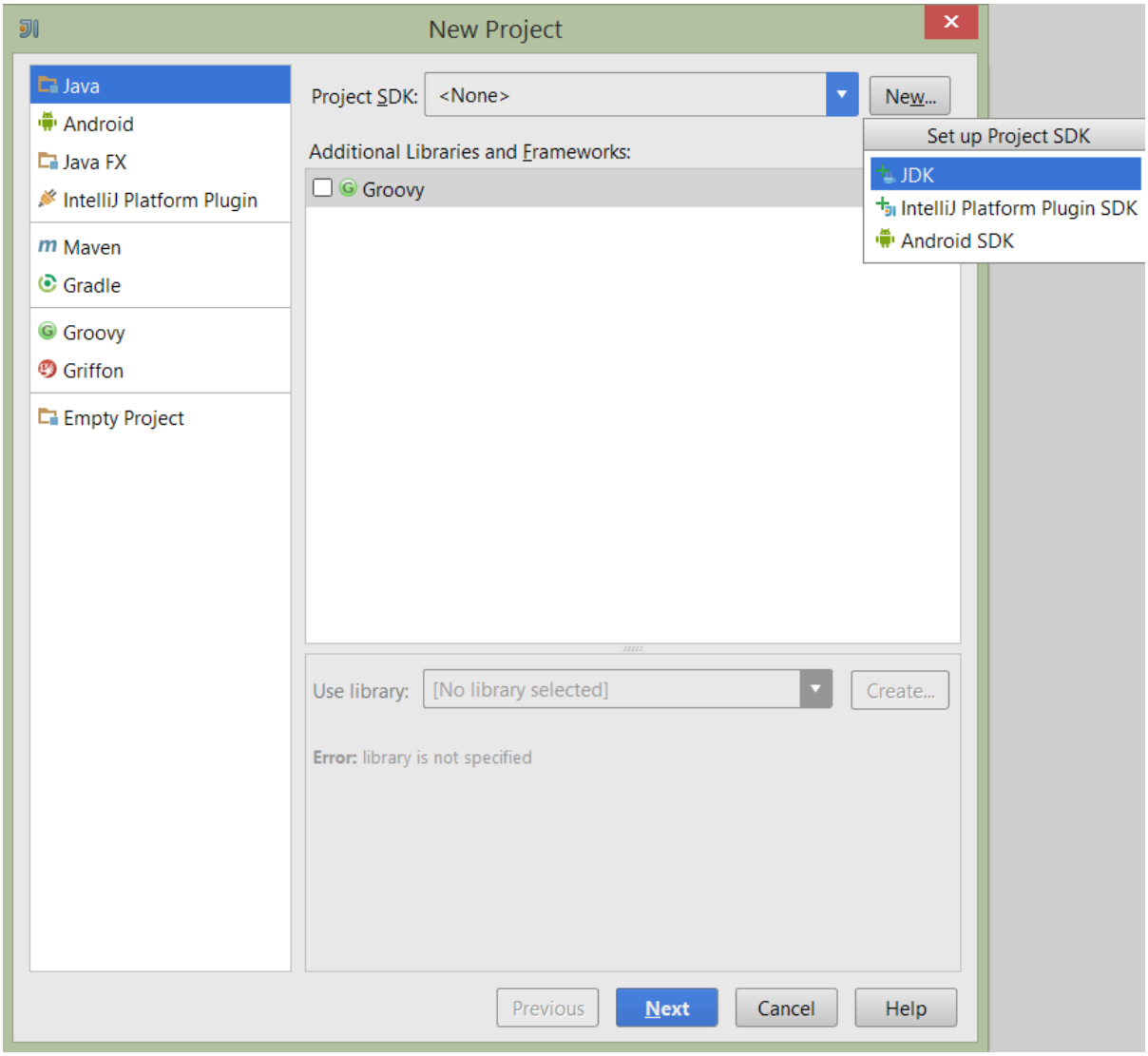


Figure 9. IntelliJ Create New Project - Create a Java project

We therefore need to specify the JDK.

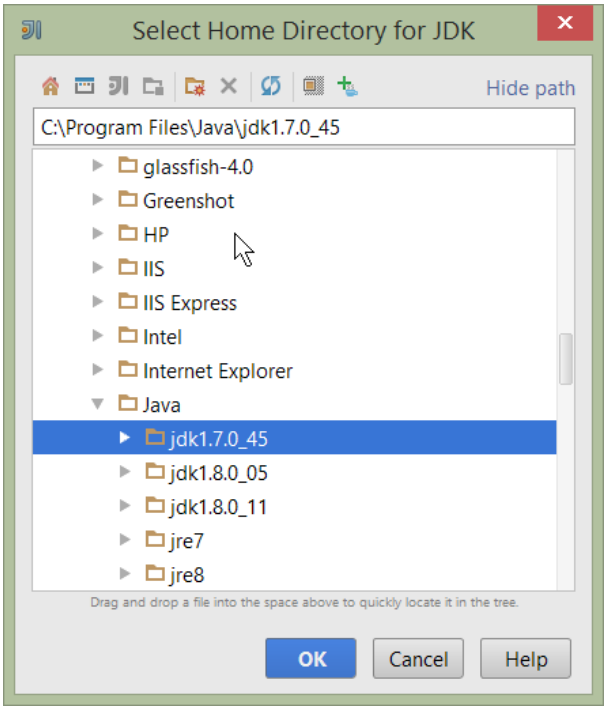


Figure 10. IntelliJ Create New Java Project - Select the JDK

Specify the directory containing the JDK:

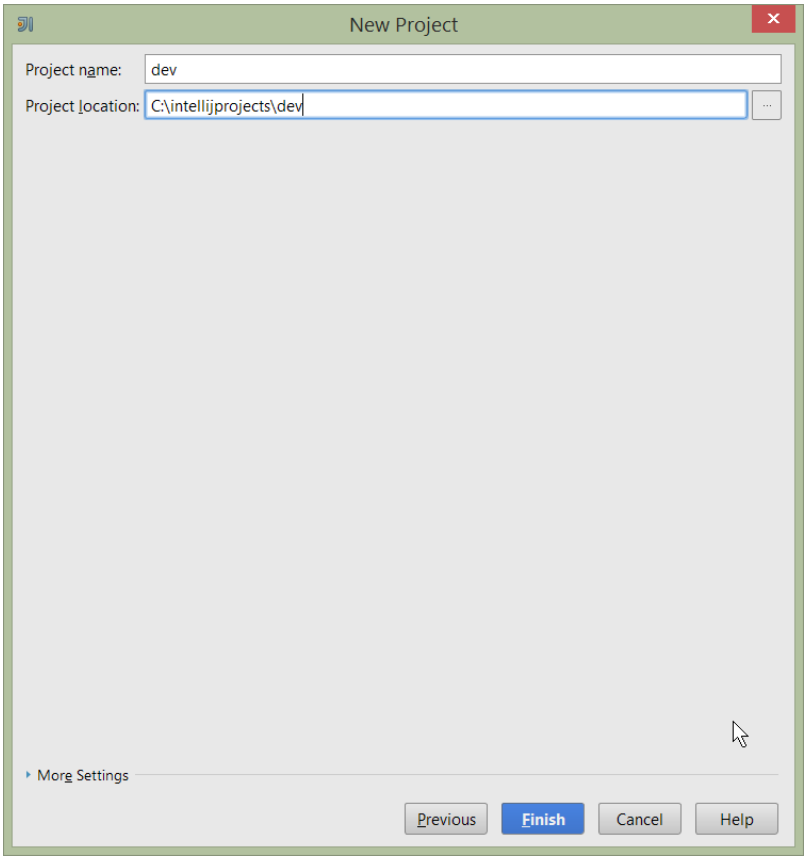


Figure 11. IntelliJ Create New Project - Select the JDK location

Finally allow IntelliJ to create the directory for the new project:

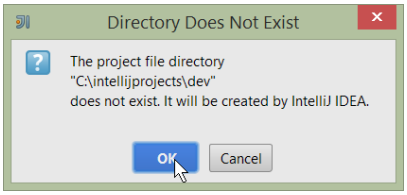


Figure 12. IntelliJ Create New Project

Import Settings

Next we need to configure IntelliJ with ASF/Apache Isis' standard templates and coding conventions. These are bundled as the `settings.jar` JAR file download from the Apache Isis website).

Import using: `File > Import Settings`, and specify the directory that you have downloaded the file to:

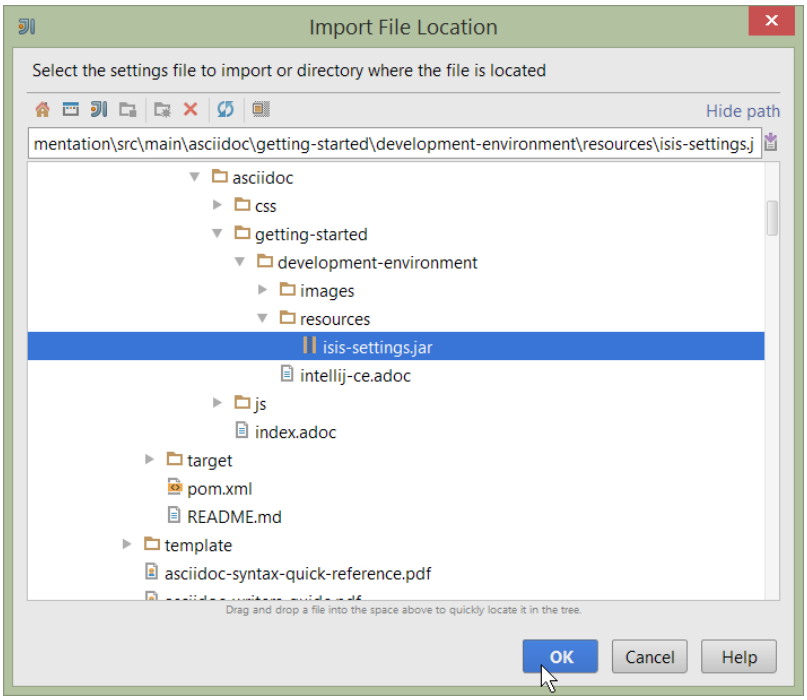


Figure 13. IntelliJ Import Settings - Specify JAR file

Select all the (two) categories of settings available in the JAR file:

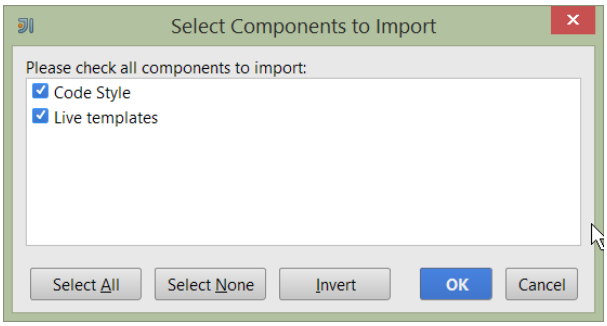


Figure 14. IntelliJ Import Settings - Select all categories

And then restart:

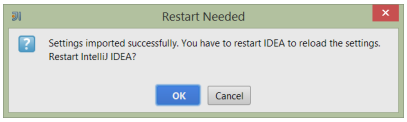


Figure 15. IntelliJ Import Settings - Restart

Other Settings (Compiler)

There are also some other settings that influence the compiler. We highly recommend you set these.

On the **Compiler** Settings page, ensure that `build automatically` is enabled (and optionally `compile independent modules in parallel`):

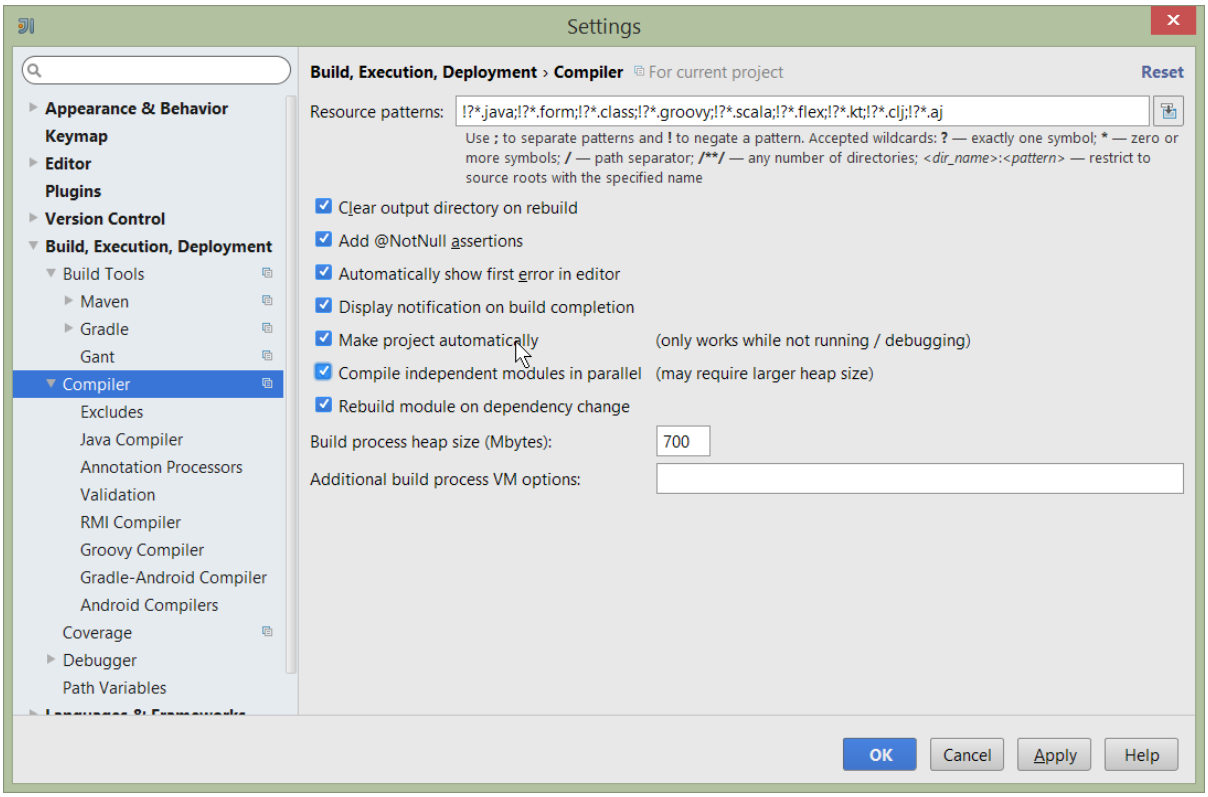


Figure 16. IntelliJ Compiler Settings

On the **Annotation Processors** page, enable and adjust for the 'default' setting:

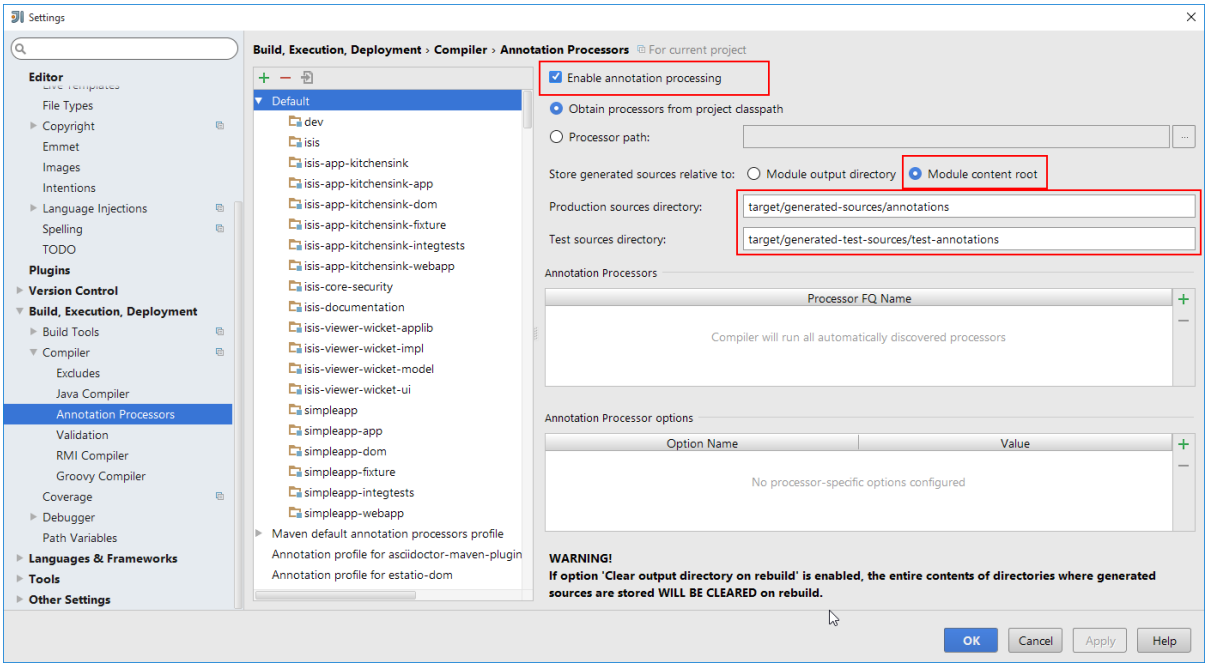


Figure 17. IntelliJ Annotation Processor Settings

This setting enables the generation of the `Q*` classes for DataNucleus type-safe queries, as well as being required for frameworks such as Project Lombok.

IntelliJ may also have inferred these settings for specific projects/modules when importing; review the list on the left to see if the default is overridden and fix/delete as required.

Other Settings (Maven)

There are also some other settings for Maven that we recommend you adjust (though these are less critical):

First, specify an up-to-date Maven installation, using `File > Settings` (or `IntelliJ > Preferences` if on MacOS):

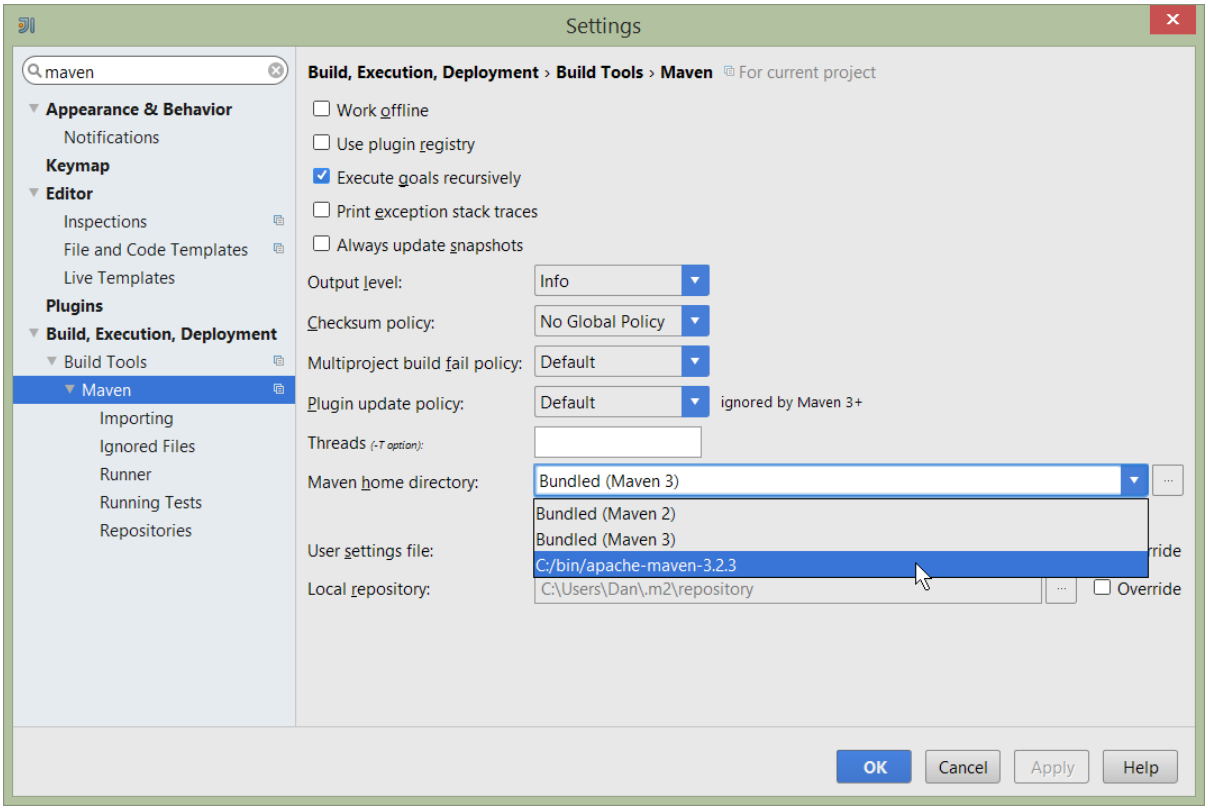


Figure 18. IntelliJ Maven Settings - Installation

Still on the Maven settings page, configure as follows:

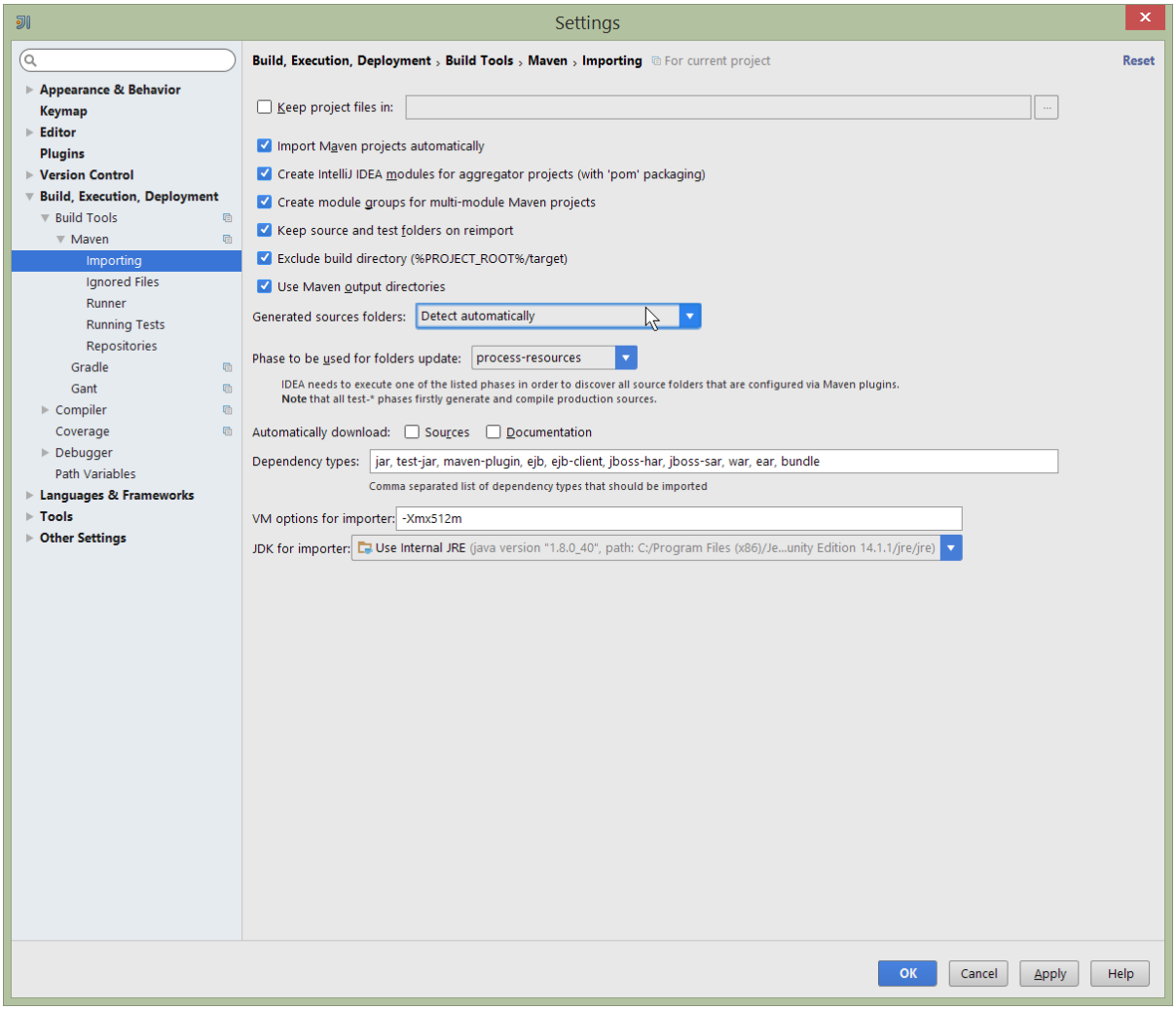


Figure 19. IntelliJ Maven Settings - Configuration

Other Settings (Misc)

These settings are optional but also recommended.

On the **auto import** page, check the `optimize imports on the fly` and `add unambiguous imports on the fly`

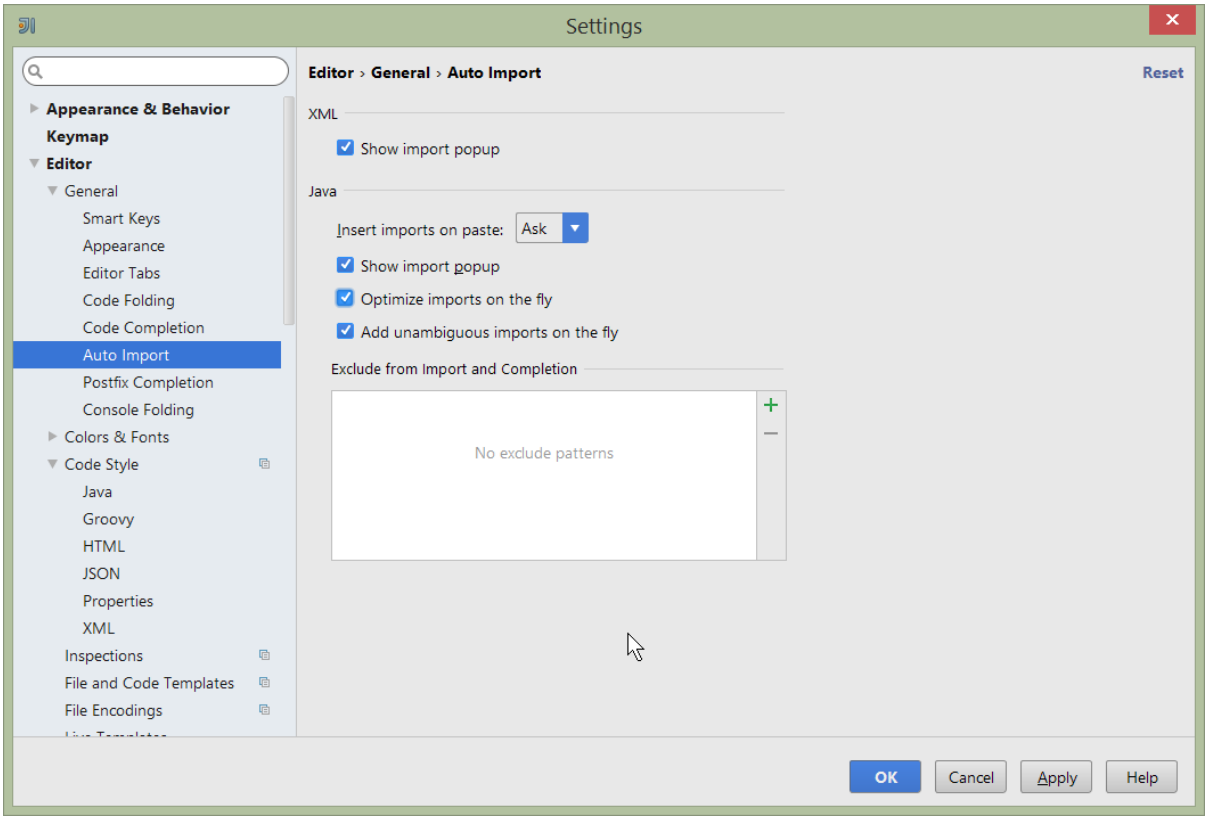


Figure 20. IntelliJ Maven Settings - Auto Import

2.1.2. Importing Maven Modules

Let's load in some actual code! We do this by importing the Maven modules.

First up, open up the Maven tool window (`View > Tool Windows > Maven Projects`). You can then use the 'plus' button to add Maven modules. In the screenshot you can see we've loaded in Apache Isis core; the modules are listed in the *Maven Projects* window and corresponding (IntelliJ) modules are shown in the *Projects* window:

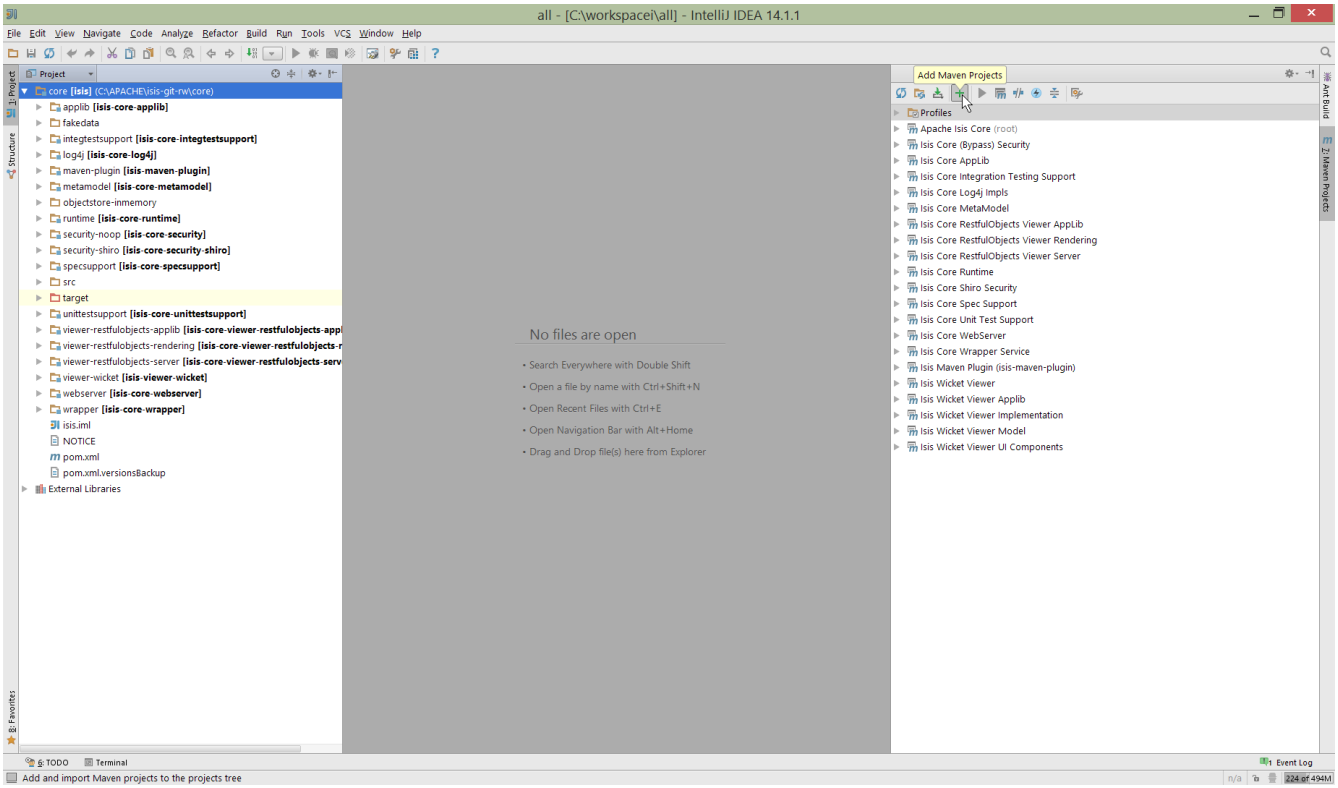


Figure 21. IntelliJ Maven Module Management - Importing Maven modules

We can then import another module (from some other directory). For example, here we are importing the Isis Addons' todoapp example:

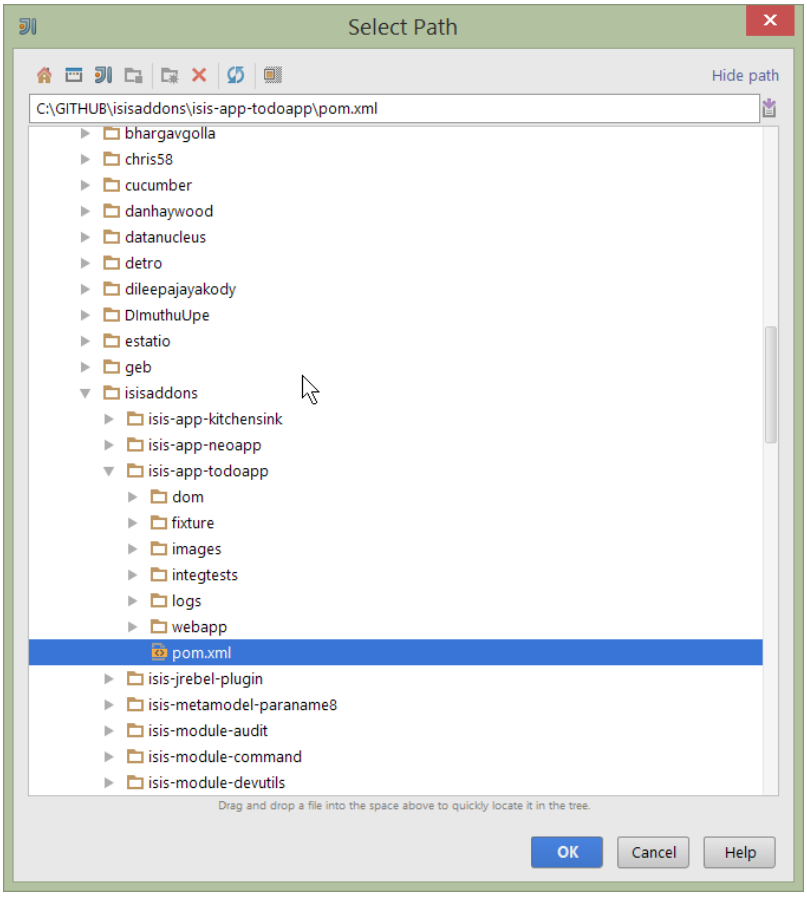


Figure 22. IntelliJ Maven Module Management - Importing another Module

You should then see the new Maven module loaded in the *Projects* window and also the *Maven*

Projects window:

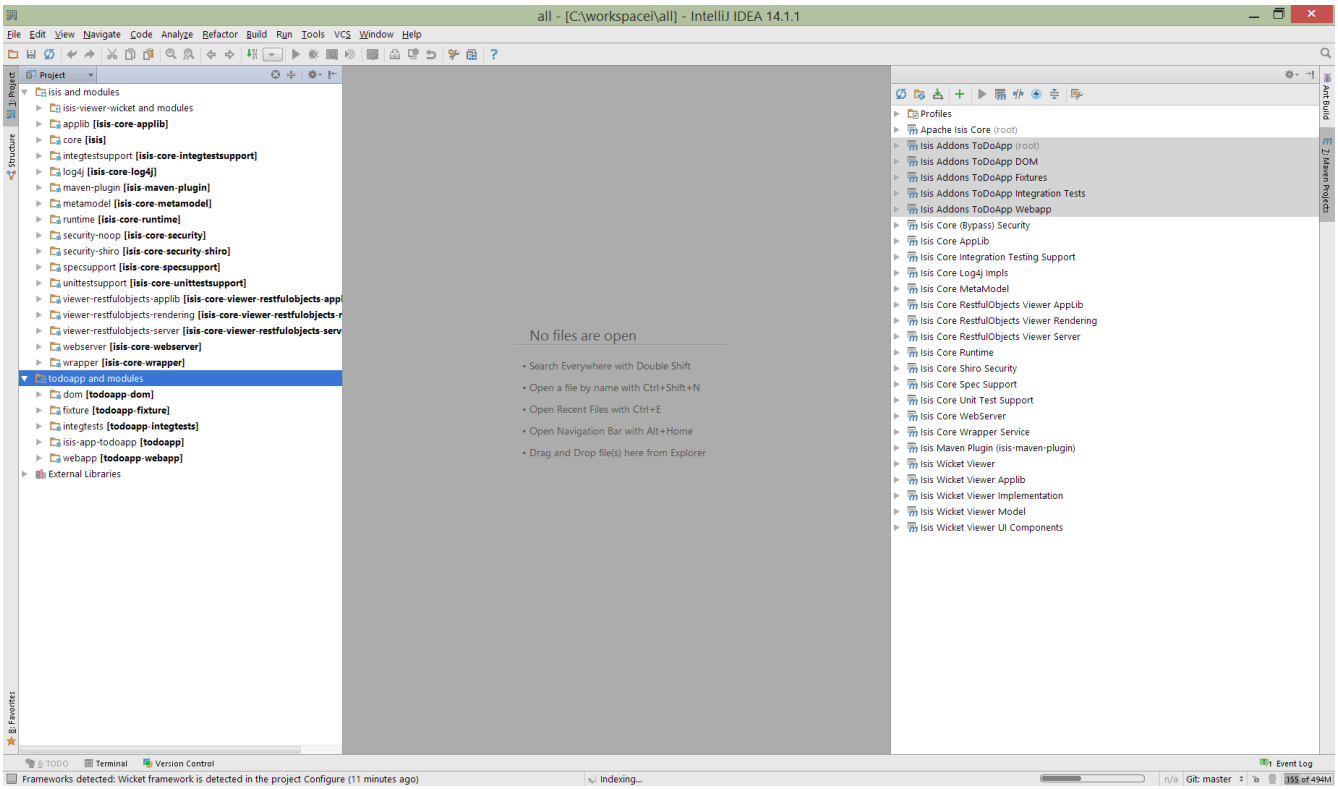


Figure 23. IntelliJ Maven Module Management -

If any dependencies are already loaded in the project, then IntelliJ will automatically update the CLASSPATH to resolve to locally held modules (rather from `.m2/repository` folder). So, for example (assuming that the `<version>` is correct, of course), the Isis todoapp will have local dependencies on the Apache Isis core.

You can press F4 (or use `File > Project Structure`) to see the resolved classpath for any of the modules loaded into the project.

If you want to focus on one set of code (eg the Isis todoapp but not Apache Isis core) then you *could* remove the module; but better is to ignore those modules. This will remove from the the *Projects* window but keep them available in the *Maven Projects* window for when you next want to work on them:

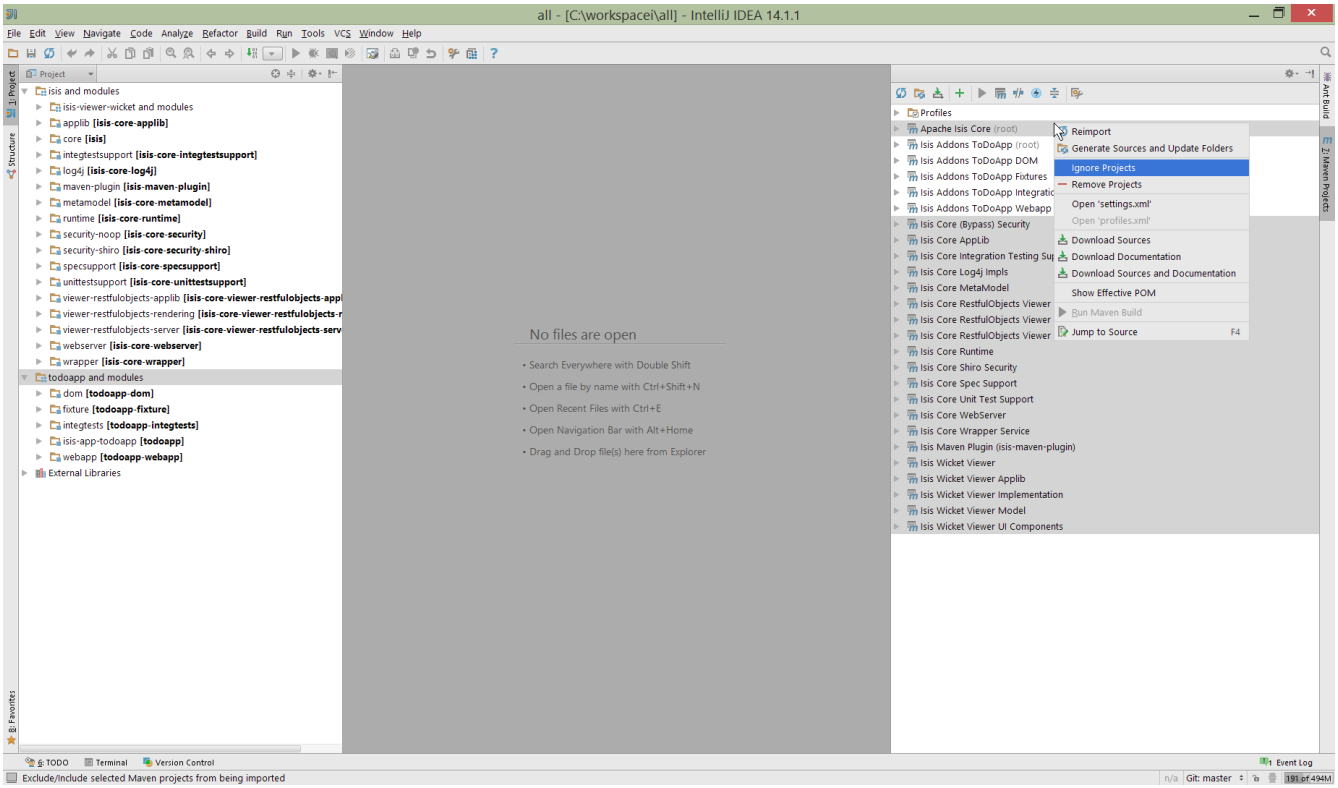


Figure 24. IntelliJ Maven Module Management - Ignoring Modules

Confirm that it's ok to ignore these modules:

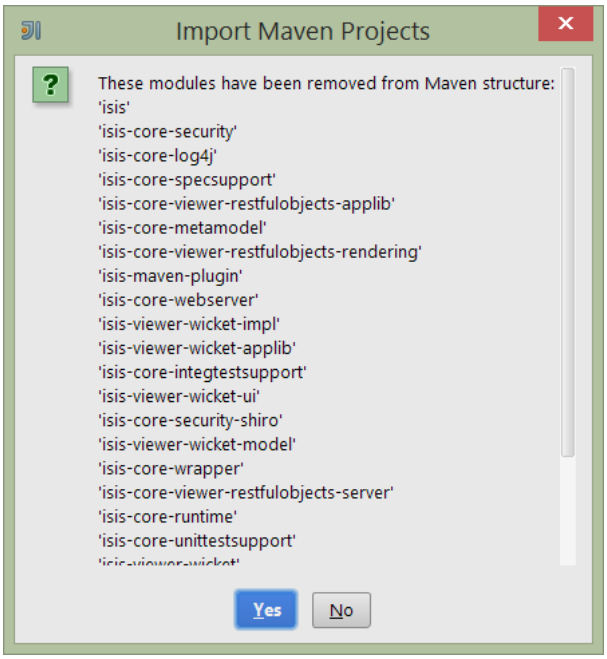


Figure 25. IntelliJ Maven Module Management - Ignoring Modules (ctd)

All being well you should see that the *Projects* window now only contains the code you are working on. Its classpath dependencies will be adjusted (eg to resolve to Apache Isis core from `.m2/repository`):

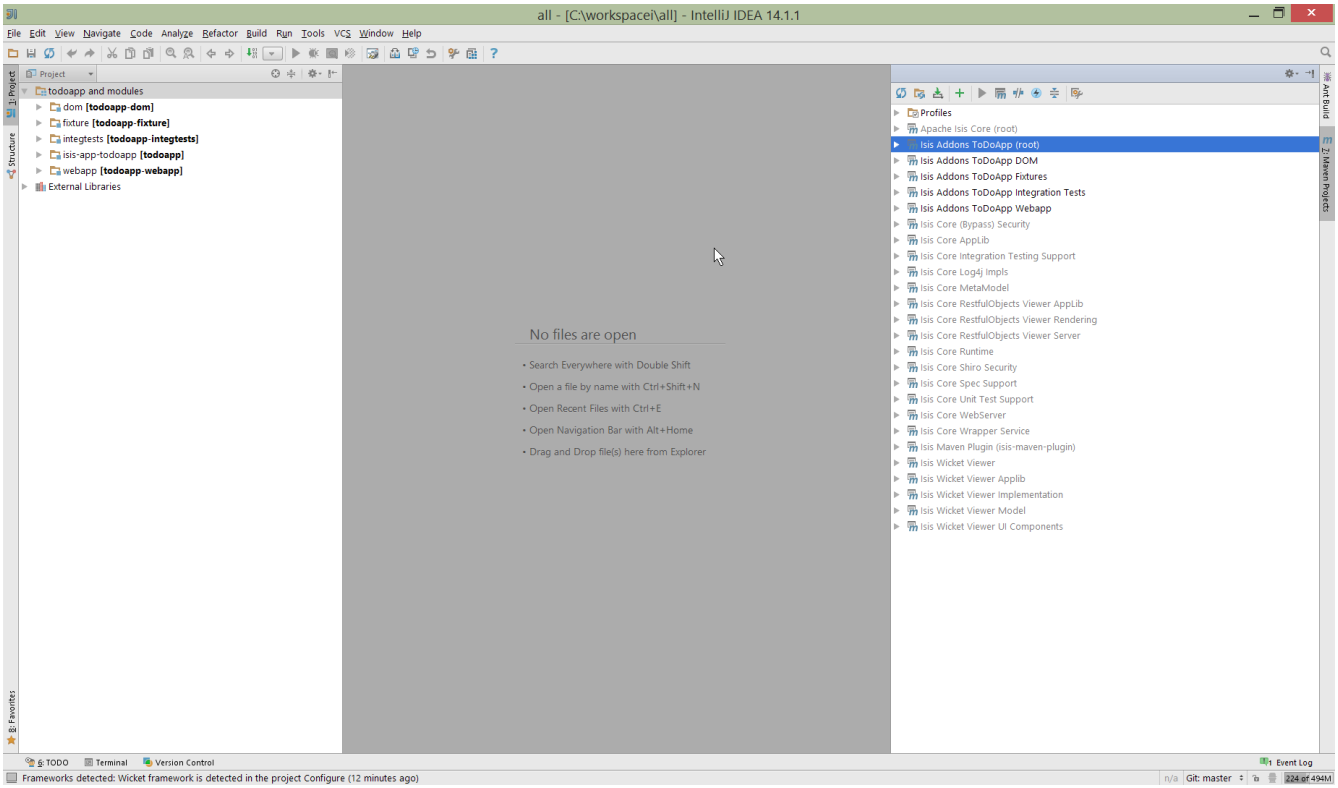


Figure 26. IntelliJ Maven Module Management - Updated Projects Window

2.1.3. Running

Let's see how to run both the app and the tests.

Running the App

Once you've imported your Isis application, we should run it. We do this by creating a Run configuration, using `Run > Edit Configurations`.

Set up the details as follows:

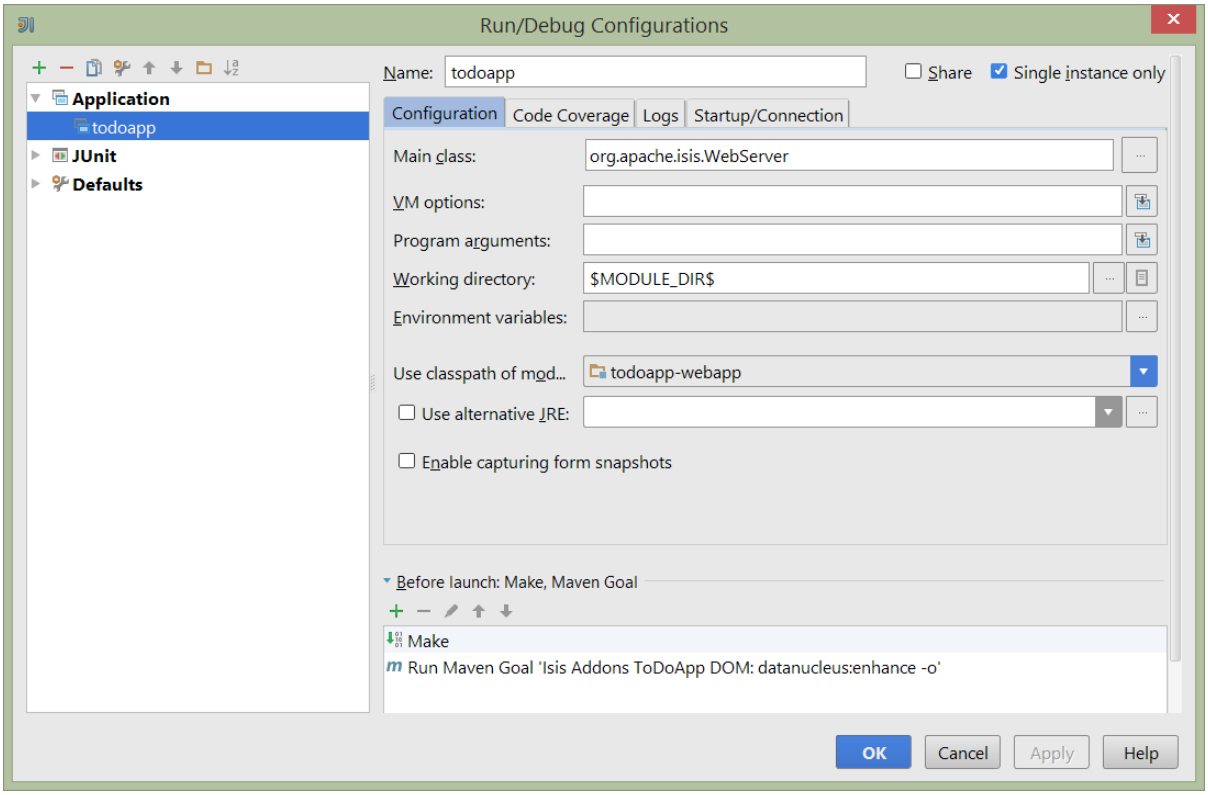


Figure 27. IntelliJ Running the App - Run Configuration

We specify the `Main class` to be `org.apache.isis.WebServer`; this is a wrapper around Jetty. It's possible to pass program arguments to this (eg to automatically install fixtures), but for now leave this blank.

Also note that `Use classpath of module` is the webapp module for your app, and that the `working directory` is `$MODULE_DIR$`.

Next, and most importantly, configure the DataNucleus enhancer to run for your `dom` goal. This can be done by defining a Maven goal to run before the app:

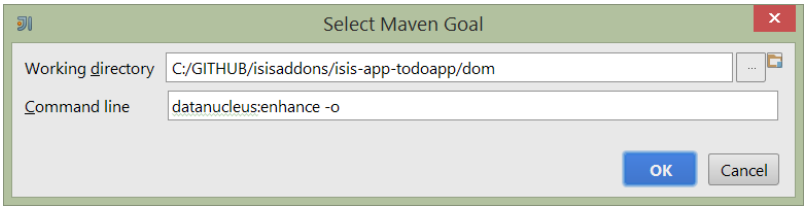


Figure 28. IntelliJ Running the App - Datanucleus Enhancer Goal

The `-o` flag in the goal means run off-line; this will run faster.

if you forget to set up the enhancer goal, or don't run it on the correct (dom) module, then you will get all sorts of errors when you startup. These usually manifest themselves as class cast exception in DataNucleus.

You should now be able to run the app using `Run > Run Configuration`. The same configuration can also be used to debug the app if you so need.

Running the Unit Tests

The easiest way to run the unit tests is just to right click on the `dom` module in the *Project Window*, and choose run unit tests. Hopefully your tests will pass (!).

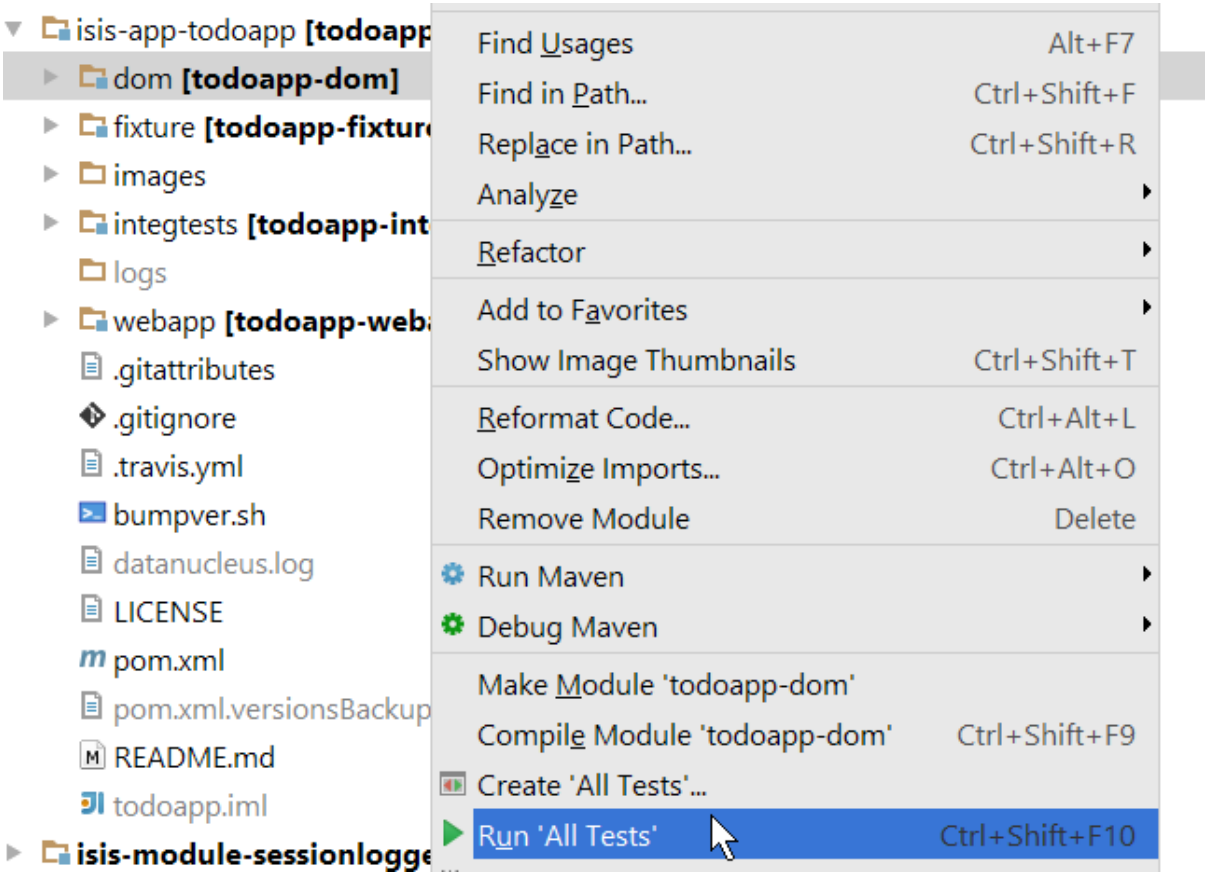


Figure 29. IntelliJ Running the App - Unit Tests Run Configuration

As a side-effect, this will create a run configuration, very similar to the one we manually created for the main app:

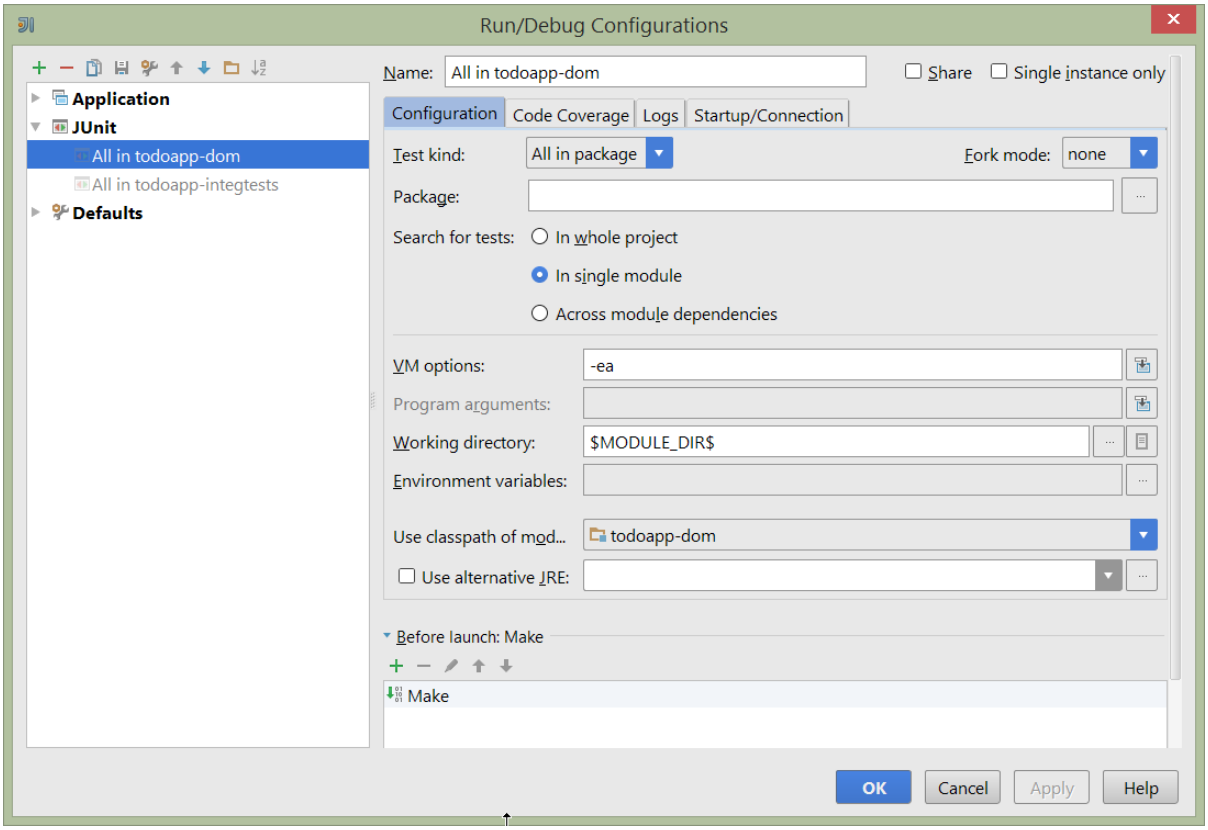


Figure 30. IntelliJ Running the App - Unit Tests Run Configuration

Thereafter, you should run units by selecting this configuration (if you use the right click approach you'll end up with lots of run configurations, all similar).

Running the Integration Tests

Integration tests can be run in the same way as unit tests, however the `dom` module must also have been enhanced.

One approach is to initially run the tests use the right click on the `integtests` module; the tests will fail because the code won't have been enhanced, but we can then go and update the run configuration to run the datanucleus enhancer goal (same as when running the application):

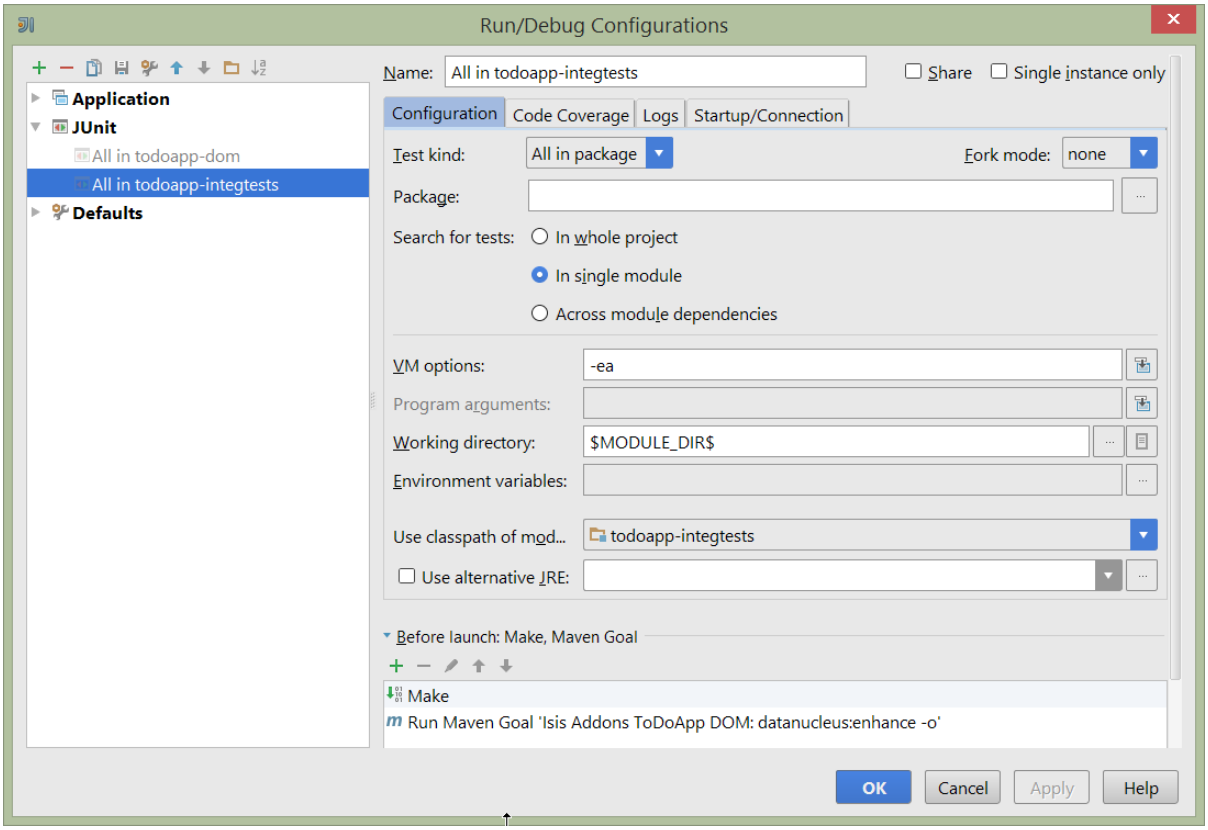


Figure 31. IntelliJ Running the App - Integration Tests Run Configuration

2.1.4. Hints and Tips

Keyboard Cheat Sheets

You can download 1-page PDFs cheat sheets for IntelliJ's keyboard shortcuts: * for Windows * for MacOS

Probably the most important shortcut on them is for `Find Action`: - `ctrl-shift-A` on Windows - `cmd-shift-A` on MacOS.

This will let you search for any action just by typing its name.

Switch between Tools & Editors

The Tool Windows are the views around the editor (to left, bottom and right). It's possible to move these around to your preferred locations.

- Use `alt-1` through `alt-9` (or `cmd-1` through `alt-9`) to select the tool windows
  - Press it twice and the tool window will hide itself; so can use to toggle
- If in the *Project Window* (say) and hit enter on a file, then it will be shown in the editor, but (conveniently) the focus remains in the tool window. To switch to the editor, just press `Esc`.
  - If in the *Terminal Window*, you'll need to press `Shift-Esc`.
- If on the editor and want to locate the file in (say) the *Project Window*, use `alt-F1`.
- To change the size of any tool window, use `ctrl-shift-arrow`

Using these shortcuts you can easily toggle between the tool windows and the editor, without using the mouse. Peachy!

Navigating Around

For all of the following, you don't need to type every letter, typing "ab" will actually search for "**.a.*b.**".

- to open classes or files or methods that you know the name of:
  - `ctrl-N` to open class
  - `ctrl-shift-N` to open a file
  - (bit fiddly this) `ctrl-shift-alt-N` to search for any symbol.
- open up dialog of recent files: `ctrl-E`
- search for any file: `shift-shift`

Navigating around: * find callers of a method (the call hierarchy): `ctrl-alt-H` * find subclasses or overrides: `ctrl-alt-B` * find superclasses/interface/declaration: `ctrl-B`

Viewing the structure (ie outline) of a class * `ctrl-F12` will pop-up a dialog showing all members ** hit `ctrl-F12` again to also see inherited members

Editing

- Extend selection using `ctrl-W`
  - and contract it down again using `ctrl-shift-W`
- to duplicate a line, it's `ctrl-D`
  - if you have some text selected (or even some lines), it'll actually duplicate the entire selection
- to delete a line, it's `ctrl-X`
- to move a line up or down: `shift-alt-up` and `shift-alt-down`
  - if you have selected several lines, it'll move them all togethe
- `ctrl-shift-J` can be handy for joining lines together
  - just hit enter to split them apart (even in string quotes; IntelliJ will "do the right thing")

Intentions and Code Completion

Massively useful is the "Intentions" popup; IntelliJ tries to guess what you might want to do. You can activate this using `alt-enter`, whenever you see a lightbulb/tooltip in the margin of the current line.

Code completion usually happens whenever you type '.'. You can also use `ctrl-space` to bring these up.

In certain circumstances (eg in methods0) you can also type `ctrl-shift-space` to get a smart list of methods etc that you might want to call. Can be useful.

Last, when invoking a method, use `ctrl-P` to see the parameter types.

Refactoring

Loads of good stuff on the `Refactor` menu; most used are:

- Rename (`shift-F6`)
- Extract
  - method: `ctrl-alt-M`
  - variable: `ctrl-alt-V`
- Inline method/variable: `ctrl-alt-N`
- Change signature

If you can't remember all those shortcuts, just use `ctrl-shift-alt-T` (might want to rebind that to something else!) and get a context-sensitive list of refactorings available for the currently selected object

Plugins

You might want to set up some additional plugins. You can do this using `File > Settings > Plugins` (or equivalently `File > Other Settings > Configure Plugins`).

Recommended are:

- Maven Helper plugin

  More on this below.

- AsciiDoctor plugin

  Useful if you are doing any authoring of documents.

Some others you might like to explore are:

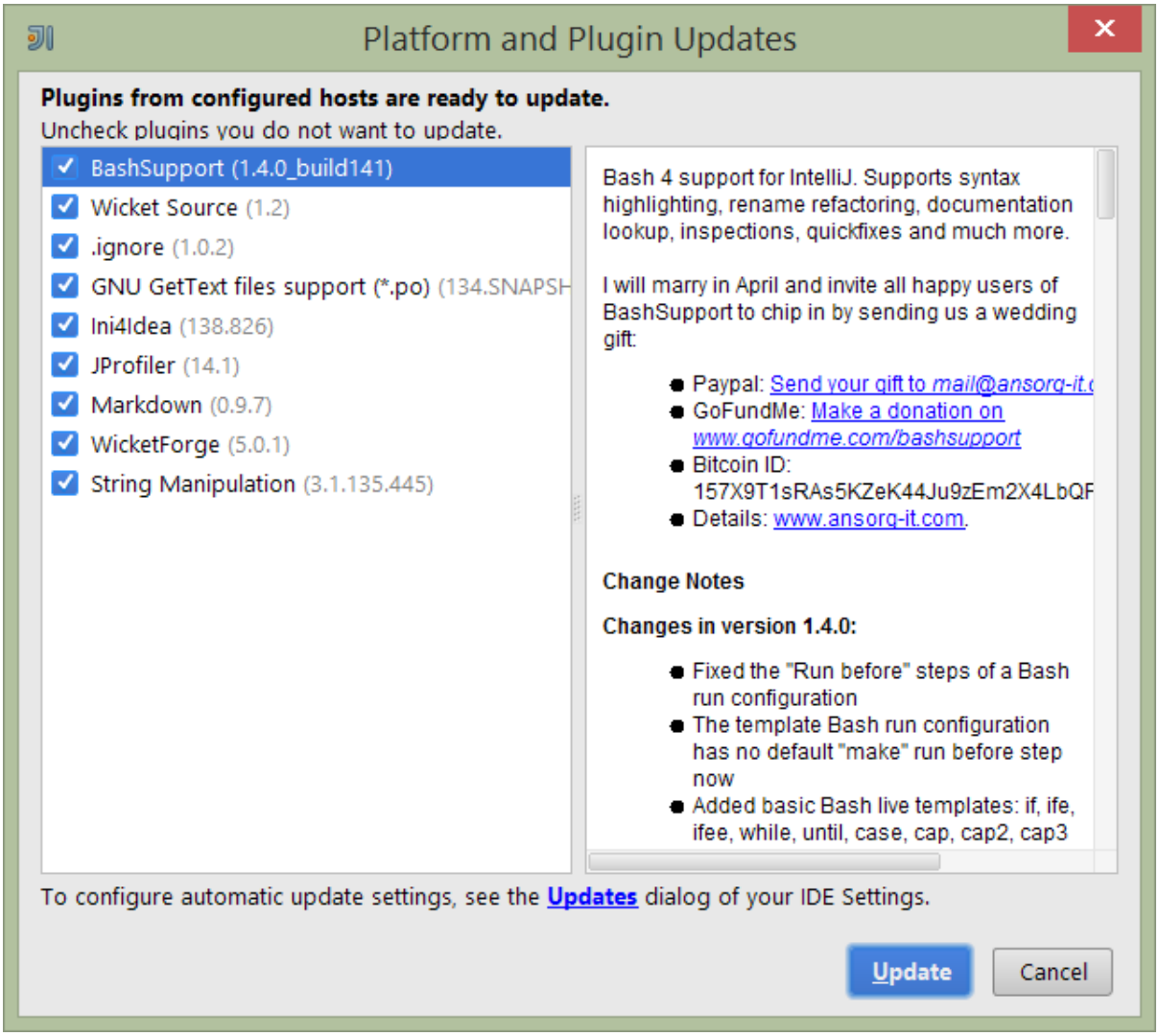
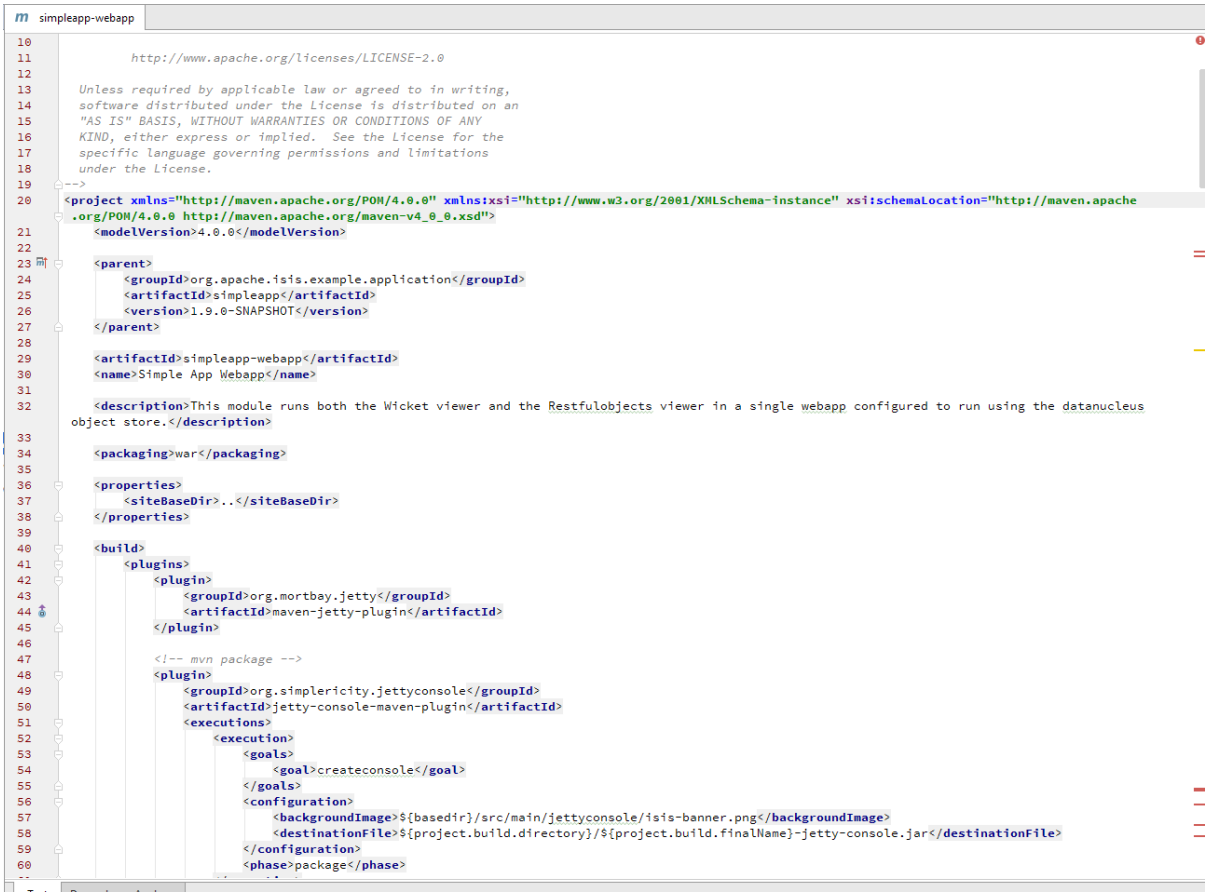


Figure 32. IntelliJ Plugins

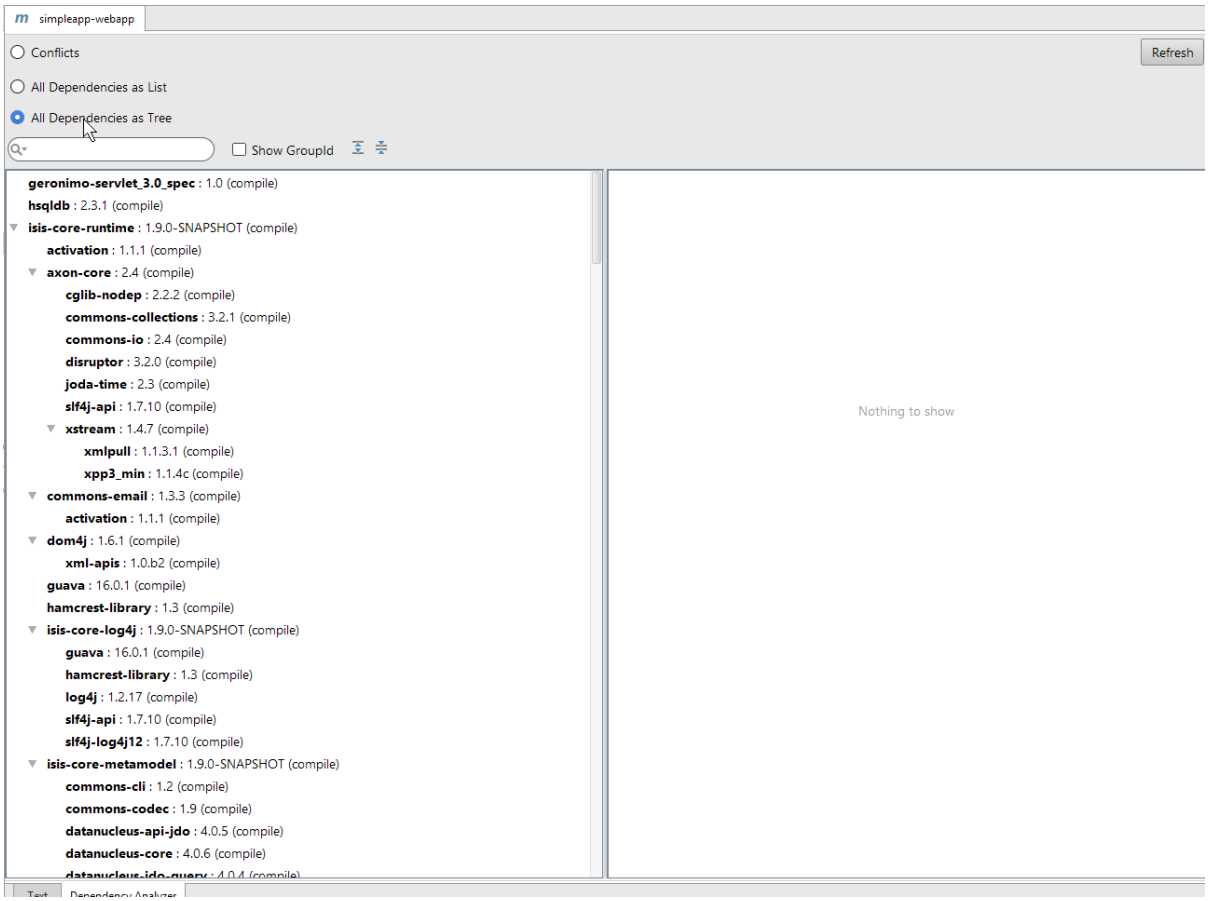
Maven Helper Plugin

This plugin provides a couple of great features. One is better visualization of dependency trees (similar to Eclipse).

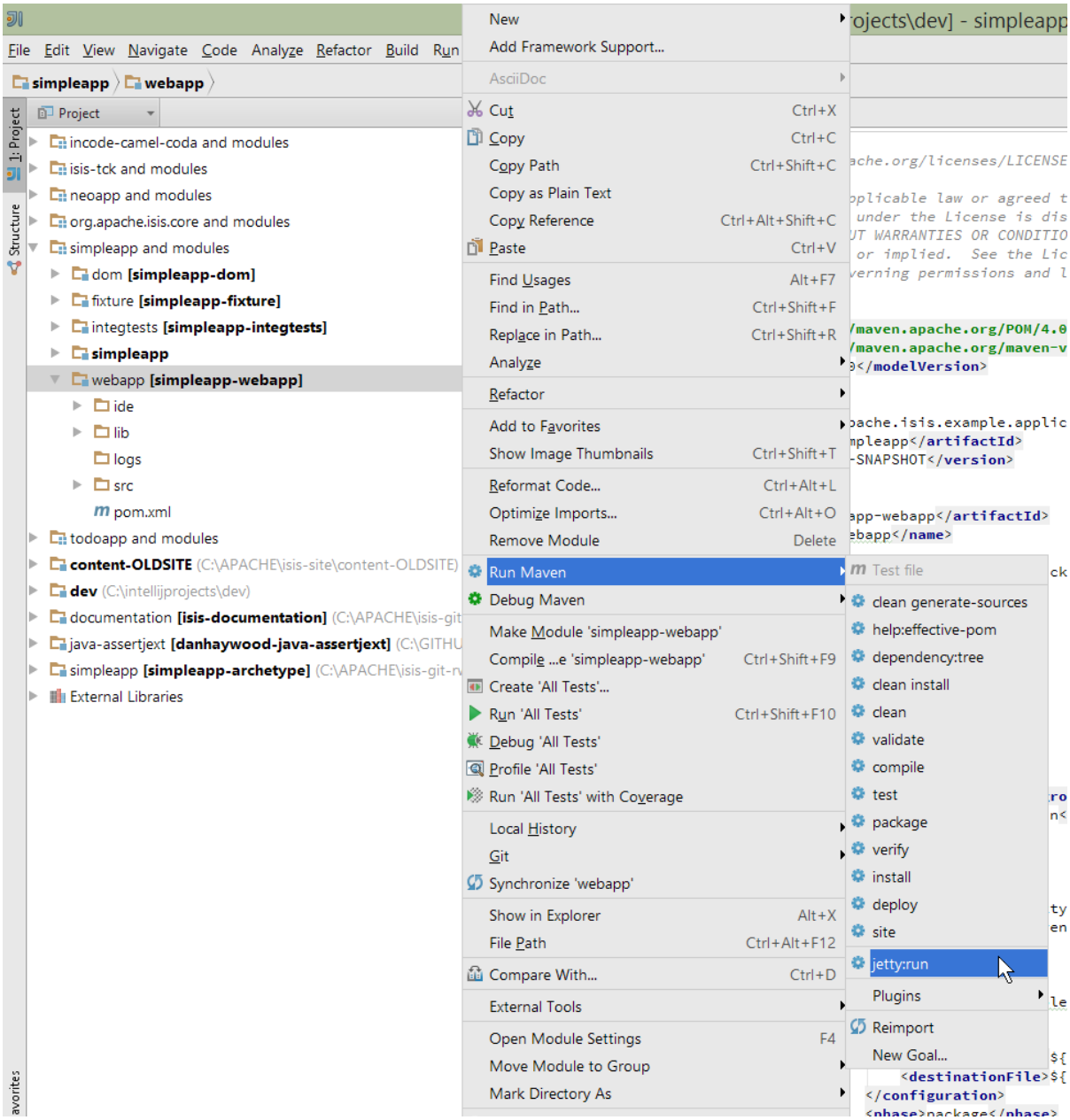
If you open a `pom.xml` file, you'll see an additional "Dependencies" tab:



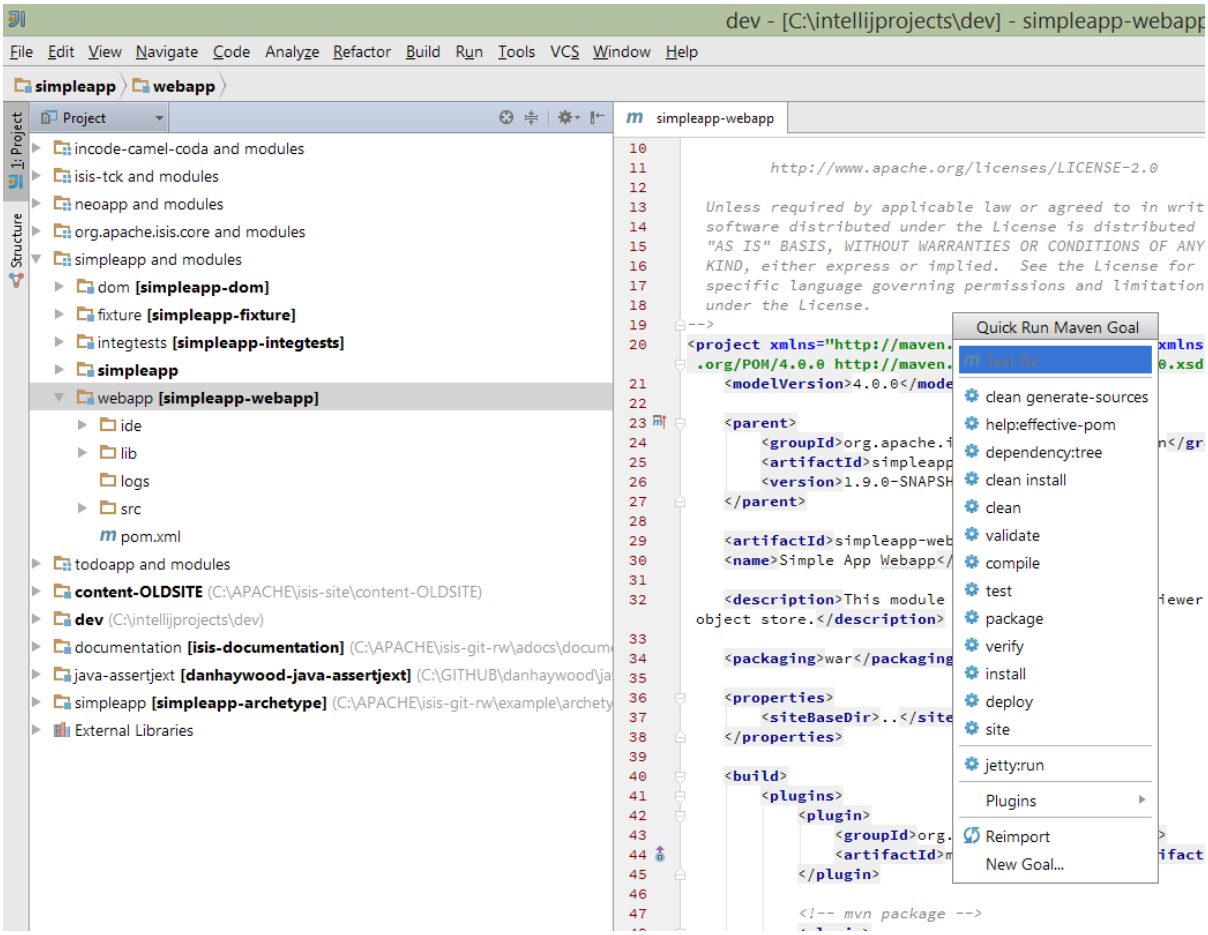
Clicking on this gives a graphical tree representation of the dependencies, similar to that obtained by `mvn dependency:tree`, but filterable.



The plugin also provides the ability to easily run a Maven goal on a project:



This menu can also be bound to a keystroke so that it is available as a pop-up:



Troubleshooting

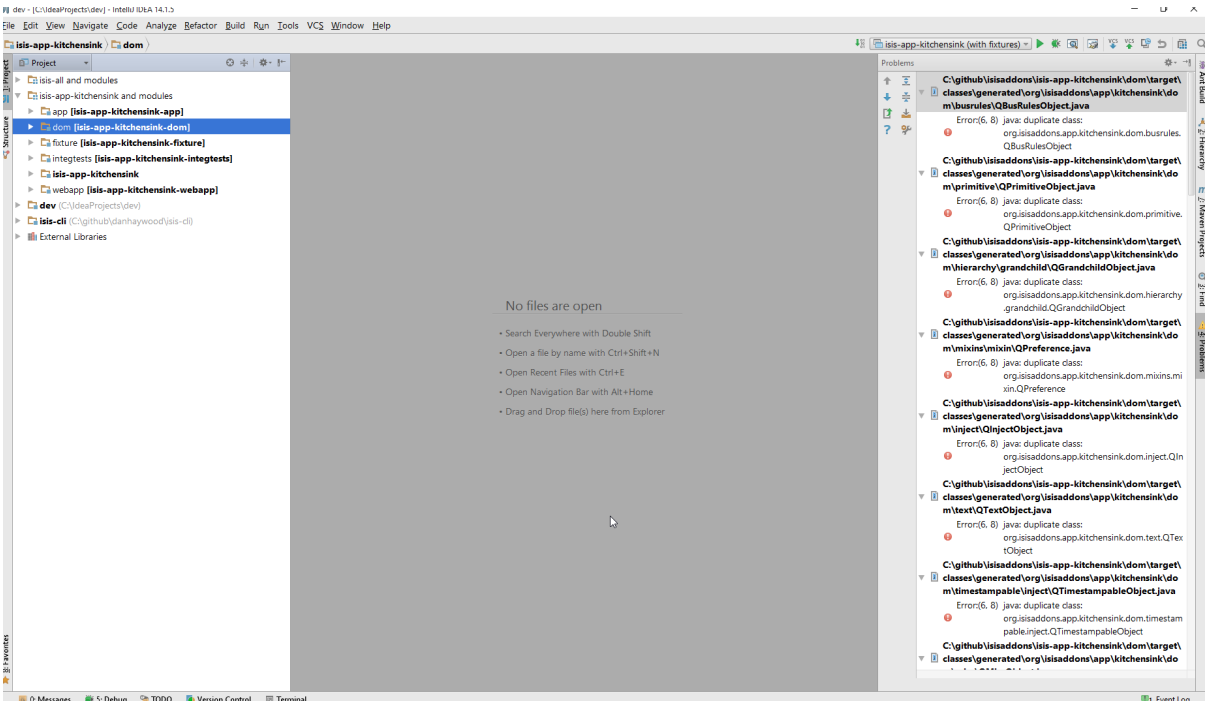
When a Maven module is imported, IntelliJ generates its own project files (suffix `.ipr`), and the application is actually built from that.

Occasionally these don't keep in sync (even if auto-import of Maven modules has been enabled).

To fix the issue, try: * reimport module * rebuild selected modules/entire project * remove and then re-add the project * restart, invalidating caches * hit StackOverflow (!)

One thing worth knowing; IntelliJ actively scans the filesystem all the time. It's therefore (almost always) fine to build the app from the Maven command line; IntelliJ will detect the changes and keep in sync. If you want to force that, use `File > Synchronize`, `ctrl-alt-Y`.

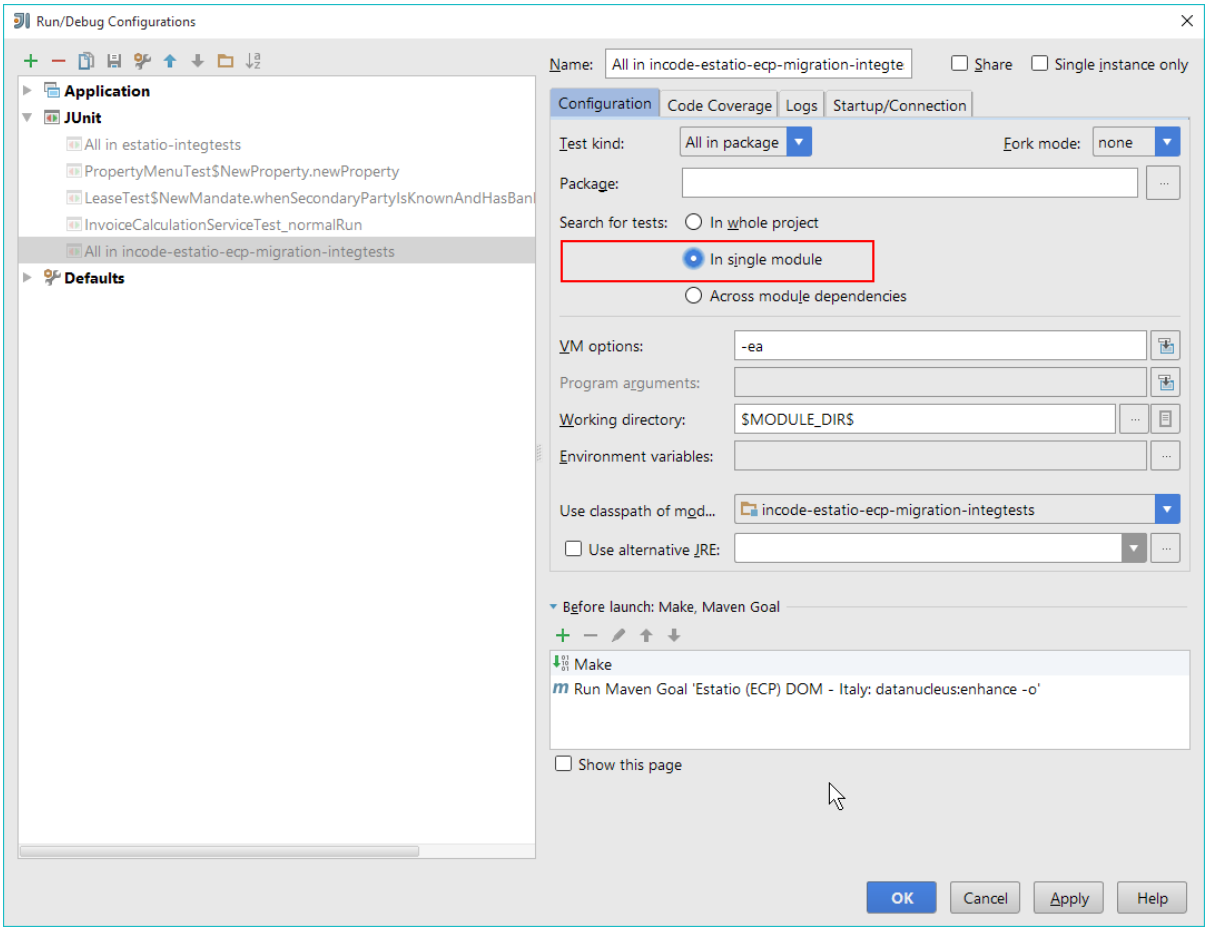
If you hit an error of "duplicate classes":



then make sure you have correctly configured the annotation processor settings. Pay attention in particular to the "Production sources directory" and "Test sources directory", that these are set up correctly.

2.1.5. Running Integration Tests

When running integration tests from within IntelliJ, make sure that the `search for tests` radio button is set to `In single module`:



If this radio button is set to one of the other options then you may obtain class loading issues; these result from IntelliJ attempting to run unit tests of the `dom` project that depend on test classes in that module, but using the classpath of the `integtests` module whereby the `dom` test-classes (`test-jar` artifact) are not exposed on the Maven classpath.

2.1.6. Advanced

In this section are a couple of options that will reduce the length of the change code/build/deploy/review feedback loop.

Setting up Dynamic Reloading

DCEVM enhances the JVM with true hot-swap adding/removing of methods as well as more reliable hot swapping of the implementation of existing methods.

In the context of Apache Isis, this is very useful for contributed actions and mixins and also view models; you should then be able to write these actions and have them be picked up without restarting the application.

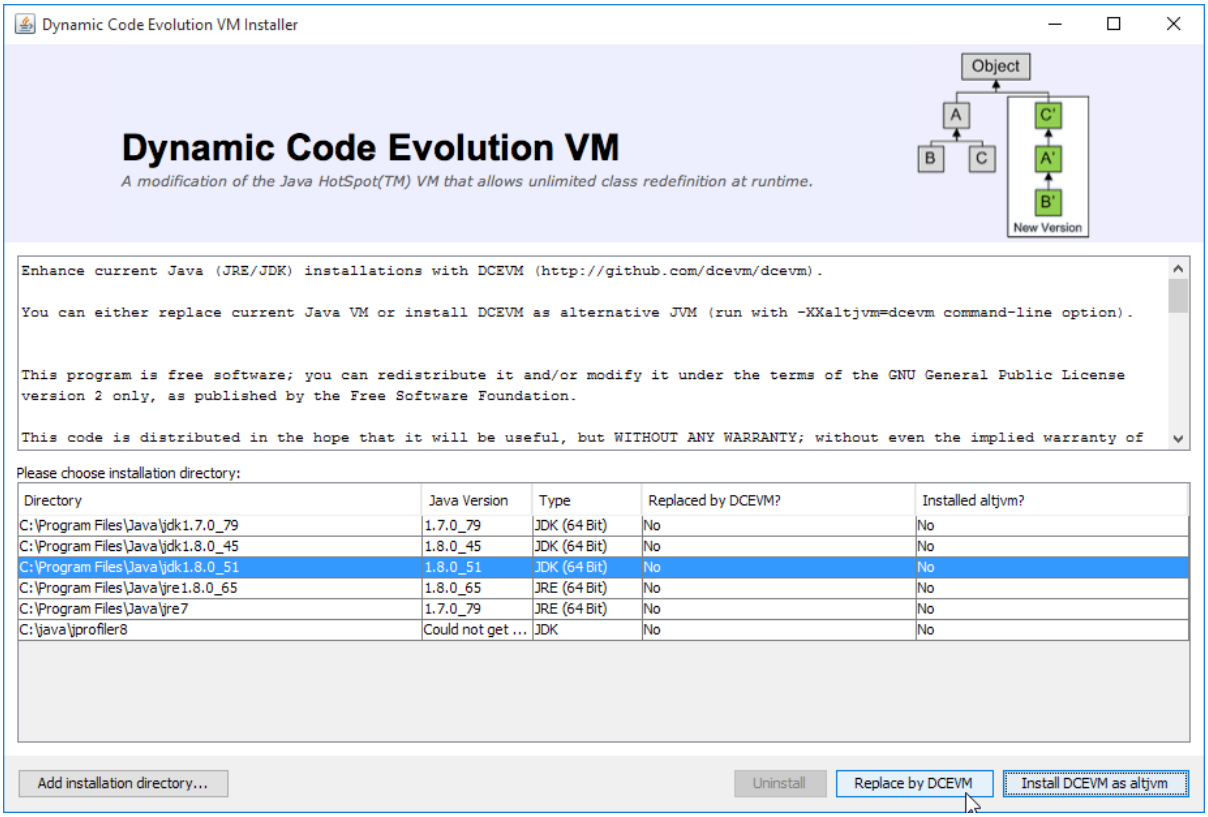
Changing persisting domain entities is more problematic, for two reasons: the JDO/DataNucleus enhancer needs to run on domain entities, and also at runtime JDO/DataNucleus would need to rebuild its own metamodel. You may find that adding actions will work, but adding new properties or collections is much less likely to.

To set up DCEVM, download the appropriate JAR from the github page, and run the installer. For example:

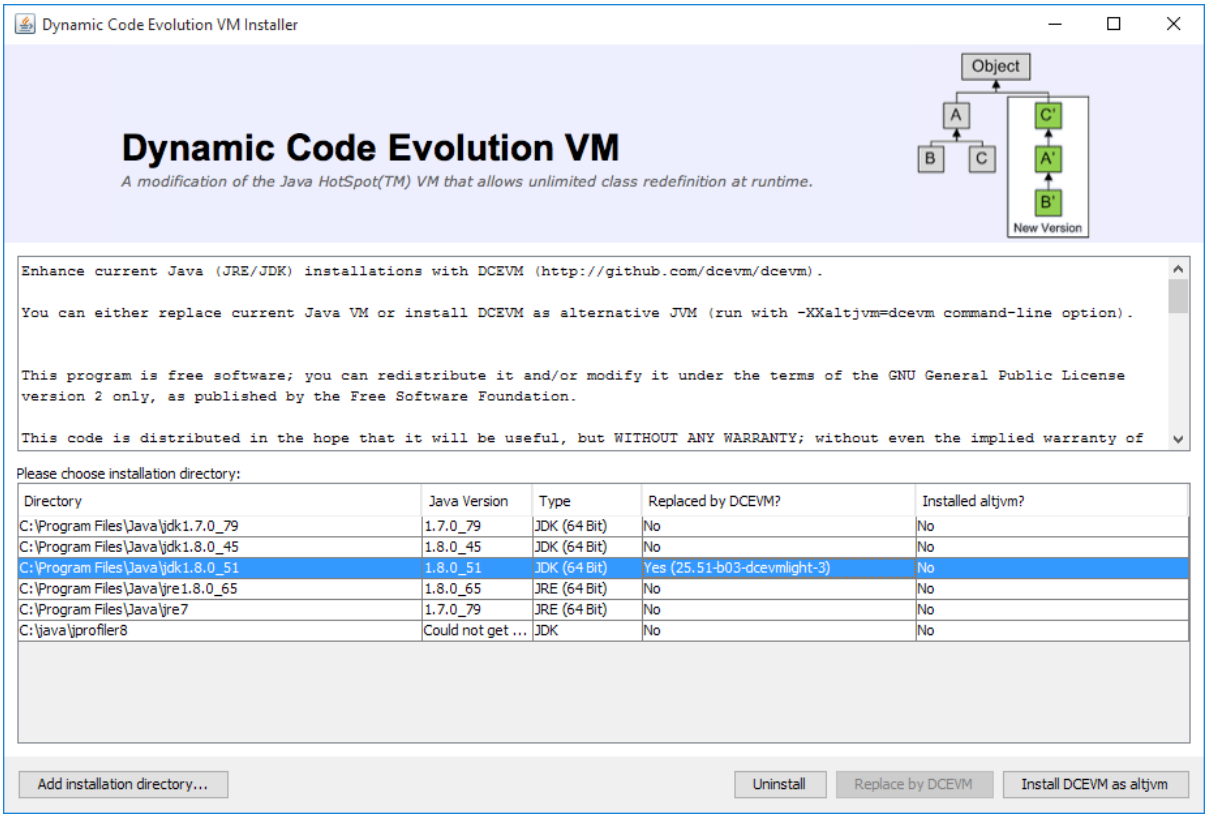
```
java -jar DCEVM-light-8u51-installer.jar
```

Be sure to run with appropriate privileges to be able to write to the installation directories of the JDK. If running on Windows, that means running as `Administrator`.

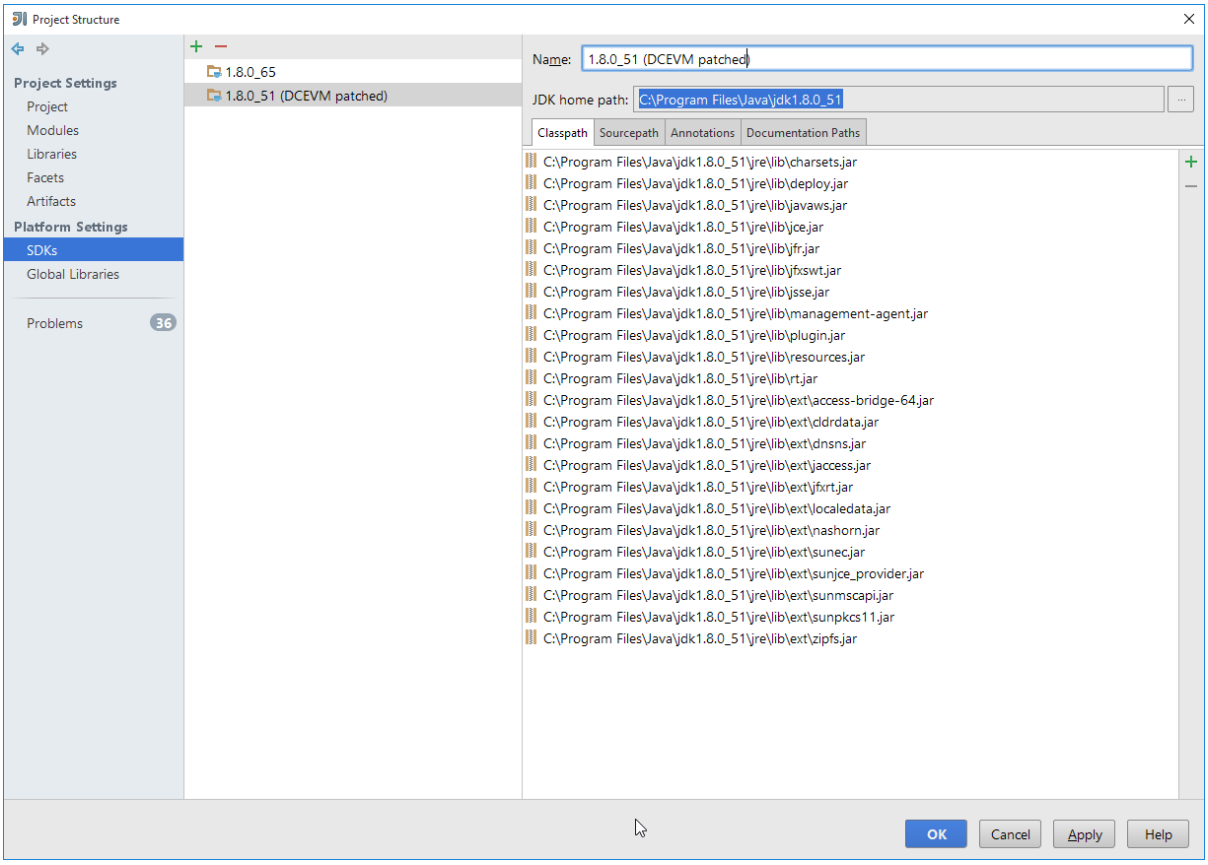
After a few seconds this will display a dialog listing all installations of JDK that have been found:



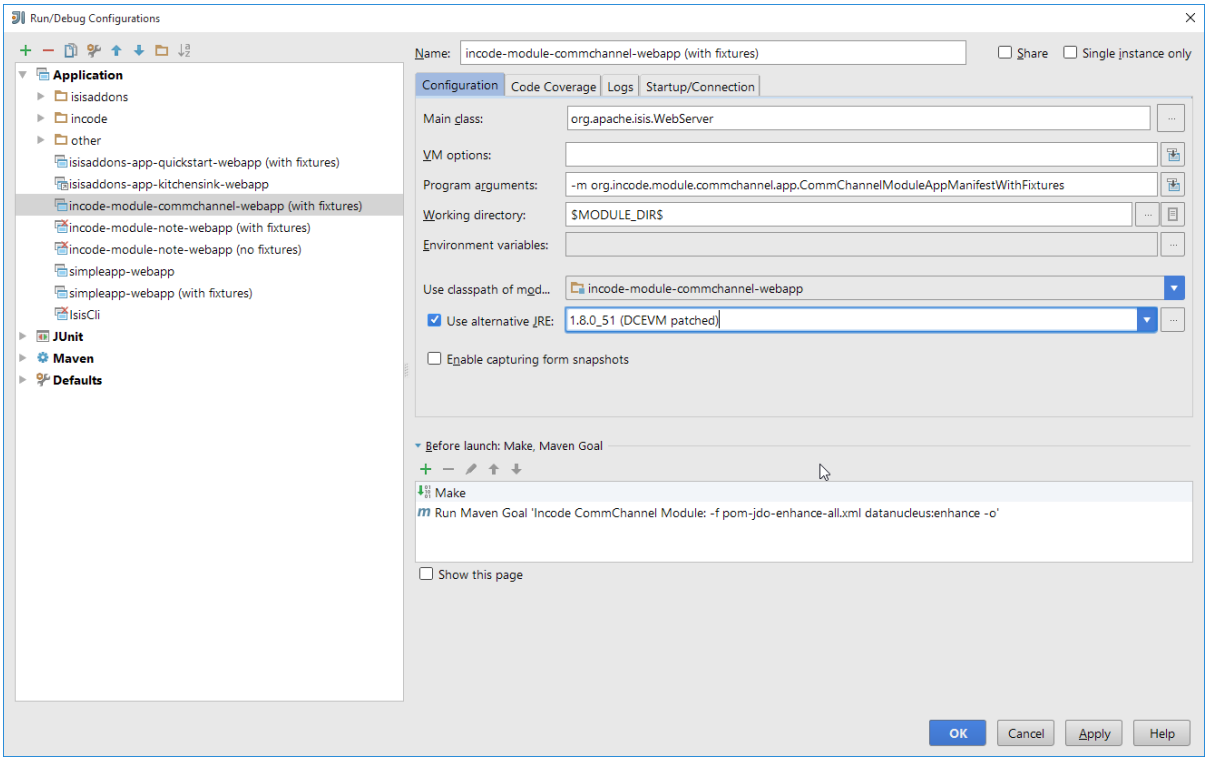
Select the corresponding installation, and select `Replace by DCEVM`.



In IntelliJ, register the JDK in `File > Project Structure` dialog:



Finally, in the run configuration, select the patched JDK:



Setting up JRebel

See the repo for the (non-ASF) Isis JRebel plugin. With some modification, this should work for IntelliJ too.

Note that JRebel is a commercial product, requiring a license. At the time of writing there is also

currently a non-commercial free license (though note this comes with some usage conditions).

2.2. Developing using Eclipse

This material does not constitute an endorsement; Eclipse foundation is not affiliated to Apache Software Foundation in any way.

If you are an Eclipse user, then we recommend you download the "Eclipse JEE package" configuration.

When running an Apache Isis application, it's necessary to setup the development environment so that the Java bytecode can be enhanced by the DataNucleus enhancer. If working in Eclipse, then JDO enhancement is most easily done by installing the DataNucleus' Eclipse plugin. This hooks the bytecode enhancement of your domain objects into Eclipse's normal incremental compilation.

This plugin needs to be configured for each of your domain modules (usually just one in any given app). The steps are therefore:

- import the project into Eclipse
- configure the DataNucleus enhancer
- run the app from the `.launch` file

2.2.1. Screencast

This screencast shows how to import an Apache Isis maven-based application into Eclipse and configure to use with the JDO Objectstore.

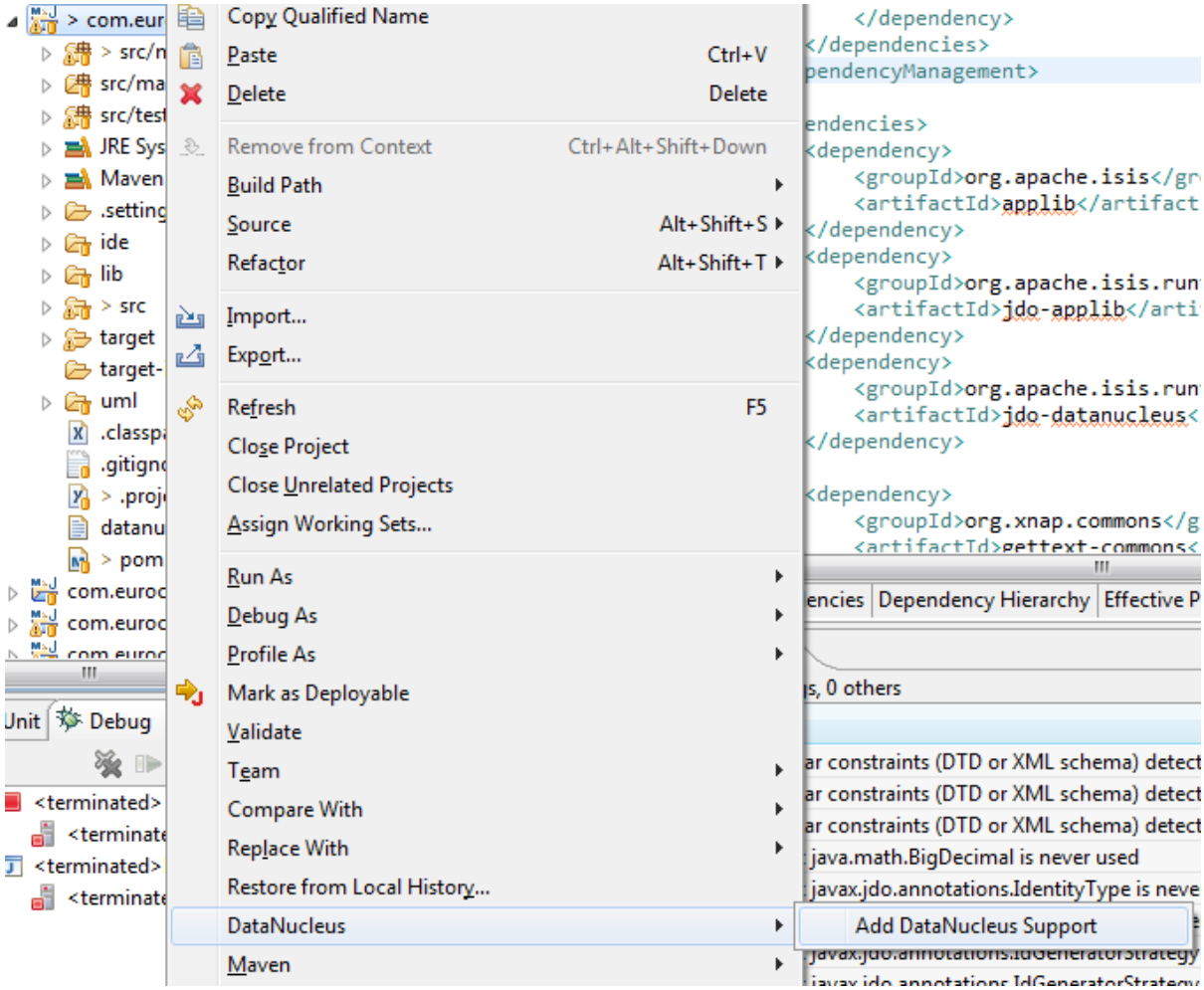
2.2.2. Importing the Project

Use File > Import, then Maven > Existing Maven Projects.

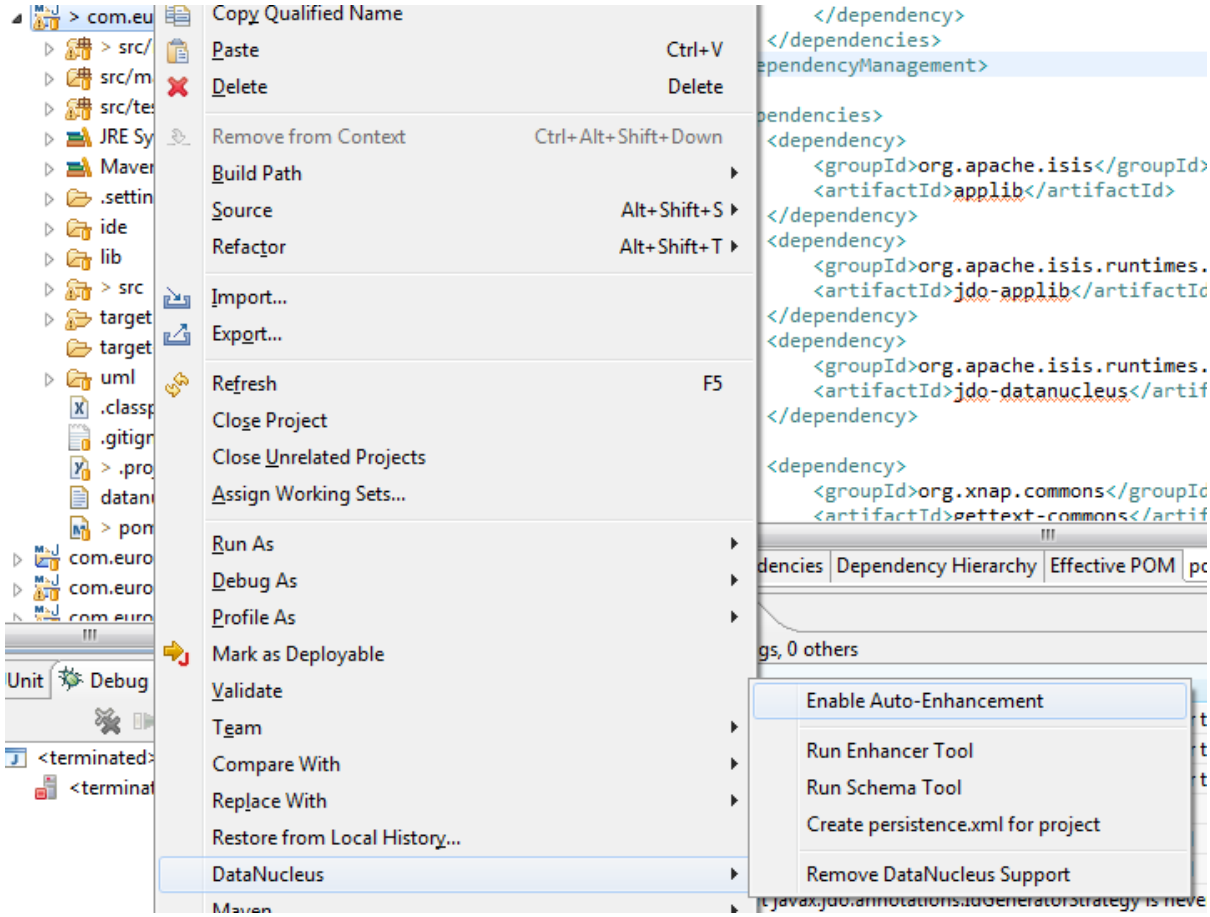
2.2.3. Add DataNucleus support

Make sure you are in the 'Java' Perspective, not the 'Java EE' Perspective.

In Eclipse, for the *domain object model* project, first add DataNucleus support:



Then turn on Auto-Enhancement:

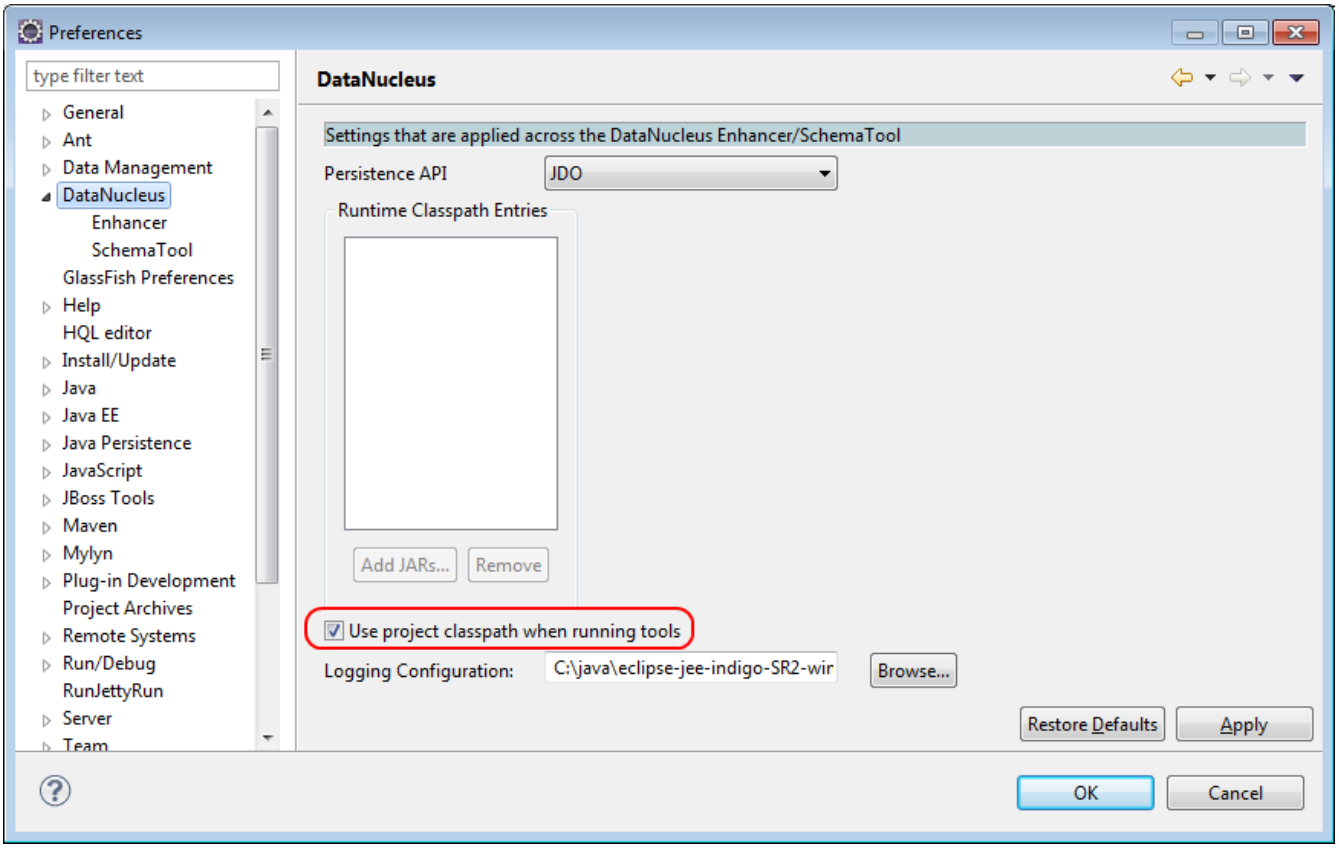


Update the classpath

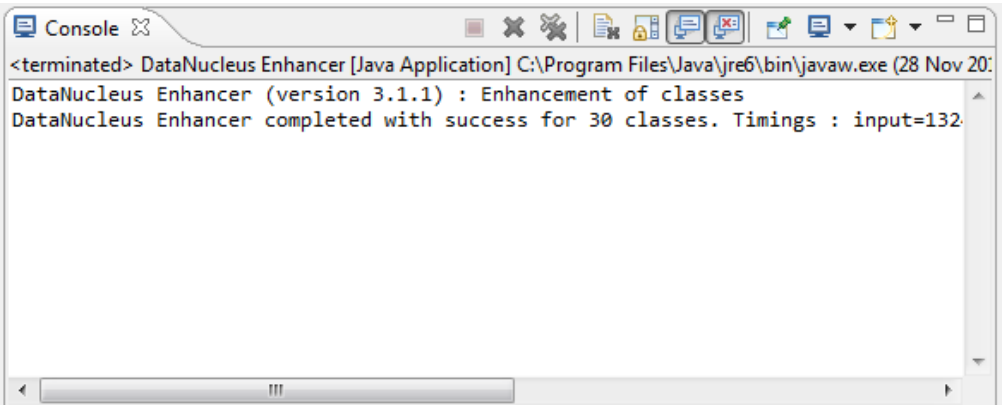
DataNucleus' enhancer uses the domain object model's own classpath to reference DataNucleus JARs. So, even though your domain objects are unlikely to depend on DataNucleus, these references must still be present.

See the earlier section on DataNucleus enhancer for details of the contents of the `pom.xml`. Chances are it is already set up from running the SimpleApp archetype.

Then, tell DataNucleus to use the project classpath:



When the enhancer runs, it will print out to the console:

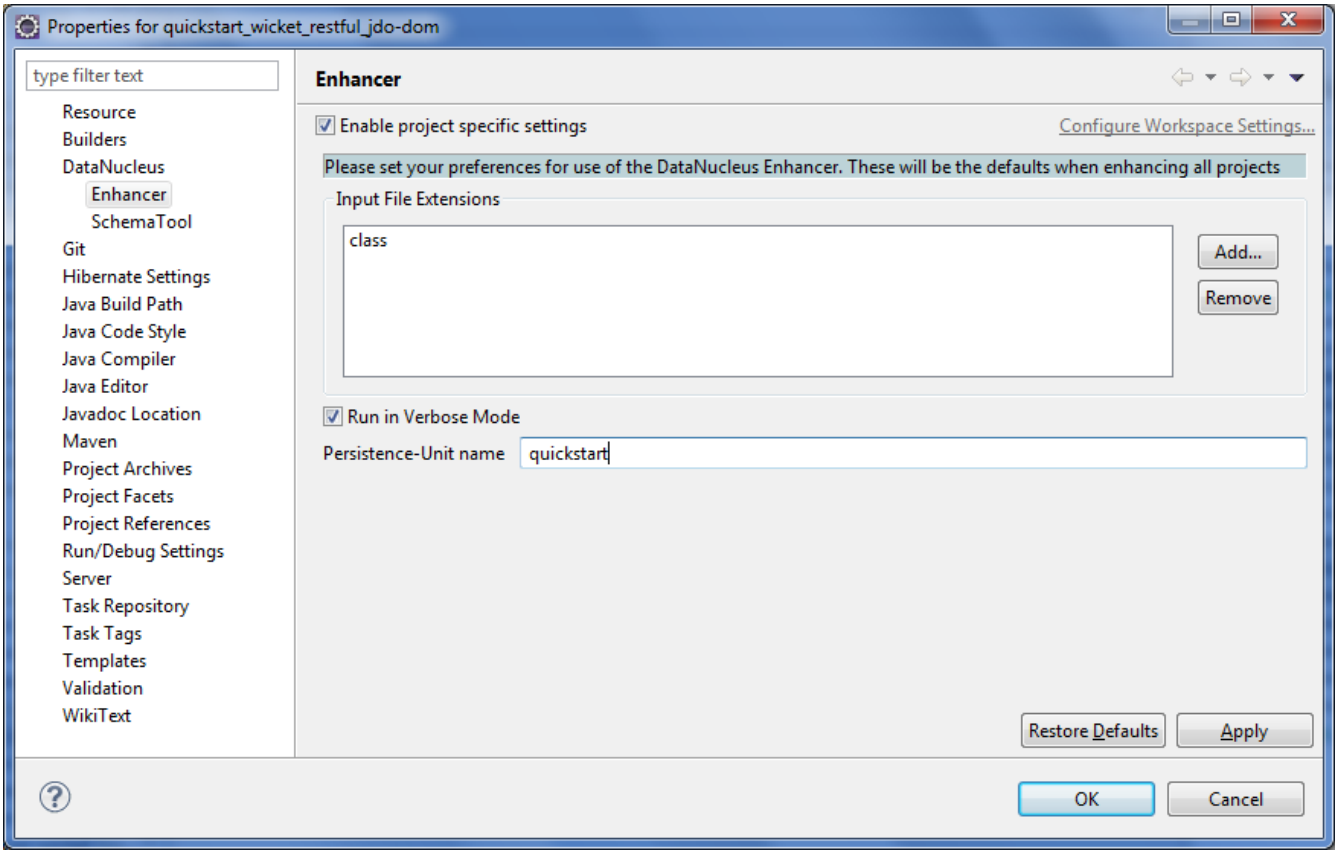


Workaround for path limits (the DN plugin to use the persistence.xml)

If running on Windows then the DataNucleus plugin is very likely to hit the Windows path limit.

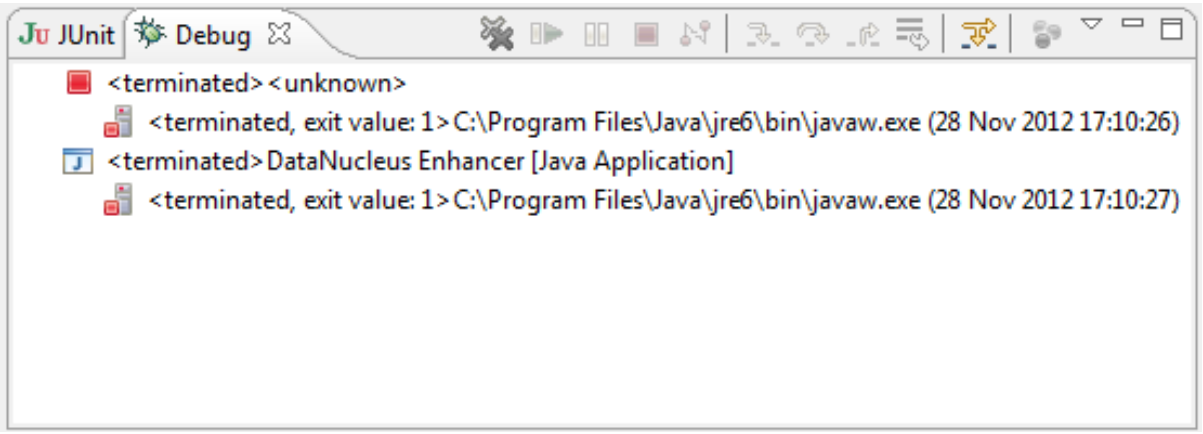
To fix this, we configure the enhancer to read from the `persistence.xml` file.

As a prerequisite, first make sure that your domain object model has a `persistence.xml` file. Then specify the `persistence-unit` in the project properties:

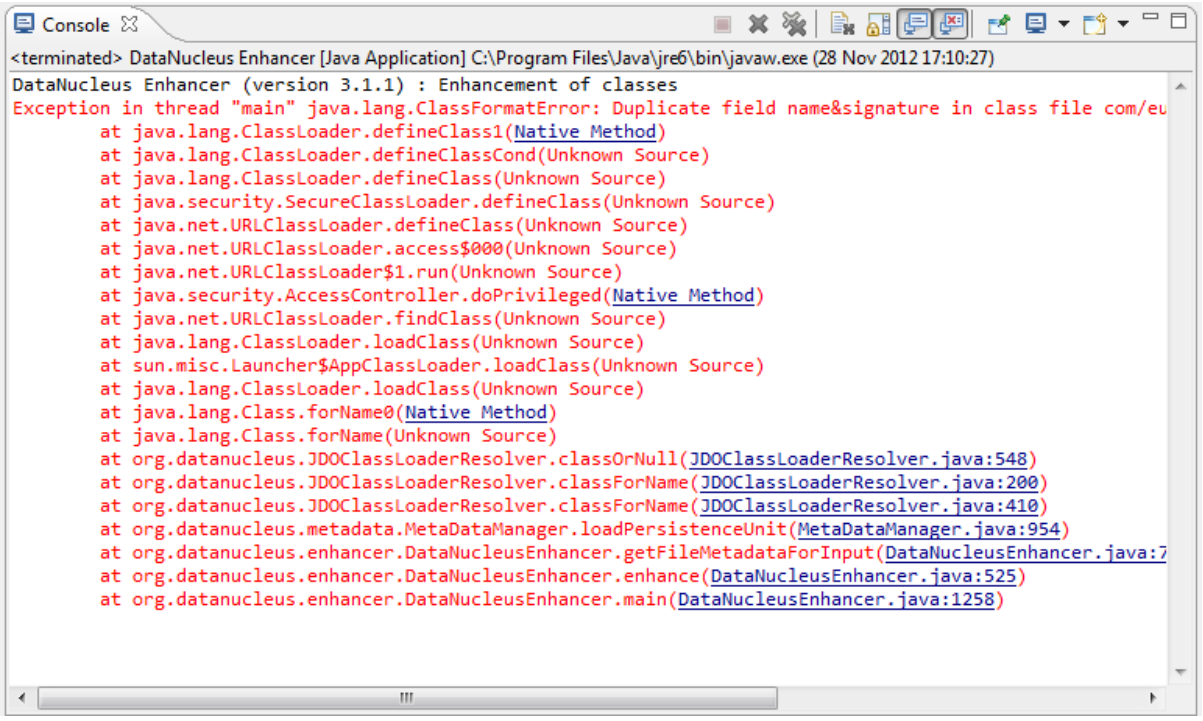


Workaround: If the enhancer fails

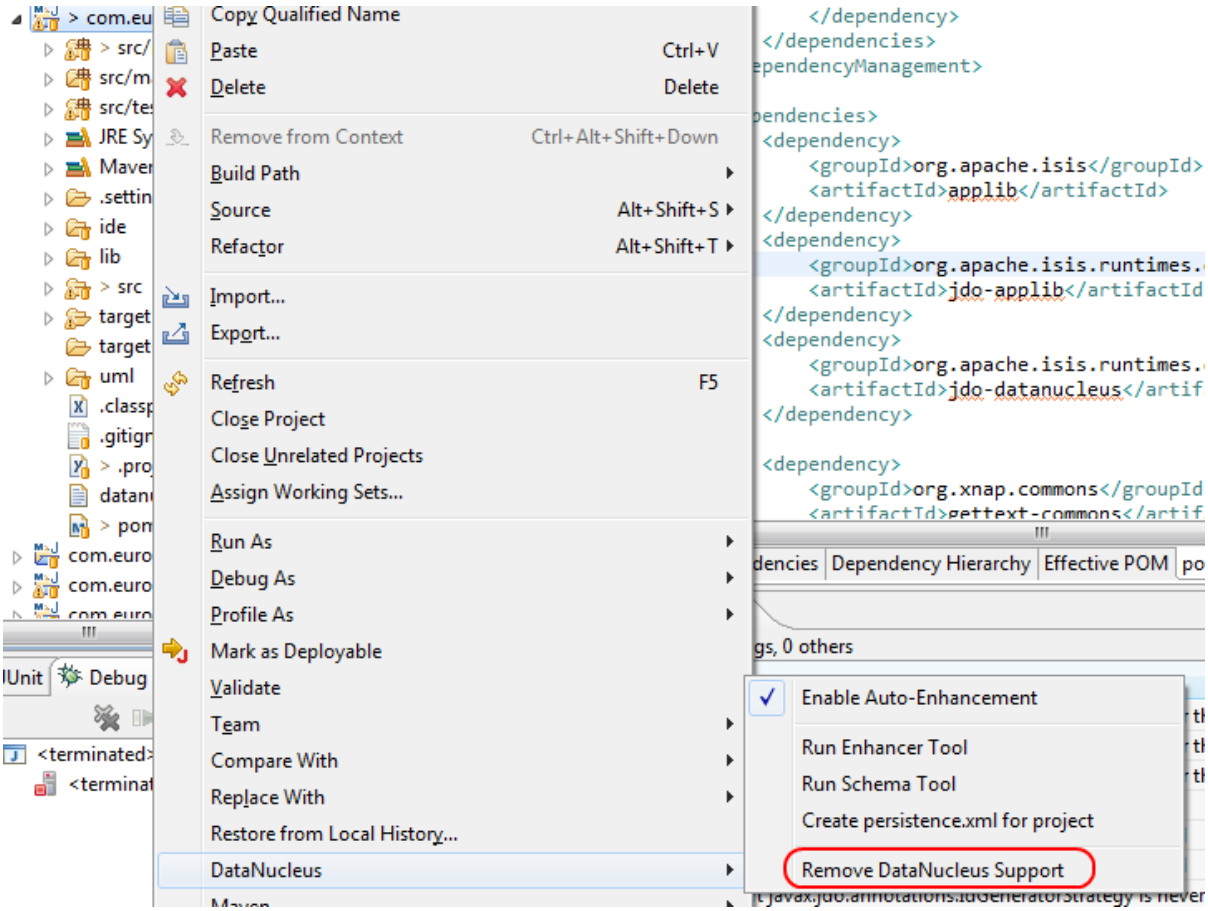
On occasion it appears that Eclipse can attempt to run two instances of the DataNucleus enhancer. This is probably due to multiple Eclipse builders being defined; we've noticed multiple entries in the Eclipse's `Debug` view:



At any rate, you'll know you've encountered this error if you see the following in the console:



The best solution is to remove DataNucleus support and then to re-add it:



If you consistently hit problems, then the final recourse is to disable the automatic enhancement and to remember to manually enhance your domain object model before each run.

Not ideal, we know. Please feel free to contribute a better solution :-)

2.2.4. Running the App

The simpleapp archetype automatically provides a `.launch` configurations in the `webapp` module. You can therefore very simply run the application by right-clicking on one of these files, and choosing "Run As..." or "Debug As...".

The screencast above shows this in action.

2.2.5. Other domain projects.

There is nothing to prevent you having multiple domain projects. You might want to do such that each domain project corresponds to a DDD module, thus guaranteeing that there are no cyclic dependencies between your modules.

If you do this, make sure that each project has its own `persistence.xml` file.

And, remember also to configure Eclipse's DataNucleus plugin for these other domain projects.

2.2.6. Advanced

In this section are a couple of options that will reduce the length of the change code/build/deploy/review feedback loop.

Setting up Dynamic Reloading

DCEVM enhances the JVM with true hot-swap adding/removing of methods as well as more reliable hot swapping of the implementation of existing methods.

In the context of Apache Isis, this is very useful for contributed actions and mixins and also view models; you should then be able to write these actions and have them be picked up without restarting the application.

Changing persisting domain entities is more problematic, for two reasons: the JDO/DataNucleus enhancer needs to run on domain entities, and also at runtime JDO/DataNucleus would need to rebuild its own metamodel. You may find that adding actions will work, but adding new properties or collections is much less likely to.

For details of setting up DCEVM, see the corresponding section in the IntelliJ documentation.

Chapter 3. Code and File Templates

We provide parameterized templates, for both IntelliJ and Eclipse, to help you write your domain applications.

On IntelliJ we provide both file templates (`File > Settings > Editor > File and Code Templates`) and also live templates (`File > Settings > Editor > Live Templates`). The former are used to create new classes or files (eg a new domain entity), while the latter are intended to modify an existing file (eg create a new property or add a `toString()` method etc).

On Eclipse we provide only the latter sort of template (Windows > Preferences > Java > Editor > Templates).

There are templates for writing Apache Isis domain objects, for writing unit tests (JUnit and JMock), and also for writing Asciidoc documentation (see also the appendix).

3.1. Download

The following table lists the templates available to download:

Purpose	IntelliJ file template	Prefix	IntelliJ live template	Eclipse template
Domain Objects	Download	`is`	Download	Download
JUnit tests	(none)	`ju`	Download	Download
JMock tests	(none)	`jm`	Download	Download
Asciidoc	(none)	`ad`	Download	(none)

The most commonly used domain objects (live) templates are also listed on the Apache Isis cheat sheet.

3.2. Installation

3.2.1. IntelliJ

To install in the live templates IntelliJ (Community edition 15), copy to the relevant `config/templates` directory, eg:

- Windows `<User home>\.IdeaIC15\config\templates`
- Linux `~/.IdeaIC15/config/templates`
- Mac OS `~/Library/Preferences/IdeaIC15/templates`

If using the Ultimate edition, the directory is `.IntelliJIdea15` rather than `IdeaIC15`.

To install the file templates, use `File > Import Settings`.

3.2.2. Eclipse

To install in Eclipse, go to `Windows > Preferences > Java > Editor > Templates` and choose `Import`.

3.3. Usage

For the live templates, enter the prefix in the editor (`is`, `ju`, `jm`) and the IDE will list all available templates in that category.

For the file templates (IntelliJ only), these are available from `File > New`.

Chapter 4. Building Apache Isis

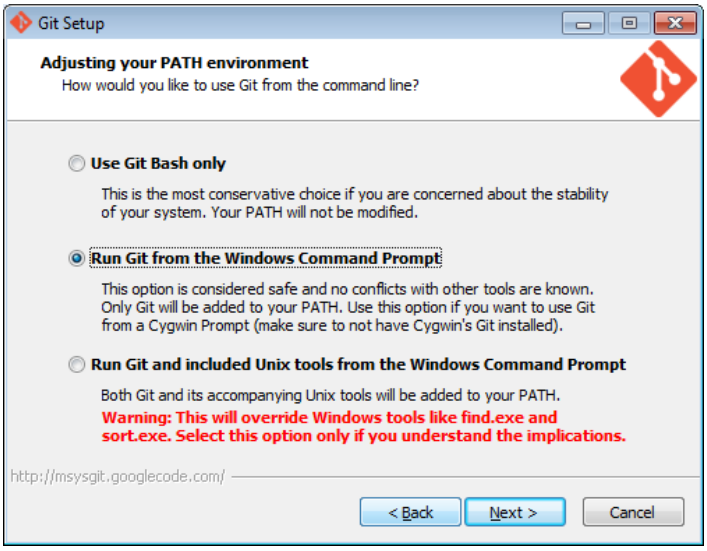
4.1. Git

The Apache Isis source code lives in a git repo.

4.1.1. Installation

The easiest place to get hold of command-line git is probably the github download page.

On Windows, this also installs the rather good mSysGit Unix shell. We recommend that you enable git for both the mSysgit and the Windows command prompt:



Once git is installed, the two main command line tools to note are:

- `git` command line tool
- `gitk` for viewing the commit history

If using Windows, note that github also have a dedicated Windows client. With a little hacking around, it can also be made to work with non-github repositories.

If using Mac, you might also want to check out Atlassian's Sourcetree.

Cloning the Apache Isis repo

First, clone the Apache Isis repo.

If you are a **committer**, then clone from the Apache read/write repo:

```
git clone https://git-wip-us.apache.org/repos/asf/isis.git
```

If you are **not a committer**, please see the contributing page for details on which repo to clone from.

Configuring Git

Next up is to configure your user name and password; see also Apache's git docs:

```
git config user.name "<i>My Name Here</i>"
git config user.email <i>myusername@apache.org</i>
```

Next, configure the `core.autocrlf` so that line endings are normalized to LF (Unix style) in the rep; again see Apache's git page:

- on Windows, use:

  ```
  git config core.autocrlf true
  ```

- on Mac/Linux, use:

  ```
  git config core.autocrlf input
  ```

The Windows setting means that files are converted back to CRLF on checkout; the Mac/Linux setting means that the file is left as LF on checkout.

We also recommend setting `core.safecrlf`, which aims to ensure that any line ending conversion is repeatable. Do this on all platforms:

```
git config core.safecrlf true
```

Note that these settings are supplemented in the repo by the `.gitattributes` file and that explicitly specifies line handling treatment for most of the common file types that we have.

Next, we recommend you setup this a refspec so that you can distinguish remote tags from local ones. To do that, locate the `[remote "origin"]` section in your `.git/config` and add the third entry shown below:

```
[remote "origin"]
    url = ... whatever ...
    fetch = ... whatever ...
    fetch = +refs/tags/*:refs/tags/origin/*
```

This will ensure that a `git fetch` or `git pull` places any remote tags under `origin/xxx`. For example, the`isis-1.0.0`tag on the origin will appear under`origin/isis-1.0.0`.

If you don't use git outside of Apache, you can add the `--global` flag so that the above settings apply for all repos managed by git on your PC.

4.1.2. Getting help

Three commands of git that in particular worth knowing:

- `git help command`

  will open the man page in your web browser

- `git gui`

  will open up a basic GUI client to staging changes and making commits.

- `gitk --all`

  will open the commit history for all branches. In particular, you should be able to see the local `master`, which branch you are working on (the `HEAD`), and also the last known position of the `master` branch from the central repo, called `origin/master`.

You might also want to explore using a freely available equivalent such as Atlassian SourceTree.

For further reading, see:

- git config man page
- .gitattributes man page
- .gitattributes git-scm.com docs

4.2. Installing Java

Apache Isis is compatible with Java 7 and Java 8. For every-day use, the framework is usually compiled against Java 8.

Releases however are cut using Java 7, leveraging the link :http://maven.apache.org/plugins/maven-toolchains-plugin/[Maven toolchains plugin]).

Therefore install either/both of Java 7 JDK and Java 8 JDK. Note that the JRE is *not* sufficient.

If you intend to contribute back patches to Apache Isis, note that while you can develop using Java 8 within your IDE, be sure not to use any Java 8 APIs.

4.2.1. Configure Maven toolchains plugin

If you are a committer that will be performing releases of Apache Isis, then you *must* configure the toolchains plugin so that releases can be built using Java 7.

This is done by placing the `toolchains.xml` file in `~/.m2` directory. Use the following file as a template, adjusting paths for your platform:

```
<?xml version="1.0" encoding="UTF8"?>
<toolchains>
    <toolchain>
        <type>jdk</type>
        <provides>
            <version>1.8</version>
            <vendor>oracle</vendor>
        </provides>
        <configuration>
            <jdkHome>/usr/lib64/jvm/jdk1.8.0_65</jdkHome>
            <!--
            <jdkHome>c:\Program Files\Java\jdk1.8.0_65</jdkHome>
            -->
        </configuration>
    </toolchain>
    <toolchain>
        <type>jdk</type>
        <provides>
            <version>1.7</version>    ①
            <vendor>oracle</vendor>
        </provides>
        <configuration>
            <jdkHome>/usr/lib64/jvm/jdk1.7.0_79</jdkHome>
            <!--
            <jdkHome>c:\Program Files\Java\jdk1.7.0_79</jdkHome>
            -->
        </configuration>
    </toolchain>
</toolchains>
```

① The Apache Isis build is configured to search for the (`1.7, oracle`) JDK toolchain.

The Apache Isis parent `pom.xml` activates this plugin whenever the `apache-release` profile is enabled.

4.3. Installing Maven

Install Maven 3.0.x, downloadable here.

Set `MAVEN_OPTS` environment variable:

```
export MAVEN_OPTS="-Xms512m -Xmx1024m"
```

Previously we suggested `-XX:MaxPermSize=256m`, but this option has been removed in Java 8. (As of 1.9.0, Apache Isis is built using Java 8 but with source and target set to JDK 1.7).

4.4. Building all of Apache Isis

To build the source code from the command line, simply go to the root directory and type:

```
mvn clean install
```

The first time you do this, you'll find it takes a while since Maven needs to download all of the Apache Isis prerequisites.

Thereafter you can speed up the build by adding the `-o` (offline flag). To save more time still, we also recommend that you build in parallel. (Per this blog post), you could also experiment with a number of JDK parameters that we've found also speed up Maven:

```
export MAVEN_OPTS="-Xms512m -Xmx1024m -XX:+TieredCompilation -XX:TieredStopAtLevel=1"
mvn clean install -o -T1C
```

For the most part, though, you may want to rely on an IDE such as Eclipse to build the codebase for you. Both Eclipse and Idea (12.0+) support incremental background compilation.

When using Eclipse, a Maven profile is configured such that Eclipse compiles to `target-ide` directory rather than the usual `target` directory. You can therefore switch between Eclipse and Maven command line without one interfering with the other.

4.5. Checking for Vulnerabilities

Apache Isis configures the OWASP dependency check Maven plugin to determine whether the framework uses libraries that are known to have security vulnerabilities.

To check, run:

```
mvn org.owasp:dependency-check-maven:aggregate -Dowasp
```

This will generate a single report under `target/dependency-check-report.html`.

The first time this runs can take 10~20 minutes to download the NVD data feeds.

To disable, either run in offline mode (add `-o` or `--offline`) or omit the `owasp` property.

4.6. Checking for use of internal JDK APIs

Apache Isis configures the jdeps maven plugin to check for any usage of internal JDK APIs. This is in preparation for Java 9 module system (Jigsaw) which will prevent such usage of APIs.

To check, run:

```
mvn clean install -Djdeps
```

This will fail the build on any module that currently uses an internal JDK API.

At the time of writing the `isis-core-schema` module fails the build.

Chapter 5. AsciiDoc Documentation

Apache Isis' documentation (meaning the website and the users' guide, the reference guide and this contributors' guide) is written using Asciidoc, specifically the Asciidoctor implementation.

The website and guides are created by running build tools (documented below) which create the HTML version of the site and guides. You can therefore easily check the documentation before raising a pull request (as a contributor) or publishing the site (if a committer).

Publishing is performed by copying the generated HTML to a different git repository (isis-site). This is synced by ASF infrastructure over to isis.apache.org.

And to help write the Asciidoc text itself, we provide some templates.

5.1. Where to find the Docs

The (Asciidoc) source code can be found at `adocs/documentation` (relative to root). Online you'll find it cloned to github here.

5.2. Naming Conventions

For documents with inclusions, use '_' to separate out the logical hierarchy:

```
xxx-xxx/xxx-xxx.adoc
        _xxx-xxx_ppp-ppp.adoc
        _xxx-xxx_qqq-qqq.adoc
        _xxx-xxx_qqq-qqq_mmm-mmm.adoc
        _xxx-xxx_qqq-qqq_nnn-nnn.adoc
```

Any referenced images should be in subdirectories of the `images` directory:

```
xxx-xxx/images/.
              /ppp-ppp/.
              /qqq-qqq/.
                      /mmm-mmm
                      /nnn-nnn
```

And similarly any resources should be in the `resources` subdirectory:

```
xxx-xxx/resources/.
                  ppp-ppp/.
                  qqq-qqq/.
                         /mmm-mmm/
                         /nnn-nnn/
```

5.3. Writing the docs

We highly recommend that you install the (IntelliJ) live templates for Asciidoctor, as described in IDE templates. These provide a large number of helper templates.

An appendix lists all the templates available, demonstrating their intended usage and output.

5.4. Build and Review (using Maven)

To (re)build the documentation locally prior to release, change into the `adocs/documentation` directory and use:

```
mvn clean compile
```

The site will be generated at `target/site/index.html`.

You could then use a web server such as Python's SimpleHTTPServer to preview (so that all Javascript works correctly). However, instead we recommend using instant preview, described next.

5.5. Instant Rebuild (using Ruby)

The ruby script, `monitor.rb` emulates the `mvn compile` command, regenerating any changed Asciidoctor files to the relevant `target/site` directory. Moreover if any included files are changed then it rebuilds the parent (per the above naming convention).

5.5.1. One-time setup

To setup:

- download and install ruby 2.0.0, from http://rubyinstaller.org/downloads/
- download devkit for the Ruby 2.0 installation, also from http://rubyinstaller.org/downloads/. Then follow the installation instructions on their wiki

We use Ruby 2.0 rather than 2.1 because the wdm gem (required to monitor the filesystem if running on Windows) is not currently compatible with Ruby 2.1.

To download the required Ruby dependencies, use:

```
gem install bundler
bundle install
```

5.5.2. Instant Rebuild

To run, we typically just use:

```
sh monitor.sh
```

This script simply runs `mvn clean compile && ruby monitor.rb -b`. The `mvn` command performs a clean rebuild of the site, and then the ruby script monitors for any further changes under `src/main/asciidoc`.

The script also starts up a web server on port 4000 so you can review results. If any `.adoc` changes, then the appropriate HTML will be regenerated. And, if any new assets (CSS, images etc) are added, they will be copied across. The `-b` flag passed through means that the script also starts a web browser pointing at the newly generated docs.

The `monitor.rb` script has a couple of other options, use `-h` flag for usage:

```
ruby monitor.rb -h
```

which should print:

```
usage: monitor.rb [options]
    -p, --port       port (default: 4000)
    -b, --browser    launch browser
    -h, --help       help
```

5.6. Publish procedure

Only Apache Isis committers can publish to isis.apache.org. We've decided to include these procedures here here (rather than put them in the Committers' Guide), just to keep things together.

5.6.1. One-time setup

The generated site is published by copying into the `content/` directory of the isis-site git repo. You therefore need to check this out this repo.

The procedure assumes that two git repos (for isis itself and for isis-site) are checked out into the same parent directory, eg:

```
/APACHE/
    isis/                       # checkout of isis.git
        adocs/
            documentation/
                README.adoc     # this file you are reading right now
                ...
    isis-site/                  # checkout of isis-site.git
        content/                # the published website
```

If this isn't the case, then it is possible to override the relative directory by passing in a system

property to the mvn goal; see below.

You also need to know that ASF's publishing script work from the 'asf-site' branch, NOT from the 'master' branch. Therefore, in the `isis.git` repo site:

```
git checkout asf-site
```

5.6.2. Publishing

Back in the `adocs/documentation` directory of the main `isis-git.repo`, to copy the generated documents to the `isis-site.git` repo, run:

```
mvn clean package
```

This deletes the entire content of `contents`, and replaces with the content under `target/site`. It also fixes up line endings, standardizing on unix-style LFs.

If you have checked out the `isis-site.git` repo into some other directory (relative to `isis.site.git`), then this can be overridden by specifying `-Disis-site.dir=⋯` when calling `mvn`.

To copy and to also commit the generated documents to the `isis-site.git` repo , run:

```
sh publish.sh "ISIS-nnnn: a custom commit message"
```

Behind the scenes this just calls `mvn clean install -Dmessage=⋯`.

Pushing the commits (in the `isis-site.git` directory, of course) will publishing the changes:

```
git push
```

Double check at isis.apache.org.

Chapter 6. Contributing

This page explains how you can contribute to Apache Isis. You'll probably also want set up your IDE and learn how to build Apache Isis.

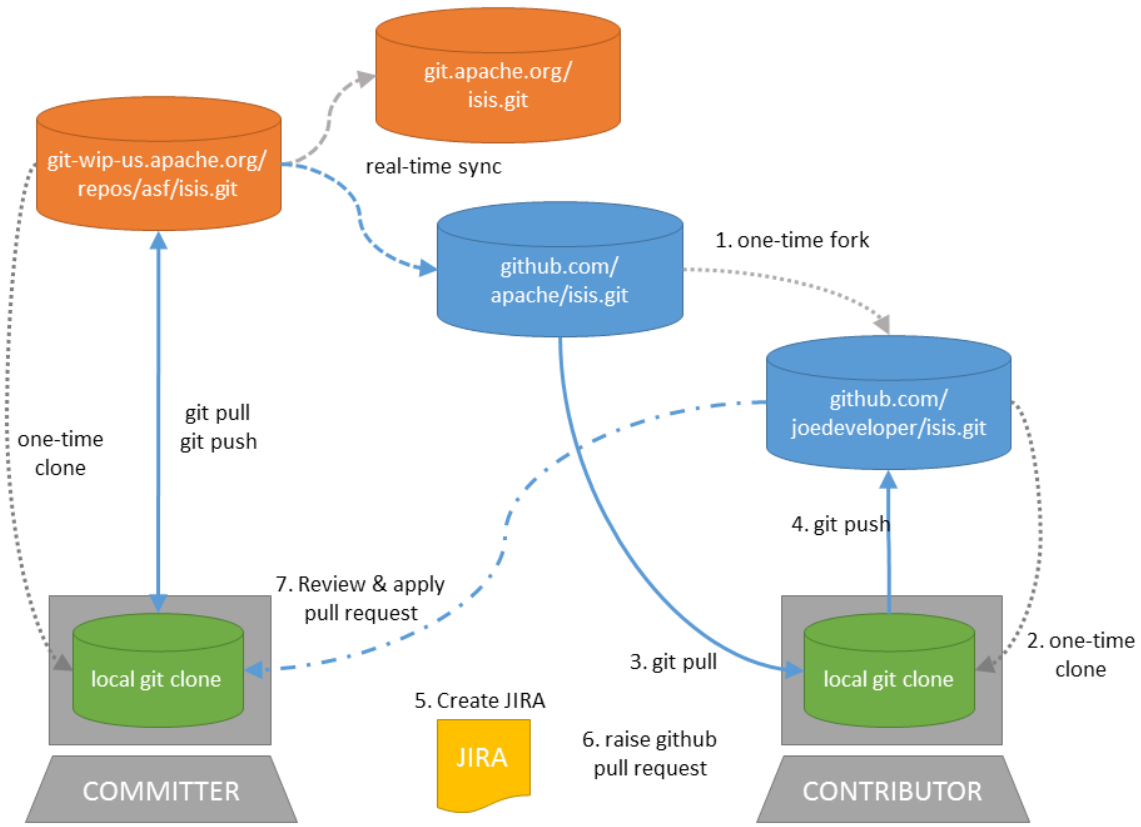
Thanks for considering to help out, your contributions are appreciated!

6.1. Recommended Workflow (github)

Apache Isis' source code is hosted in an Apache git repo (https, http), with a clone on github (https, or ssh: `git@github.com:apache/isis.git`.

As you might imagine, only committers are permitted to push changes to the central git repo. As a contributor, we recommend that you fork the apache/isis repo in github, and then use your fork as a way of publishing your patches for the Apache Isis committers to apply.

The diagram below illustrates the process:



That is:

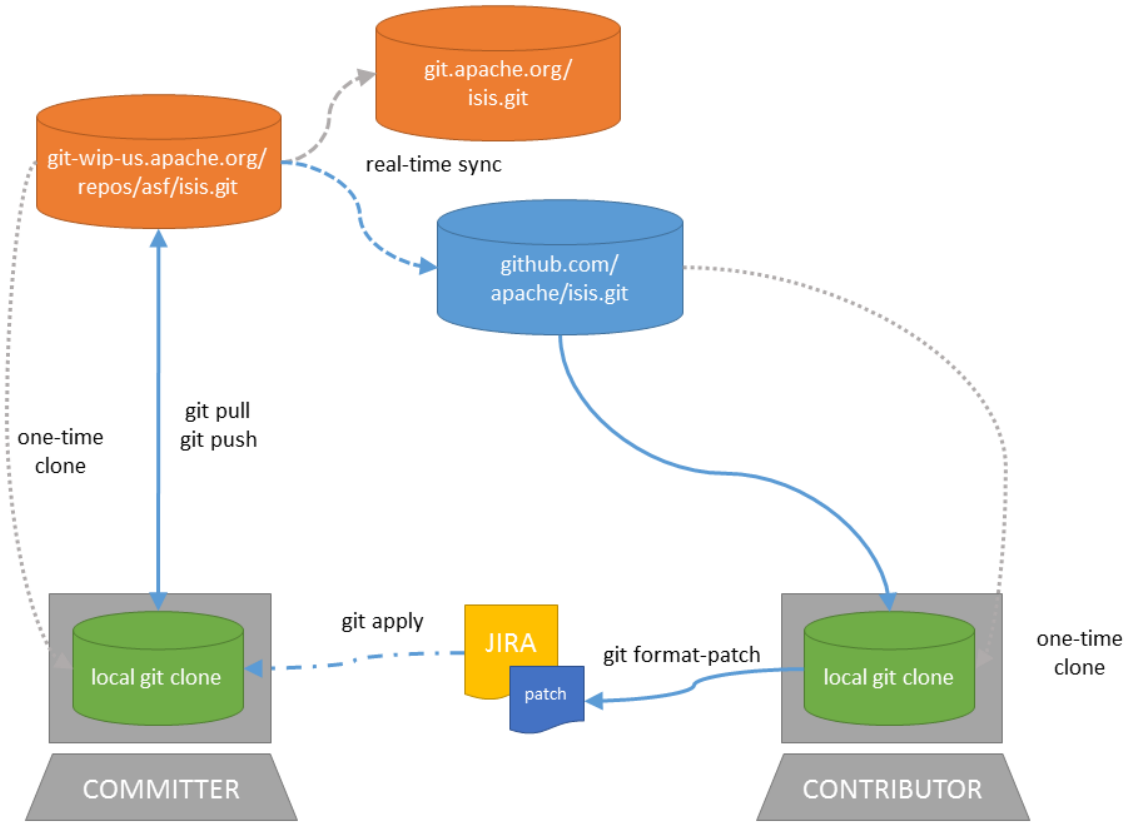
1. as a one-time activity, you fork the github.com/apache/isis repo into your own fork on github.com
2. as a one-time activity, you clone your fork to your local computer
3. you set the github.com/apache/isis as your upstream branch; this will allow you to keep your local clone up-to-date with new commits

- note the asymmetry here: the `upstream` repo (the Apache github repo) is **not** the same as the `origin` repo (your fork).

4. you work on your changes locally; when done, you push them to your github fork
5. to contribute back a change, raise a JIRA ticket, and ensure your commit message is in the form: `ISIS-nnnn: ⋯` so that changes can be tracked (more discussion on this point below). In any case, before you decide to start hacking with Apache Isis, it's always worth creating a ticket in JIRA and then have a discussion about it on the mailing lists.
6. Use github to raise a pull request for your feature
7. An Apache Isis committer will review your change, and apply it if suitable.

6.2. Alternative Workflow (JIRA patches)

As an alternative, you may decide to clone directly from github.com/apache/isis rather than create your own fork:



In this case your `upstream` repo is the same as your `origin` repo, which might seem more straightforward. On the other hand, if you go this route then you'll need create patches locally and attach them to the JIRA ticket.

For the Apache Isis committers it really doesn't matter which route you take, so go with whatever's most comfortable.

6.3. Setting up your fork/clone

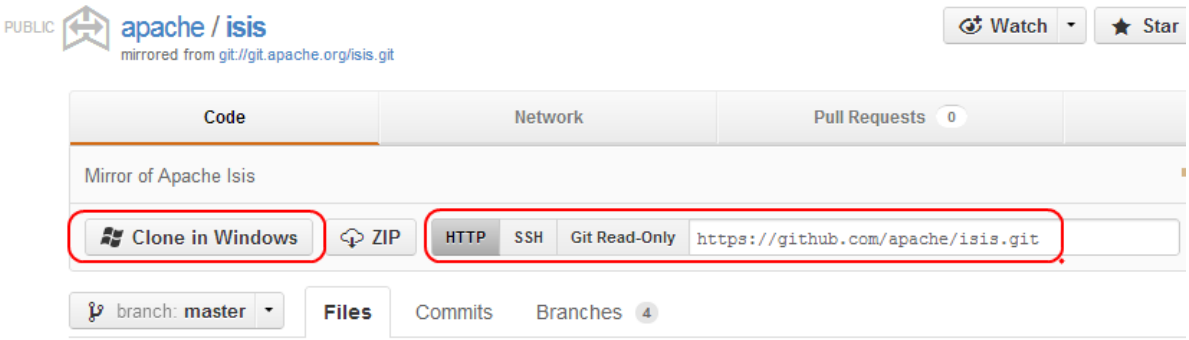
If you choose to create your own fork then you'll need an account on github.com. You then fork simply by pressing the "Fork" button:

An account isn't needed if you just clone straight from the github.com/apache/isis.

Whether you've forked or not, you then need to clone the repo onto your computer. Github makes this very easy to do:

- for Windows users, we suggest you use github's 'Clone in Windows' feature
- for Mac/Linux users, create a clone from the command line:

Again, the info is easily found in the github page:



If you've created your own fork, then you need to add the `upstream` remote to the github.com/apache/isis. This remote is traditionally called `upstream`. You should then arrange for your `master` branch to track the `upstream/master` remote branch:

If you didn't create your own fork, you can omit the above step. Either way around, you can now fetch new commits using simply:

```
git fetch
```

For more info on tracking branches here and here.

6.4. Commit messages

Although with git your commits are always performed on your local repo, those commit messages become public when the patch is applied by an Apache Isis committer. You should take time to write a meaningful commit message that helps explain what the patch refers to; if you don't then there's a chance that your patch may be rejected and not applied. No-one likes hard work to go to waste!

We therefore recommend that your commit messages are as follows [1]:

```
ISIS-999: Make the example in CONTRIBUTING imperative and concrete

Without this patch applied the example commit message in the CONTRIBUTING
document is not a concrete example.  This is a problem because the
contributor is left to imagine what the commit message should look like
based on a description rather than an example.  This patch fixes the
problem by making the example concrete and imperative.

The first line is a real life imperative statement with a ticket number
from our issue tracker.  The body describes the behavior without the patch,
why this is a problem, and how the patch fixes the problem when applied.
```

Once your git repo is setup, the next thing you'll most likely want to do is to setup your development environment. See here for more details.

6.5. Creating the patch file

If you are working without a github fork of Apache Isis, then you can create the patches from your own local git repository.

As per this stackoverflow question, create the patch using `git format-patch`:

```
git format-patch -10 HEAD --stdout > 0001-last-10-commits.patch
```

Here `-10` is the last 10 commits you have done. You need to change that integer according to the commits you need to apply into the patch.

6.6. Sample Contribution Workflow

Assuming you're development environment is all setup, let's walk through how you might make contribute a patch. In this example, suppose that you've decided to work on JIRA ticket #123, an enhancement to support Blob/Clob datatypes.

6.6.1. Update your master branch

The first thing to do is to make sure your local clone is up-to-date. We do this by retrieving new commits from upstream repo and then merging them as a fast-forward into your local branch.

Irrespective of whether you are using a github fork, the upstream for your local `master` branch will be tracking the appropriate remote's `master` branch. So n either case, the same commands work:

Alternatively, you can combine the `git fetch` and `git merge` and just use `git pull`: <pre> git checkout master git pull –ff-only </pre>

If the `merge` or `pull` fails, it means that you must have made commits and there have been changes

meanwhile on the remote `master's branch. You can use `gitk --all` to confirm. If this fails, see our git cookbook page for a procedure to retrospectively sort out this situation.

6.6.2. Create a topic branch

We recommend you name topic branches by the JIRA ticket, ie <tt>ISIS-nnn-description</tt>. So let's create a new branch based off `master` and call it "ISIS-123-blobs"

You can confirm the branch is there and is your new `HEAD` using either `gitk --all`. Alternatively, use the command line:

```
$ git checkout -b ISIS-123-blobs
```

The command line prompt should also indicate you are on a branch, isolated from any changes that might happen on the `master` branch.

6.6.3. Make File Changes and Commit

Next, make changes to your files using the usual commands (see also our git cookbook section):

- `git add`
- `git mv`
- `git rm`
- `git commit`
- `git status`

and so on.

Continue this way until happy with the change. Remember to run all your tests on the topic branch (including a full `mvn clean install`).

6.6.4. Rebasing with `master`

Before you can share your change, you should rebase (in other words replay) your changes on top of the `master` branch.

The first thing to do is to pull down any changes made in upstream remote's `master` since you started your topic branch:

These are the same commands that you would have run before you created your topic branch. If you use `gitk --all`, there's a good chance that new commits have come in.

Next, we reintegrate our topic branch by rebasing onto `master`: <pre> git checkout ISIS-123-blobs git rebase master </pre>

This takes all of the commits in your branch, and applies them on top of the new `master` branch. When your change is eventually integrated back in, it will result in a nice clear linear history on the public repo.

If the rebase fails because of a conflict, then you'll be dumped into REBASE mode. Edit the file that has the conflict, and make the appropriate edits. Once done:

Once the rebase has completed, re-run your tests to confirm that everything is still good.

6.6.5. Raising a pull request

If you have your own fork, you can now simply push the changes you've made locally to your fork:

This will create a corresponding branch in the remote github repo. If you use `gitk --all`, you'll also see a `remotes/origin/ISIS-123-blobs` branch.

Then, use github to raise a pull request. Pull requests sent to the Apache GitHub repositories will forward a pull request e-mail to the dev mailing list. You'll probably want to sign up to the dev mailing list first before issuing your first pull request (though that isn't mandatory).

The process to raise the pull request, broadly speaking:

- Open a web browser to your github fork of isis
- Select your topic branch (pushed in the previous step) so that the pull request references the topic branch.
- Click the `Pull Request` button.
- Check that the Apache Isis mailing list email came through.

6.7. If your pull request is accepted

To double check that your pull request is accepted, update your `master` branch from the `upstream` remote:

You can then use `gitk --all` (or `git log` if you prefer the command line) to check your contribution has been added.

You can now delete your topic branch and remove the branch in your github:

Finally, you might want to push the latest changes in master back up to your github fork. If so, use:

6.7.1. If your pull request is rejected

If your pull request is rejected, then you'll need to update your branch from the main repository and then address the rejection reason.

You'll probably also want to remove the remote branch on github:

```
git push origin –delete ISIS-123-blobs
```

... and continue as before until you are ready to resubmit your change.

[1] inspiration for the recommended commit format comes from the puppet project's contributing

page.

Chapter 7. Appendix: Git Cookbook

This appendix describes the commands often used while working with git. In addition to these basic commands, please make sure you have read:

- building Apache Isis
- Contributing
- Git policy

7.1. Modifying existing files

To modify existing files:

```
git add filename
git commit -m "ISIS-nnn: yada yada"
```

The `git add` command adds the changes to the file(s) to the git index (aka staging area). If you were to make subsequent changes to the file these would not be committed.

The `git commit` takes all the staged changes and commits them locally. Note that these changes are not shared public with Apache Isis' central git repo.

You can combine these two commands using `-am` flag to git commit:

```
git commit -am "ISIS-nnn: yada yada"
```

7.2. Adding new files

To add a new file:

```
git add .
git commit -m "ISIS-nnn: yada yada"
```

Note that this sequence of commands is identical to modifying an existing file. However, it isn't possible to combine the two steps using `git commit -am`; the `git add` is always needed when adding new files to the repo.

7.3. Deleting files

To delete a file:

```
git rm filename
git commit -m "ISIS-nnn: yada yada"
```

7.4. Renaming or moving files

To rename or move a file:

```
git mv <i>filename</i> <i>newfilename</i>
git commit -m "ISIS-nnn: yada yada"
```

7.5. Common Workflows

The contributing page describes the workflow for non-committers. The Git policy page describes a workflow for Apache Isis **committers**.

7.6. Backing up a local branch

If committing to a local branch, the changes are still just that: local, and run risk of a disk failure or other disaster.

To create a new, similarly named branch on the central repo, use:

```
git push -u origin <i>branchname</i>
```

Using `gitk --all` will show you this new branch, named **origin/branchname**.

Thereafter, you can push subsequent commits using simply:

```
git push
```

Doing this also allows others to collaborate on this branch, just as they would for `master`.

When, eventually, you have reintegrated this branch, you can delete the remote branch using:

```
git push origin --delete <i>branchname</i>
```

For more detail, see these blogs/posts here and here.

7.7. Quick change: stashing changes

If you are working on something but are not ready to commit, then use:

```
git stash
```

If you use `gitk --all` then you'll see new commits are made that hold the current state of your working directory and staging area.

You can then, for example, pull down the latest changes using `git pull --rebase` (see above).

To reapply your stash, then use:

```
git stash pop
```

Note that stashing works even if switching branches

7.8. Ignoring files

Put file patterns into `.gitignore`. There is one at the root of the git repo, but they can additionally appear in subdirectories (the results are cumulative).

See also:

- github's help page
- man page

7.9. More advanced use cases

7.9.1. If accidentally push to remote

Suppose you committed to `master`, and then pushed the change, and then decided that you didn't intend to do that:

```
C1  -  C2  -  C3  -  C4  -  C5  -  C6  -  C7
                                          ^
                                          master
                                          ^
                                          origin/master
```

To go back to an earlier commit, first we wind back the local `master`:

```
git reset --hard C5
```

where `C5` is the long sha-id for that commit.

This gets us to:

```
C1  -  C2  -  C3  -  C4  -  C5  -  C6  -  C7
                            ^
                            master
                                        ^
                                        origin/master
```

Then, do a force push:

```
git push origin master --force
```

If this doesn't work, it may be that the remote repo has disabled this feature. There are other hacks to get around this, see for example here.

7.10. If you've accidentally worked on `master` branch

If at any time the `git pull` from your upstream fails, it most likely means that you must have made commits on the `master` branch. You can use `gitk --all` to confirm; at some point in time both `master` and `origin\master` will have a common ancestor.

You can retrospectively create a topic branch for the work you've accidentally done on `master`.

First, create a branch for your current commit:

```
git branch <i>newbranch</i>
```

Next, make sure you have no outstanding edits. If you do, you should commit them or stash them:

```
git stash
```

Finally, locate the shaId of the commit you want to roll back to (easily obtained in `gitk -all`), and wind `master` branch back to that commit:

```
git checkout master
git reset --hard <i>shaId</i>      # move master branch shaId of common ancestor
```

7.11. If you've forgotten to prefix your commits (but not pushed)

One of our committers, Alexander Krasnukhin, has put together some git scripts to help his workflow. Using one of these, `git prefix`, you can just commit with proper message without bothering about prefix and add prefix only in the end **before** the final push.

For example, to prefix all not yet prefixed commits `master..isis/666` with `ISIS-666` prefix, use:

```
git prefix ISIS-666 master..isis/666
```

You can grab this utility, and others, from this repo.

Chapter 8. Appendix: Asciidoc Templates

This appendix lists the (IntelliJ) live templates available for writing documentation using Asciidoc. Instructions for installing the templates can be found here.

In the examples below the text xxx, yyy, zzz are correspond to template variables (ie placeholders).

8.1. Callouts

The Asciidoctor terminology is an "admonition".

Abbrev.	Produces	Example
adadmimportant	[IMPORTANT] ==== xxx ====	[IMPORTANT] ==== xxx ====
adadmnote	[NOTE] ==== xxx ====	[NOTE] ==== xxx ====
adadmtip	[TIP] ==== xxx ====	[TIP] ==== xxx ====
adadmwarning	[WARNING] ==== xxx ====	[WARNING] ==== xxx ====

8.2. TODO notes

Add as a placeholder for documentation still to be written or which is work-in-progress.

Abbrev.	Produces	Example
adtodo	<pre>NOTE: TODO</pre>	NOTE: TODO
adwip	<pre>NOTE: WIP - xxx</pre> where: * <code>xxx</code> is additional explanatory text	NOTE: WIP - cool new feature

8.3. Xref to Guides

Cross-references (links) to the various guides

Abbrev.	Produces	Example
adcgcom	<pre>xref:cgcom.adoc#xxx[ttt]</pre> a hyperlink to a bookmark within the committers' guide, where: * <code>xxx</code> is the bookmark’s anchor * <code>ttt</code> is the text to display as the hyperlink for example: <pre>xref:dg.adoc#_cgcom_cutting-a-release[Cutting a release]</pre>	addg
<pre>xref:dg.adoc#xxx[ttt]</pre> a hyperlink to a bookmark within the developers' guide, where: * <code>xxx</code> is the bookmark’s anchor * <code>ttt</code> is the text to display as the hyperlink for example: <pre>xref:dg.adoc#_dg_asciidoc-templates[Asciidoc templates]</pre>	Asciidoc templates	adrgant

Abbrev.	Produces	Example
<pre>xref:rgant.adoc#xxx[ttt]</pre> a hyperlink to a bookmark within the reference guide for annotations, where: * <code>xxx</code> is the bookmark * <code>ttt</code> is the text to display as the hyperlink for example: <pre>xref:rgant.adoc#_rgant_aaa_main[Core annotations]</pre>	Core annotations	`adrgcfg`

Abbrev.	Produces	Example
<pre>xref:rgcfg.adoc#xxx[ttt]</pre> a hyperlink to a bookmark within the reference guide for configuration properties guide, where: * <code>xxx</code> is the bookmark * <code>ttt</code> is the text to display as the hyperlink for example: <pre>xref:rgcfg.adoc#_rgcfg_configuring-core[Configuring Core]</pre>	Configuring Core	`adrgcms`

Abbrev.	Produces	Example
<pre>xref:rgcms.adoc#xxx[ttt]</pre> a hyperlink to a bookmark within the reference guide for classes, methods and schema, where: * <code>xxx</code> is the bookmark * <code>ttt</code> is the text to display as the hyperlink for example: <pre>xref:rgcms.adoc#_rgcms_classes_super_AbstractService[`AbstractService`]</pre>	`AbstractService`	`adrgsvc`

Abbrev.	Produces	Example
<pre>xref:rgsvc.adoc#xxx[ttt]</pre> a hyperlink to a bookmark within the reference guide for domain services, where: * <code>xxx</code> is the bookmark * <code>ttt</code> is the text to display as the hyperlink for example: <pre>xref:rgsvc.adoc#_rgcms_classes_AppManifest-bootstrapping[`AppManifest` bootstrapping]</pre>	`AppManifest` bootstrapping	`adrgmvn`

Abbrev.	Produces	Example
<pre>xref:rgmvn.adoc#xxx[ttt]</pre> a hyperlink to a bookmark within the reference guide for the maven plugin, where: * <code>xxx</code> is the bookmark * <code>ttt</code> is the text to display as the hyperlink for example: <pre>xref:rgmvn.adoc#_rgmvn_validate[validate goal]</pre>	validate goal	`adrgna`

Abbrev.	Produces	Example
<pre>xref:rgant.adoc#_rgant-xxx[`@xxx`</pre>] a hyperlink to the "man page" for an annotation within the reference guide for annotations, where: * <code>xxx</code> is the annotation type (eg <code>@Action</code>) for example: <pre>xref:rgant.adoc#_rgant-Action[`@Action`]</pre>	`@Action`	`adrgnt`

Abbrev.	Produces	Example
<pre>xref:rgant.adoc#_rgant-xxx_ttt[`@xxx#ttt()`</pre>] a hyperlink to the "man page" for the specific attribute (field) of an annotation within the reference guide for annotations, where: * <code>xxx</code> is the annotation type (eg <code>@Action</code>) * <code>ttt</code> is the attribute (eg <code>@semantics</code>) for example: <pre>xref:rgant.adoc#_rgant-Action_semantics[`@Action#semantics()`]</pre>	`@Action#semantics()`	`adrgsa`

Abbrev.	Produces	Example
<pre></pre> a hyperlink to the "man page" for an (API) domain service within the reference guide for domain services, where: * <code>xxx</code> is the domain service (eg <code>DomainObjectContainer</code>) for example: <pre>xref:rgsvc.adoc#_rgsvc_api_DomainObjectContainer[`DomainObjectContainer`]</pre>	`DomainObjectContainer`	`adrgss`

Abbrev.	Produces	Example
<pre></pre> a hyperlink to the "man page" for an (SPI) domain service within the reference guide for domain services, where: * <code>xxx</code> is the domain service (eg <code>ContentMappingService</code>) for example: <pre>xref:rgsvc.adoc#_rgsvc_spi_ContentMappingService[`ContentMappingService`]</pre>	`ContentMappingService`	`adugfun`

Abbrev.	Produces	Example
<pre>xref:ugfun.adoc#xxx[ttt]</pre> a hyperlink to a bookmark within the fundamentals users' guide, where: * <code>xxx</code> is the bookmark’s anchor * <code>ttt</code> is the text to display as the hyperlink for example: <pre>xref:ugfun.adoc#_ugfun_core-concepts[Core concepts]</pre>	Core concepts	`adugvw`

Abbrev.	Produces	Example
<pre>xref:ugvw.adoc#xxx[ttt]</pre> A hyperlink to a bookmark within the Wicket viewer guide, where: * <code>xxx</code> is the bookmark’s anchor * <code>ttt</code> is the text to display as the hyperlink. for example: <pre>xref:ugvw.adoc#_ugvw_customisation[Customisation]</pre>	Customisation	`adugvro`

Abbrev.	Produces	Example
<pre>xref:ugvro.adoc#xxx[ttt]</pre> A hyperlink to a bookmark within the Restful Objects viewer guide, where: * <code>xxx</code> is the bookmark’s anchor * <code>ttt</code> is the text to display as the hyperlink. for example: <pre>xref:ugvro.adoc#_ugvro_ro-spec[Restful Objects specification]</pre>	RestfulObjects specification	`adugsec`

Abbrev.	Produces	Example
<pre>xref:ugsec.adoc#xxx[ttt]</pre> A hyperlink to a bookmark within the Secrurity guide, where: * <code>xxx</code> is the bookmark’s anchor * <code>ttt</code> is the text to display as the hyperlink. for example: <pre>xref:ugsec.adoc#_ugsec_shiro-caching[Caching and other Shiro Features]</pre>	Caching and other Shiro Features	`adugtst`

Abbrev.	Produces	Example
<pre>xref:ugtst.adoc#xxx[ttt]</pre> A hyperlink to a bookmark within the Testing guide, where: * <code>xxx</code> is the bookmark’s anchor * <code>ttt</code> is the text to display as the hyperlink. for example: <pre>xref:ugtst.adoc#_ugtst_bdd-spec-support[BDD Spec Support]</pre>	BDD Spec Support	adugbtb

8.4. Link to Isis Addons

Links to (non-ASF) Isis Addons

Abbrev.	Produces	Example
adlinkaddons	<pre>(non-ASF) link:http://isisaddons.org[Isis Addons]</pre> link to the Isis Addons website.	(non-ASF) Isis Addons
adlinkaddonsapp	<pre>(non-ASF) http://github.com/isisaddons/isis-app-xxx[Isis addons' xxx</pre>] link to the github repo for an example app from the Isis addons; where: * <code>xxx</code> is the name of the example app being linked to for example: <pre>(non-ASF) http://github.com/isisaddons/isis-app-todoapp[Isis addons' todoapp]</pre>	(non-ASF) Isis addons' todoapp

Abbrev.	Produces	Example
adlinkaddons module	<pre></pre> link to the github repo for a module from the Isis addons; where: * <code>xxx</code> is the name of the module being linked to for example: <pre>(non-ASF) http://github.com/isisaddons/isis-module-security[Isis addons' security] module</pre>	(non-ASF) Isis addons' security module
adlinkaddons wicket	<pre></pre> link to the github repo for a wicket UI component from the Isis addons; where: * <code>xxx</code> is the name of the wicket UI component being linked to for example: <pre>(non-ASF) http://github.com/isisaddons/isis-wicket-gmap3[Isis addons' gmap3] wicket extension</pre>	(non-ASF) Isis addons' gmap3 wicket extension

8.5. Source code

Abbrev.	Produces	Example
adsrcjava	[source,java] ---- xxx ---- where: * xxx is the source code snippet.	[source,java] ---- public class Foo { ... } ----
adsrcjavac	as for adsrcjava, but with a caption above	
adsrcjavascript	[source,javascript] ---- xxx ---- where: * xxx is the source code snippet.	[source,javascript] ---- $(document).ready(function() { ... }); ----
adsrcjavascriptc	as for adsrcjavascript, but with a caption above	
adsrcother	[source,nnn] ---- xxx ---- where: * nnn is the programming language * xxx is the source code snippet.	
adsrcotherc	as for adsrcother, but with a caption above	
adsrcxml	[source,javascript] ---- xxx ---- where: * xxx is the source code snippet.	[source,xml] ---- <html> <title> hello world! </title> </html> ----
adsrcxmlc	as for adsrcxml, but with a caption above	

8.6. Images

Abbrev.	Produces	Example
adimgfile	<pre>image::{_imagesdir}xxx/yyy.png[width="WWWpx",link="{_imagesdir}xxx/yyy.png"]</pre> embeds specified image, where: * <code>xxx</code> is the subdirectory under the <code>images/</code> directory * <code>yyy</code> is the image * <code>WWW</code> is the width, in pixels. for example: <pre>image::{_imagesdir}wicket-viewer/layouts/estatio-Lease.png[width="300px",link="{_imagesdir}wicket-viewer/layouts/estatio-Lease.png"]</pre>	image::images/wicket-viewer/layouts/estatio-Lease.png[width="300px",link="images/wicket-viewer/layouts/estatio-Lease.png"]
adimgfilec	as for adimgfile, but with a caption above	
adimgurl	<pre>image::xxx[width="WWWpx",link="xxx"]</pre> embeds image from specified URL, where: * <code>xxx</code> is the URL to the image * <code>WWW</code> is the width, in pixels.	
adimgurlc	as for adimgurl, but with a caption above	

8.7. YouTube (screencasts)

Embedded youtube screencasts. (Don't use these in guides, as they cannot be rendered as PDF).

Abbrev.	Produces	Example
adyoutube	<pre>video::xxx[youtube,width="WWWpx",height="HHHpx"]</pre> where: * <code>xxx</code> is the youtube reference * <code>WWW</code> is the width, in pixels * <code>HHH</code> is the height, in pixels for example: <pre>video::bj8735nBRR4[youtube,width="210px",height="118px"] </pre>	video::bj8735nBRR4[youtube,width="210px",height="118px"]
adyoutubec	as for youtube, but with a caption above	

8.8. Tables

Abbrev.	Produces	Example
adtbl3	Table with 3 columns, 3 rows.	

8.9. Misc.

Abbrev.	Produces	Example
adai	<pre>Apache Isis</pre> That is, the literal text "Apache Isis".	Apache Isis

Abbrev.	Produces	Example
adlink	<pre>link:xxx[ttt]</pre> , where: * <code>xxx</code> is * <code>ttt</code> is the text to display as the hyperlink for example: <pre>link:http://isis.apache.org[Apache Isis website]</pre>	Apache Isis website
adanchany	<pre>= anchor:[xxx]</pre> defines an inline anchor to any heading, where: * <code>xxx</code> is the anchor text. For example: <pre>= anchor:[_ugfun_i18n] Internationalization</pre> An alternative (more commonly used in our documentation) is to use the <code>[[…]]</code> directly above the heading: <pre>[[_ugfun_i18n]] = Internationalization</pre>	
adxrefany	<pre>xref:[xxx]</pre> cross-reference to any document/anchor, where: * <code>xxx</code> is the fully qualified document with optional anchor	
adfootnote	<pre>.footnote:[]</pre> defines a footnote	. [1: this is a footnote]

Chapter 9. Appendix: Project Lombok

Project Lombok is an open source project to reduce the amount of boilerplate in your code.

For example, rather than write:

```
private String name;
public String getName() {
    return name;
}
public void setName(String name) {
    this.name = name;
}
```

you can instead write simply:

```
@Getter @Setter
private String name;
```

Under the covers it is implemented as an annotation processor; it basically hooks into the Java compiler so that it can emit additional bytecode (eg for the getter and setter). See here for details of setting up in IntelliJ (Eclipse has very similar support).

Apache Isis supports Project Lombok, in that the annotations that would normally be placed on the getter (namely `Property`, `@PropertyLayout`, `@Collection`, `@CollectionLayout` and `@MemberOrder`) can be placed on the field instead.

There are plugins for Lombok for maven; it's just a matter of adding the required dependency. To compile the code within your IDE (eg so that its compiler "knows" that there is, actually, a getter and setter) will require an Lombok plugin appropriate to that IDE. See the Lombok download page for more information.

9.1. Future thoughts

In the future we might extend/fork Lombok so that it understands Isis' own annotations (ie `@Property` and `@Collection`) rather than Lombok's own `@Getter` and `@Setter.

It might also be possible to use Lombok to generate the domain event classes for each member.

Chapter 10. Appendix: AgileJ

This material does not constitute an endorsement; AgileJ Structure Views is not affiliated to Apache Software Foundation in any way.

AgileJ Structure Views is a commercial product to reverse engineer and visualize Java classes from source code.

The key to using the tool is in developing a suitable filter script, a DSL. You can use the following script as a starting point for visualizing Apache Isis domain models:

```
// use CTRL+SPACE for completion suggestions
hide all fields
hide setter methods
hide private methods
hide methods named compareTo
hide methods named toString
hide methods named inject*
hide methods named disable*
hide methods named default*
hide methods named hide*
hide methods named autoComplete*
hide methods named choices*
hide methods named title
hide methods named iconName
hide methods named validate*
hide methods named modify*
hide protected methods
hide types annotated as DomainService
hide types named Constants
hide types named InvoicingInterval
hide enums
hide constructors
hide inner types named *Event
hide inner types named *Functions
hide inner types named *Predicates
show getter methods in green
show methods annotated as Programmatic in orange
show methods annotated as Action in largest
hide dependency lines
hide call lines
hide method lines
```

For more information on AgileJ, see Paul Wells' 8-part tutorial series on Youtube; the first can be found here (view the "show more" comments to click through to other parts).