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 ......................................................................................... 38
3. Building Apache Isis ........................................................................................................... 45
   3.1. Git ............................................................................................................................. 45
   3.2. Installing Java ......................................................................................................... 47
   3.3. Installing Maven ....................................................................................................... 48
   3.4. Building all of Apache Isis ...................................................................................... 49
   3.5. Checking for Vulnerabilities .................................................................................. 49
   3.6. Checking for use of internal JDK APIs .................................................................. 49
4. Asciidoc Documentation ..................................................................................................... 51
   4.1. Where to find the Docs ......................................................................................... 51
   4.2. Naming Conventions .............................................................................................. 51
   4.3. Writing the docs ...................................................................................................... 52
   4.4. Build and Review (using Maven) ........................................................................ 52
   4.5. Instant Rebuild (using Ruby) ................................................................................ 52
   4.6. Publish procedure .................................................................................................. 53
5. Contributing .......................................................................................................................... 54
   5.1. Recommended Workflow (github) ........................................................................ 54
   5.2. Alternative Workflow (JIRA patches) ................................................................... 55
   5.3. Setting up your fork/clone ..................................................................................... 56
   5.4. Commit messages .................................................................................................... 56
   5.5. Creating the patch file ............................................................................................. 57
   5.6. Sample Contribution Workflow ............................................................................ 57
   5.7. If your pull request is accepted .............................................................................. 59
6. Appendix: Git Cookbook ...................................................................................................... 61
   6.1. Modifying existing files .......................................................................................... 61
   6.2. Adding new files ...................................................................................................... 61
   6.3. Deleting files ............................................................................................................ 61
   6.4. Renaming or moving files ...................................................................................... 62
   6.5. Common Workflows ............................................................................................... 62
   6.6. Backing up a local branch ..................................................................................... 62
   6.7. Quick change: stashing changes .......................................................................... 62
   6.8. Ignoring files ........................................................................................................... 63
   6.9. More advanced use cases ....................................................................................... 63
   6.10. If you’ve accidentally worked on *master* branch ............................................ 64
6.11. If you've forgotten to prefix your commits (but not pushed) ........................................ 64

7. Appendix: Working with Many Repos  ................................................................. 66
   7.1. Prerequisites ................................................................. 66
   7.2. _repos.txt ............................................................... 66
   7.3. Bash functions ......................................................... 67

8. Appendix: Asciidoc Templates ................................................................. 69
   8.1. Callouts ................................................................. 69
   8.2. TODO notes ............................................................ 69
   8.3. Xref to Guides .......................................................... 69
   8.4. Link to Isis Addons ................................................... 84
   8.5. Source code ............................................................ 85
   8.6. Images ................................................................. 85
   8.7. YouTube (screencasts) ............................................... 86
   8.8. Tables ................................................................. 86
   8.9. Misc. ................................................................. 86

9. Appendix: Project Lombok ................................................................. 88
   9.1. Future thoughts .......................................................... 88

10. Appendix: AgileJ ................................................................. 89
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
- DataNucleus object store
- Security
- Testing
- Beyond the Basics

The reference guides are:

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

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. JetBrains does however provide complimentary copies of the IntelliJ IDE to Apache committers.

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/screenshots were prepared using IntelliJ Community Edition 14.1.x, but are believed to be compatible with more recent versions/other editions of the IDE.

Download and Install

Download latest version of IntelliJ Community Edition, and install:

Start the wizard, click through the welcome page:

![IntelliJ IDEA Installation Wizard - Welcome page](image)

Choose the location to install the IDE:
Adjust any installation options as you prefer:

and the start menu:
and finish up 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:
Finally, if you are a trendy hipster, set the UI theme to Darcula:

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. Apache Isis supports both Java 7 and Java 8.
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:
File templates

Next we recommend you import a set of standard file templates. These are used to create new classes or supporting files:

The file templates are provided as a settings JAR file, namely `isis-settings-file-templates.jar`. Download this file.

Next, import using `File > Import Settings`, specifying the directory that you have downloaded the file to:
Select all the categories (there should just be one), and hit OK. then hit restart.

Apparently no categories are shown if importing into IntelliJ 2016.1.1 Community Edition (and perhaps other 2016 versions). The file does import ok into IntelliJ 15.0.x, so we think this is a bug in the 2016 version.

The workaround is to extract the .jar file locally and copy the files into IntelliJ's config directory, somewhere in your home directory:

- Windows `<User home>\.IdeaIC2016\config`
- Linux `~/.IdeaIC2016/config`
- Mac OS `~/Library/Preferences/IdeaIC2016`

**Live templates**

We also recommend you import a set of live templates. These are used to add new methods to existing classes:

```
public class CustomerOrder implements Comparable<CustomerOrder> {
```

![Live templates](image)

*Figure 15. Live templates*
The live templates have a prefix of prefixed either:

- **is** : for Apache Isis domain objects
- **ju** : for JUnit tests
- **jm** : for JMock mocks or libraries
- **ad** : for Asciidoc documentation; a full list can be found in the appendix.

The live templates are also provided as a settings JAR file, namely *isis-settings-live-templates.jar*. Download and import (as for the previous settings JAR files).

**Coding Standards**

Next, we suggest you recommend you import settings for standard ASF/Apache Isis coding conventions. This file is also provided as a settings file, namely *isis-settings-code-style.jar*. Download and import (as for the above settings JAR files).

**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**):

![IntelliJ Compiler Settings](image)

*Figure 16. IntelliJ Compiler Settings*

On the **Annotation Processors** page, enable and adjust for the 'default' setting:
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):
Still on the Maven settings page, configure as follows:
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.
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:
We can then import another module (from some other directory). For example, here we are importing the Isis Addons' todoapp example:

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

![IntelliJ Maven Module Management](image)

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:
Confirm that it’s ok to ignore these modules:

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):
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:
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:

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 (!).

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

Also 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.

### 2.1.4. 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 together
- 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
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
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. Faster turnaround times

In this section are several options that will reduce the time it takes between making a source code edit and seeing the results in the running app. code/build/deploy/review feedback loop.

Using Grade to compile/enhance

Running an Apache Isis application requires that the DataNucleus enhancer runs on the compiled bytecode. As described above, the recommended way to do this with IntelliJ is to use a Run configuration that runs the enhancer goal prior to launch.

Alternative, you can use the following build.gradle script in your dom module:
The script is intended to be in the background as a daemon while editing/developing; whenever a change is made to any source code, gradle will automatically compile and enhance the code. In this way it eliminates the need to start up Maven and run the enhancer goal.

To use, you must disable the IntelliJ’s automatic building of the ‘dom’ project. This is done using: File > Settings > Build, Execution, Deployment > Compiler > Excludes, and then exclude the …/dom/src/main/java directory:
The script can be run in the background using:

```bash
gradle -t --offline &
```

from the command line (in the `dom` module).

**Using Gradle for liveReload**

Similarly, gradle can be run to reduce the turn-around time when tweaking the UI (defined by the `*.layout.xml` file for each domain class), when the app is running.

The framework will automatically notice any changes to `.layout.xml` files, but these are read from the the classpath (the `target/classes` directory), not the source path. With IntelliJ these can be
copied over manually by invoking Run > Reload Changed Classes. Once the browser is refreshed, the new layout will be rendered.

We’ve occasionally noticed that this interferes with Wicket’s own javascript - switching tabs becomes unresponsive. The work-around is just to reload the page.

To reduce the turn-around time there are therefore two steps to be automated:

• the copying of the .layout.xml files over to the target/classes directory
• the triggering of a page refresh by the browser.

The layouts.gradle script takes care of the first of these; whenever a change is made to any .layout.xml file, gradle will automatically copy over the file to the target/classes directory:

```gradle
defaultTasks 'copyLayouts'
task copyLayouts(type:Copy) {
    from 'src/main/java'
    into 'target/classes'
    include '**/*.layout.xml'
}
```

Similarly, the liveReload.gradle script takes care of the browser refresh:

```gradle
defaultTasks 'liveReload'
buildscript {
    repositories {
        jcenter()
    }
    dependencies {
        classpath 'org.kordamp.gradle:livereload-gradle-plugin:0.2.1'
    }
}
apply plugin: 'org.kordamp.gradle.livereload'
liveReload {
    docRoot new File('target/classes').canonicalPath
}
```

These scripts can be run together using:

```
gradle -t --offline -b layouts.gradle &
gradle -t --offline -b liveReload.gradle &
```

from the command line (in the dom module):
Live reload also requires that the `isis.viewer.wicket.liveReloadUrl` configuration property is set appropriately:

```
viewer_wicket.properties

isis.viewer.wicket.liveReloadUrl=http://localhost:35729/livereload.js?snipver=1
```

You can confirm the script is loaded correctly using the web browser's development tools, eg:
Setting up DCEVM

**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](https://github.com), and run the installer. For example:

```sh
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. Editor Templates

We provide a set of editor templates. These are used to add new methods to existing classes. (These are equivalent to the IntelliJ live templates):

- is (Apache Isis domain objects). Download
- ju (for JUnit tests) Download
- jm (for JMock mocks or libraries) Download

To install, download each XML file, then go to Windows > Preferences > Java > Editor > Templates and choose Import.

2.2.3. Importing the Project

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

2.2.4. 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:

![Console output](image)

**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 :-)

---

Exception in thread "main" java.lang.ClassFormatError: Duplicate field name and signature in class file com/eu

at java.lang.ClassLoader.defineClass1(Native Method)
at java.lang.ClassLoader.defineClass0(Native Method)
at java.security.SecureClassLoader.defineClass(Unknown Source)
at java.net.URLClassLoader.defineClass(Unknown Source)
at java.net.URLClassLoader.access$000(Unknown Source)
at java.net.URLClassLoader$1.run(Unknown Source)
at java.security.AccessController.doPrivileged(Native Method)
at java.net.URLClassLoader.findClass(Unknown Source)
at java.lang.ClassLoader.loadClass(Unknown Source)
at sun.misc.Launcher$AppClassLoader.loadClass(Unknown Source)
at java.lang.ClassLoader.loadClass(Unknown Source)
at java.lang.Class.forName0(Native Method)
at java.lang.Class.forName(Unknown Source)
at org.datanucleus.JDOClassLoaderResolver.classOrNull(JDOClassLoaderResolver.java:548)
at org.datanucleus.JDOClassLoaderResolver.classForName(JDOClassLoaderResolver.java:280)
at org.datanucleus.JDOClassLoaderResolver.classForName(JDOClassLoaderResolver.java:410)
at org.datanucleus.metadata.MetaDataManager.loadPersistenceUnit(MetaDataManager.java:954)
at org.datanucleus.enhancer.DataNucleusEnhancer.getFileMetadataForInput(DataNucleusEnhancer.java:132)
at org.datanucleus.enhancer.DataNucleusEnhancer.enhance(DataNucleusEnhancer.java:525)
at org.datanucleus.enhancer.DataNucleusEnhancer.main(DataNucleusEnhancer.java:1258)
2.2.5. 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.

2.2.6. 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.

2.2.7. 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 DCEVM

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. Building Apache Isis

3.1. Git

The Apache Isis source code lives in a git repo.

3.1.1. 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.
3.1.2. 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`

3.2. 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 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.

3.2.1. 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:
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.

### 3.3. Installing Maven

Install Maven 3.0.x, downloadable [here](#). Set MAVEN_OPTS environment variable:

```bash
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).
3.4. 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.

3.5. 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 `--offline` or `--offline`) or omit the `owasp` property.

3.6. 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 4. 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).

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

Publishing is performed by copying the generated HTML to a different git repository (isis-site). Since this can only be done by Apache Isis committers, the process for doing this is described in the committers’ guide. This is synced by ASF infrastructure over to isis.apache.org.

4.1. 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.

4.2. 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/
```
4.3. 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.

4.4. 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.

4.5. 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).

4.5.1. 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
```

4.5.2. 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
```

### 4.6. Publish procedure

Only Apache Isis committers can publish to `isis.apache.org`. See the [committers' guide](#) for further details.
Chapter 5. 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!

5.1. 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.

5.2. 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 commuters it really doesn’t matter which route you take, so go with whatever’s most comfortable.
5.3. 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](#).

5.4. 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.

5.5. 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.

5.6. 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.

5.6.1. 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 in either case, the same commands work:

```
Alternatively, you can combine the `git fetch` and `git merge` and just use `git pull`:
```

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.

5.6.2. 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:

```bash
$ 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.

5.6.3. 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`).

5.6.4. 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.

5.6.5. 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.

5.7. 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:

5.7.1. 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
Chapter 6. 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

6.1. 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"
```

6.2. 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.

6.3. Deleting files

To delete a file:
git rm filename
git commit -m "ISIS-nnn: yada yada"

6.4. 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"
```

6.5. Common Workflows

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

6.6. 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.

6.7. 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

6.8. 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

6.9. More advanced use cases

6.9.1. 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:
Then, do a force push:

```bash
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](#).

### 6.10. 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 `git --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:

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

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

```bash
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:

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

### 6.11. 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:
You can grab this utility, and others, from this repo.
Chapter 7. Appendix: Working with Many Repos

Applications built with Apache Isis often (should) consist of multiple modules, each in separate git repositories. For example, there are the various (non-ASF) Isis Addons that provide various cross-cutting concerns/technical services (security, mail-merge etc), as well the modules eg as provided by the (non-ASF) Incode Catalog, which provides generic business subdomains (communication channels, documents etc).

In addition, your own application may well be structured as a number of distinct modules (probably with the entities in each module being mapped to a different schema), and using such techniques as the event bus and mixins so that these modules are decoupled from each other.

All of which is a preamble to say that you will likely have multiple directories on your local development computer, for each such git repository that you contribute to.

In this appendix we provide some simple but useful bash scripts to help you manage each such.

7.1. Prerequisites

We recommend that you adopt a convention for your directories. For example, open source repositories (such as the Isis Addons and Incode Catalog mentioned above) reside in github.com, while your own proprietary code might reside in some other service, eg bitbucket. For example:

```
/users/home/me/
  BITBUCKET/
    mycompany/
      myapp-moduleA
      myapp-moduleB
      myapp
  GITHUB/
    isisaddons/
      isis-module-docx
      isis-module-security
    incodehq/
      incode-module-commchannel
      incode-module-classification
```

7.2. _repos.txt

Create a file _repos.txt that catalogues the repositories, eg:
7.3. Bash functions

The `.bash_functions` file (downloadable from this gist) provides the following two functions:

- **repo**

  Switches (using `pushd`) to the specified directory (as listed in the `_repos.txt` file).

- **foreach**

  Runs the specified command for all (or matching) repositories (as listed in `_repos.txt` file).

For example,

```bash
repo doc
```

would switch to `/users/home/me/GITHUB/isisaddons/isis-module-docx`, the first module that matches the fragment.

Meanwhile:

```bash
foreach git status
```

would perform a `git status` on every git repository, while

```bash
foreach -g isisaddons git fetch
```

would perform a `git fetch` but only to those repositories which match "isisaddons" (-g flag standing for `grep`).

To load the functions into your profile (`.bashrc` or `.profile` or similar), use:
. ~/.bash_functions
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".

<table>
<thead>
<tr>
<th>Abbrev.</th>
<th>Produces</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>adadminporta nt</td>
<td><img src="adadminporta%20nt" alt="IMPORTANT" /></td>
<td><img src="adadminporta%20nt" alt="IMPORTANT" /></td>
</tr>
<tr>
<td>adadminnote</td>
<td><img src="adadminnote" alt="NOTE" /></td>
<td><img src="adadminnote" alt="NOTE" /></td>
</tr>
<tr>
<td>adadmtip</td>
<td><img src="adadmtip" alt="TIP" /></td>
<td><img src="adadmtip" alt="TIP" /></td>
</tr>
<tr>
<td>adadmwarning</td>
<td><img src="adadmwarning" alt="WARNING" /></td>
<td><img src="adadmwarning" alt="WARNING" /></td>
</tr>
</tbody>
</table>

### 8.2. TODO notes

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

<table>
<thead>
<tr>
<th>Abbrev.</th>
<th>Produces</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>adtodo</td>
<td><img src="adtodo" alt="pre" /> NOTE: TODO&lt;/pre&gt;</td>
<td>NOTE: TODO</td>
</tr>
<tr>
<td>adwip</td>
<td><img src="adwip" alt="pre" /> NOTE: WIP - xxx&lt;/pre&gt; where: *&lt;br/&gt;<img src="adwip" alt="code" /> xxx&lt;/code&gt; is additional explanatory text</td>
<td>NOTE: WIP - cool new feature</td>
</tr>
</tbody>
</table>

### 8.3. Xref to Guides

Cross-references (links) to the various guides
<table>
<thead>
<tr>
<th>Abbrev.</th>
<th>Produces</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>adcgcom</td>
<td><code>&lt;pre&gt;xref:cgcom.adoc#xxx[ttt]&lt;/pre&gt;</code> a hyperlink to a bookmark within the committers' guide, where: * <code>&lt;code&gt;xxx&lt;/code&gt;</code> is the bookmark's anchor * <code>&lt;code&gt;ttt&lt;/code&gt;</code> is the text to display as the hyperlink for example: <code>&lt;pre&gt;xref:dg.adoc#_cgcom_cutting-a-release[Cutting a release]&lt;/pre&gt;</code></td>
<td>addg</td>
</tr>
<tr>
<td></td>
<td>&lt;pre&gt;xref:dg.adoc#xxx[ttt]&lt;/pre&gt; a hyperlink to a bookmark within the developers' guide, where: * <code>&lt;code&gt;xxx&lt;/code&gt;</code> is the bookmark's anchor * <code>&lt;code&gt;ttt&lt;/code&gt;</code> is the text to display as the hyperlink for example: <code>&lt;pre&gt;xref:dg.adoc#_dg_asciidoc-templates[Asciidoc templates]&lt;/pre&gt;</code></td>
<td>adrgant</td>
</tr>
<tr>
<td>Abbrev.</td>
<td>Produces</td>
<td>Example</td>
</tr>
<tr>
<td>--------</td>
<td>----------</td>
<td>---------</td>
</tr>
<tr>
<td><code>&lt;pre&gt;xref:rg ant.adoc#xx x[ttt]&lt;/pre&gt;</code></td>
<td>Core annotations</td>
<td>adrgcfg</td>
</tr>
</tbody>
</table>

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:rg ant.adoc#_rg ant-aaa_main[Core annotations]</pre>
```
<table>
<thead>
<tr>
<th>Abbrev.</th>
<th>Produces</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;pre&gt;xref:rg cfg.adoc#xxx [ttt]&lt;/pre&gt;</td>
<td>Configuring Core</td>
<td>adrgcms</td>
</tr>
</tbody>
</table>

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:rg cfg.adoc#_rg cfg_configuring-core[Configuring Core]</pre>
<table>
<thead>
<tr>
<th>Abbrev.</th>
<th>Produces</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;pre&gt;xref:rg cms.adoc#xx x[ttt]&lt;/pre&gt;</code></td>
<td><em>AbstractService</em></td>
<td><code>adrgsvc</code></td>
</tr>
</tbody>
</table>

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:

```markdown
<pre>xref:rg cms.adoc#_rgcms_classes_super_AbstractService[`AbstractService`]</pre>
```
<table>
<thead>
<tr>
<th>Abbrev.</th>
<th>Produces</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;pre&gt;xref:rg svc.adoc#xxx [ttt]&lt;/pre&gt;</td>
<td>AppManifest bootstrapping</td>
<td>adrgmvn</td>
</tr>
</tbody>
</table>

A hyperlink to a bookmark within the reference guide for domain services, where:
* `xxx` is the bookmark
* `ttt` is the text to display as the hyperlink for example:

```
<pre>xref:rg svc.adoc#_rg cms_classes_AppManifest -bootstrappi ng[`AppManifest` bootstrappin g]</pre>
```
<table>
<thead>
<tr>
<th>Abbrev.</th>
<th>Produces</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;pre&gt;xref:rg mvn.adoc#x xx[ttt]&lt;pre&gt;</code> a hyperlink to a bookmark within the reference guide for the maven plugin, where: * <code>&lt;code&gt;xxx&lt;/code&gt;</code> is the bookmark * <code>&lt;code&gt;ttt&lt;/code&gt;</code> is the text to display as the hyperlink for example: <code>&lt;pre&gt;xref:rg mvn.adoc#_rgmvn_validat evalidate goal&lt;/pre&gt;</code></td>
<td><code>validate goal</code></td>
<td><code>adr gna</code></td>
</tr>
<tr>
<td>Abbrev.</td>
<td>Produces</td>
<td>Example</td>
</tr>
<tr>
<td>--------</td>
<td>----------</td>
<td>---------</td>
</tr>
<tr>
<td><code>&lt;pre&gt;xref:rg ant.adoc#_rg ant-xxx[</code>@xxx` &lt;/pre&gt;]</td>
<td>@Action</td>
<td><code>adrgnt</code></td>
</tr>
<tr>
<td>Abbrev.</td>
<td>Produces</td>
<td>Example</td>
</tr>
<tr>
<td>---------</td>
<td>----------</td>
<td>---------</td>
</tr>
<tr>
<td><code>&lt;pre&gt;xref:rg ant.adoc#_rg ant-xxx_ttt[</code>@xx x#ttt()<code> &amp;lt;/pre&amp;gt;</code>] a hyperlink to the &quot;man page&quot; for the specific attribute (field) of an annotation within the reference guide for annotations, where: * <code>&lt;code&gt;xxx&lt;/code&gt;</code> is the annotation type (eg <code>&lt;code&gt;@Action&lt;/code&gt;</code>) * <code>&lt;code&gt;ttt&lt;/code&gt;</code> is the attribute (eg <code>&lt;code&gt;@semantics&lt;/code&gt;</code>) for example: <code>&lt;pre&gt;xref:rg ant.adoc#_rg ant-Action_semantics[</code>@Action#semantics<code>()]&lt;/pre&gt;</code></td>
<td><code>@Action#semantics()</code></td>
<td><code>adrgsa</code></td>
</tr>
<tr>
<td>Abbrev.</td>
<td>Produces</td>
<td>Example</td>
</tr>
<tr>
<td>---------</td>
<td>----------</td>
<td>---------</td>
</tr>
<tr>
<td>&lt;pre&gt;&lt;/pre&gt; a hyperlink to the &quot;man page&quot; for an (API) domain service within the reference guide for domain services, where: * &lt;code&gt;xxx&lt;/code&gt; is the domain service (eg &lt;code&gt;DomainObjectContainer&lt;/code&gt; ) for example: &lt;pre&gt;xref:rg svc.adoc#_rg svc_api_DomainObjectContainer[<code>DomainObjectContainer</code>]&lt;/pre&gt;</td>
<td>DomainObjectContainer</td>
<td>addrss</td>
</tr>
<tr>
<td>Abbrev.</td>
<td>Produces</td>
<td>Example</td>
</tr>
<tr>
<td>--------</td>
<td>--------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td><code>&lt;pre&gt;</code></td>
<td>a hyperlink to the &quot;man page&quot; for an (SPI) domain service within the reference guide for domain services, where: * <code>&lt;code&gt;xxx&lt;/code&gt;</code> is the domain service (eg <code>&lt;code&gt;ContentMappingService&lt;/code&gt;</code>) for example: <code>&lt;pre&gt;xref:rg svc.adoc#_rg svc_spi_ContentMappingService[</code>ContentMappingService<code>]&lt;/pre&gt;</code></td>
<td>adugfun</td>
</tr>
<tr>
<td>Abbrev.</td>
<td>Produces</td>
<td>Example</td>
</tr>
<tr>
<td>--------</td>
<td>----------</td>
<td>---------</td>
</tr>
<tr>
<td>&lt;pre&gt;xref:ug fun.adoc#xx x[ttt]&lt;/pre&gt;</td>
<td>a hyperlink to a bookmark within the fundamental users' guide, where: * &lt;code&gt;xxx&lt;/code&gt; is the bookmark's anchor * &lt;code&gt;ttt&lt;/code&gt; is the text to display as the hyperlink for example: &lt;pre&gt;xref:ug fun.adoc#_ugfun_core-concepts[Core concepts]&lt;/pre&gt;</td>
<td>adugvw</td>
</tr>
<tr>
<td>Abbrev.</td>
<td>Produces</td>
<td>Example</td>
</tr>
<tr>
<td>--------</td>
<td>----------</td>
<td>---------</td>
</tr>
<tr>
<td><code>&lt;pre&gt;xref:ug vw.adoc#xxx [ttt]&lt;/pre&gt;</code></td>
<td>Customisation</td>
<td>adugvro</td>
</tr>
<tr>
<td>Abbrev.</td>
<td>Produces</td>
<td>Example</td>
</tr>
<tr>
<td>---------</td>
<td>-----------</td>
<td>---------</td>
</tr>
<tr>
<td><code>&lt;pre&gt;xref:ug vro.adoc#xx x[ttt]&lt;/pre&gt;</code></td>
<td>RestfulObjects specification</td>
<td>adugsec</td>
</tr>
<tr>
<td>Abbrev.</td>
<td>Produces</td>
<td>Example</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td><code>&lt;pre&gt;xref:ug sec.adoc#xxx [ttt]&lt;/pre&gt;</code></td>
<td>A hyperlink to a bookmark within the Security guide, where: * <code>&lt;code&gt;xxx&lt;/code&gt;</code> is the bookmark’s anchor * <code>&lt;code&gt;ttt&lt;/code&gt;</code> is the text to display as the hyperlink. For example: <code>&lt;pre&gt;xref:ug sec.adoc#_ug sec_shiro-caching[Caching and other Shiro Features]&lt;/pre&gt;</code></td>
<td><code>adugtst</code></td>
</tr>
</tbody>
</table>
**8.4. Link to Isis Addons**

Links to (non-ASF) Isis Addons

<table>
<thead>
<tr>
<th>Abbrev.</th>
<th>Produces</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>adlinkaddons</td>
<td>&lt;pre&gt;(non-ASF) link:<a href="http://isisaddons.org%5BIsis">http://isisaddons.org[Isis</a> Addons]&lt;/pre&gt; link to the Isis Addons website.</td>
<td>(non-ASF) Isis Addons</td>
</tr>
<tr>
<td>adlinkaddons</td>
<td>&lt;pre&gt;(non-ASF) <a href="http://github.com/isisaddons/isis-app-xxx%5BIsis">http://github.com/isisaddons/isis-app-xxx[Isis</a> addons' xxx]&lt;/pre&gt; link to the github repo for an example app from the Isis addons; where: * &lt;code&gt;xxx&lt;/code&gt; is the name of the example app being linked to for example: &lt;pre&gt;(non-ASF) <a href="http://github.com/isisaddons/isis-app-todoapp%5BIsis">http://github.com/isisaddons/isis-app-todoapp[Isis</a> addons' todoapp]&lt;/pre&gt;</td>
<td>(non-ASF) Isis addons' todoapp</td>
</tr>
<tr>
<td>Abbrev.</td>
<td>Produces</td>
<td>Example</td>
</tr>
<tr>
<td>--------</td>
<td>----------</td>
<td>---------</td>
</tr>
<tr>
<td><strong>adlinkaddons module</strong></td>
<td>&lt;pre&gt;&lt;/pre&gt; link to the github repo for a module from the Isis addons; where: * &lt;code&gt;xxx&lt;/code&gt; is the name of the module being linked to for example: &lt;pre&gt;(non-ASF) <a href="http://github.com/isisaddons/isis-module-security%5BIsis">http://github.com/isisaddons/isis-module-security[Isis</a> addons' security] module&lt;/pre&gt;</td>
<td>(non-ASF) Isis addons' security module</td>
</tr>
<tr>
<td><strong>adlinkaddons wicket</strong></td>
<td>&lt;pre&gt;&lt;/pre&gt; link to the github repo for a wicket UI component from the Isis addons; where: * &lt;code&gt;xxx&lt;/code&gt; is the name of the wicket UI component being linked to for example: &lt;pre&gt;(non-ASF) <a href="http://github.com/isisaddons/isis-wicket-gmap3%5BIsis">http://github.com/isisaddons/isis-wicket-gmap3[Isis</a> addons' gmap3] wicket extension&lt;/pre&gt;</td>
<td>(non-ASF) Isis addons' gmap3 wicket extension</td>
</tr>
</tbody>
</table>

### 8.5. Source code

<table>
<thead>
<tr>
<th>Abbrev.</th>
<th>Produces</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>adsrcjava</strong></td>
<td><strong>[source,java]</strong>&lt;br&gt;----&lt;br&gt;xxx&lt;br&gt;---- where: * xxx is the source code snippet.</td>
<td><strong>[source,java]</strong> --- public class Foo { ... } ---</td>
</tr>
<tr>
<td><strong>adsrcjavac</strong></td>
<td>as for <strong>adsrcjava</strong>, but with a caption above</td>
<td></td>
</tr>
<tr>
<td><strong>adsrcjavascr ipt</strong></td>
<td><strong>[source,javascript]</strong>&lt;br&gt;----&lt;br&gt;xxx&lt;br&gt;---- where: * xxx is the source code snippet.</td>
<td><strong>[source,javascript]</strong> --- $(document).ready(function() { ... }); ---</td>
</tr>
<tr>
<td><strong>adsrcjavascr iptc</strong></td>
<td>as for <strong>adsrcjavascript</strong>, but with a caption above</td>
<td></td>
</tr>
<tr>
<td><strong>adsrcother</strong></td>
<td><strong>[source,nnn]</strong>&lt;br&gt;----&lt;br&gt;xxx&lt;br&gt;---- where: * nnn is the programming language * xxx is the source code snippet.</td>
<td></td>
</tr>
<tr>
<td><strong>adsrcotherc</strong></td>
<td>as for <strong>adsrcother</strong>, but with a caption above</td>
<td></td>
</tr>
<tr>
<td><strong>adsrcxml</strong></td>
<td><strong>[source,javascript]</strong>&lt;br&gt;----&lt;br&gt;xxx&lt;br&gt;---- where: * xxx is the source code snippet.</td>
<td><strong>[source,xml]</strong> --- &lt;html&gt;&lt;title&gt; hello world! &lt;/title&gt; &lt;/html&gt; ---</td>
</tr>
<tr>
<td><strong>adsrcxmlc</strong></td>
<td>as for <strong>adsrcxml</strong>, but with a caption above</td>
<td></td>
</tr>
</tbody>
</table>

### 8.6. Images
### 8.7. YouTube (screencasts)

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

<table>
<thead>
<tr>
<th>Abbrev.</th>
<th>Produces</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>adyoutube</td>
<td><code>&lt;pre&gt;video::xxx[youtube,width=&quot;WWWpx&quot;,height=&quot;HHHpx&quot;]&lt;/pre&gt;</code></td>
<td><code>video::bj8735nBRR4[youtube,width=&quot;210px&quot;,height=&quot;118px&quot;]</code></td>
</tr>
<tr>
<td>adyoutubec</td>
<td>as for <code>youtube</code>, but with a caption above</td>
<td></td>
</tr>
</tbody>
</table>

### 8.8. Tables

<table>
<thead>
<tr>
<th>Abbrev.</th>
<th>Produces</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>adtbl3</td>
<td>Table with 3 columns, 3 rows.</td>
<td></td>
</tr>
</tbody>
</table>

### 8.9. Misc.

<table>
<thead>
<tr>
<th>Abbrev.</th>
<th>Produces</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>adai</td>
<td>`&lt;pre&gt;Apache Isis&lt;/pre&gt;&lt;br&gt; That is, the literal text &quot;Apache Isis&quot;.</td>
<td>Apache Isis</td>
</tr>
<tr>
<td>Abbrev.</td>
<td>Produces</td>
<td>Example</td>
</tr>
<tr>
<td>---------</td>
<td>----------</td>
<td>---------</td>
</tr>
<tr>
<td>adlink</td>
<td><code>&lt;pre&gt;link:xxx[ttt]&lt;/pre&gt;</code>, where: * <code>&lt;code&gt;xxx&lt;/code&gt;</code> is * <code>&lt;code&gt;ttt&lt;/code&gt;</code> is the text to display as the hyperlink for example: <code>&lt;pre&gt;link:http://isis.apache.org[Apache Isis website]&lt;/pre&gt;</code></td>
<td>Apache Isis website</td>
</tr>
<tr>
<td>adanchany</td>
<td><code>&lt;pre&gt;= anchor:[xxx]&lt;/pre&gt;</code> defines an inline anchor to any heading, where: * <code>&lt;code&gt;xxx&lt;/code&gt;</code> is the anchor text. For example: <code>&lt;pre&gt;= anchor:[_ugfun_i18n] Internationalization&lt;/pre&gt;</code> An alternative (more commonly used in our documentation) is to use the <code>&lt;code&gt;[[&amp;#8230;&amp;#8203;]]&lt;/code&gt;</code> directly above the heading: <code>&lt;pre&gt;[_ugfun_i18n] = Internationalization&lt;/pre&gt;</code></td>
<td></td>
</tr>
<tr>
<td>adxrefany</td>
<td><code>&lt;pre&gt;xref:[xxx]&lt;/pre&gt;</code> cross-reference to any document/anchor, where: * <code>&lt;code&gt;xxx&lt;/code&gt;</code> is the fully qualified document with optional anchor</td>
<td></td>
</tr>
<tr>
<td>adfootnote</td>
<td><code>&lt;pre&gt;.footnote:[]&lt;/pre&gt;</code> defines a footnote</td>
<td>. [1: this is a footnote]</td>
</tr>
</tbody>
</table>
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:

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

you can instead write simply:

```java
@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 has however provided a complimentary copy of its software to Apache Isis committers.

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:

```java
// 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).