Naked Objects' Sister Projects 1.0: Developers Guide

aka Star Objects

Version 0.1

Preface

Star Objects is an umbrella project for the various sister projects for the Naked Objects framework. Its purpose is to :

- describe the development environment common to all sister projects:
  - how to set up the development environment;
  - how to write site and docbook documentation
- define standards common to all sister projects:

  a Maven corporate POM that defines common third party dependencies, configuration of build plugins and so on;
- provide a Maven site skin to provide a common look-n-feel of the Maven sites created for the sister projects.

This developers guide breaks into several parts:

- the first part is for those just wanting to build the sister projects from source, without necessarily contributing any changes back to the projects. As such, it is quite short;
- the second part is for those who are intending to contribute changes back. It deals with such matters as setting up the development environment, explaining source code structure, how to use the corporate POM, writing documentation and site documentation;
- the third part is for use only by those maintaining *this* (umbrella) project. It deals with releasing the corporate POM and Maven site skin.

Both Naked Objects and all its sister projects are currently hosted on Sourceforge, under Apache Software License v2.

Part I

Building from Source

This part of the developers guide describes how to build any of the sister projects from source, for example, just to get a better understanding of how the project works, or because your organization requires any open source projects to be re-built in-house.

This part of the guide also describes the prerequisite software that needs to be installed prior to building from source.

If you are looking to contribute to any of the sister projects, the steps in this part of the guide also apply, but see also Part II, “Contributing Changes” for additional steps.

Chapter 1

Prerequisite Software

This chapter describes the (Java) software needed to build the sister projects. Follow the steps in this chapter if you want to build the sister projects from source, or want to contribute back to the sister projects.

For the most part the sister projects are implemented using Java. Some sister projects may have their own additional prerequisites; consult their documentation.

1.1. Command Line Software

Install the following software (you may well already have these installed):

- a Subversion client;
- Java 5 or 6 and setup the `JAVA_HOME` environment variable;
- Maven 2.1.0 or later, setup the `MAVEN_HOME` environment variable.

Add Subversion's `svn` executable and Maven's `mvn` executable to your `PATH` environment variable.

1.2. IDE

Although it is possible to develop from the command line tools, if you are looking to develop on or contribute back to any of the sister projects then you should set up a development environment.

Any IDE that supports Maven will do; we use Eclipse along with a number of plugins:

- Install Eclipse 3.5 (Java or JEE edition).
- the m2eclipse plugin, for Maven support

  Normally Eclipse uses its own `.project` and `.classpath` files to (respectively) define the layout of the project and the classpath for a project. With Maven however this same information is available in the `pom.xml`. What *m2eclipse* does is generate the `.project` and `.classpath` files on-the-fly, and uses

Eclipse's own "Classpath Container" to reference Maven modules in the local repository. m2eclipse will also download any referenced modules from remote repositories into the local repository.

- the Subclipse plugin, for Subversion support
- the eclipse-cs plugin, for Checkstyle support
- the PMD for eclipse plugin, for PMD support
- the EclEmma plugin, for Emma code coverage support

Install each of these from their respective update sites.

Chapter 2

Manual Install of Maven Artifacts

This chapter describes how to install some artifacts into your local Maven repository. Follow the steps in this chapter if you want to build the sister projects from source, or want to contribute back to the sister projects.

Generally speaking all Maven artifacts are automatically downloads from the Maven central repo (though note that all the sister projects also release to the Sister projects Maven repo). However, some artifacts must be manually installed or built, either because they haven't been formally released or because there are licensing restrictions preventing them from being hosted in the repository. These must therefore be downloaded and manually installed into your local repository (`~/.m2/repository`) .

2.1. JIMI Jar File

The sister projects currently use the JIMI jar to generate documentation. Therefore:

- Download the *jimi* jar file. You'll find it within the `JimiProClasses.zip` downloadable from the Jimi project page.
- Install into your local Maven repository using:

```
mvn install:install-file       \
    -D groupId=com.java        \
    -D artifactId=jimi         \
    -D version=1.0             \
    -D packaging=jar           \
    -D file=/path/to/jimi.jar
```

Chapter 3

Building Projects from Source

This chapter describes the general organization of the source code in any sister project, and provides general guidance on how to build sister projects from source. Follow the steps in this chapter if you want to build the sister projects from source, or want to contribute back to the sister projects.

So far as possible, the codebase for each sister project follows the same basic directory structure, though the modules in each and their implementation will vary of course. The instructions given here are general a guide, but you should also consult the developers' guides for each sister project for any additional details, and any artifacts that fall outside the standard directory structure. (For example, the *Star Objects* project itself supplies the corporate POM and a site template, the steps for which documented in Part IV, “Building and Deploying the Corporate Artifacts” of part three of this guide.)

3.1. Source Tree Hierarchy

Although the code in every sister project is different, they all (try to) follow the same general hierarchy:

```
trunk/
  main/               # main module, aggregates submodules
    pom.xml
    applib/           # application library (if any)
    documentation/    # users' and developers' guides
    xxx/              #
    yyy/              # modules specific to the sister project
    zzz/              #

  support/            # optional, built after main
    pom.xml
    release/          # a consistent stack of dependencies for this release
    archetype/        # quick-start archetype (if any)

  testapp/            # optional, not released
    pom.xml
```

The 'main' module

The main module is a parent that aggregates the various submodules that make up the sister project:

- the applib (if any) contains the application library. For example, Restful Objects defines an applib for client-side applications calling the RESTful web service;
- the documentation, holding users' and developers' guides;
- any further submodules that make up the application. For example, *Restful Objects* has a viewer module; *JPA Objects* has a module for an object store implementation.

The steps for building the main module for most sister projects will amount to little more than:

```
cd trunk/main
mvn clean install
```

If you want to build a single module, then use the '-pl' flag. For example:

```
cd trunk/main
mvn clean install -pl documentation
```

just builds the documentation.

Alternatively, in general it's also possible to cd into the appropriate directory and build from there:

```
cd trunk/main/documentation
mvn clean install
```

The 'support' module

The support module, if it exists, is another parent that is built after the main module. It aggregates submodules that to help developers use the sister project:

- a release artifact, which defines (in `<dependencyManagement>`) a consistent set of artifacts. This can be used as a parent (or at least documents the set of dependencies that have been tested together);
- an archetype artifact, which can be used for quick-starts. The archetype should be designed to work with the Naked Objects "claims" example. In general the modules created by running the archetype should use the release artifact.

The steps for building the support module for most sister projects will again amount to little more than:

```
cd trunk/main
mvn clean install
```

The reason that the support module is built after the main module is because it requires the main's submodules to be have released.

The 'testapp' module

The testapp module, if it exists, is used for testing. It should be built using the archetype, if there is one, and/or should use the release artifact as its parent.

Part II

Contributing Changes

If you are looking to contribute to any of the sister projects, then this part of the guide offers guidance on how to contribute changes. It builds on the first part, Part I, “Building from Source”, and assumes that you have installed any prerequisite software

Chapter 4
Coding Standards

This chapter describes how we enforce coding standards for sister projects. Follow the steps in this chapter if you intend to contribute back to the sister projects. There's no need to perform these steps if you are just building the sister projects from source.

All the sister projects use Checkstyle and PMD to enforce coding standards, and use Cobertura and Emma to capture code coverage. Each of these tools has integration with both Maven and Eclipse. In addition, a number of Eclipse config files define formatting and clean-up standards that are complementary to the Checkstyle standards.

All of the config files described here are checked into Subversion under trunk/main/standards/src/main/resources. To ease distribution they are released as part of the Star Objects sourceforge website. The config files are not versioned (other than by Subversion itself).

4.1. Checkstyle

Checkstyle is a powerful tool for enforcing coding standards and detecting certain classes of likely errors. The checkstyle definition project is derived from Sun's standards, with a number of modifications, and can be accessed from the Star Objects sourceforge website.

Checkstyle integrates with both Maven and with Eclipse.

Maven Plugin

The corporate POM (see Part IV, “Building and Deploying the Corporate Artifacts”) automatically includes the Maven checkstyle plugin:

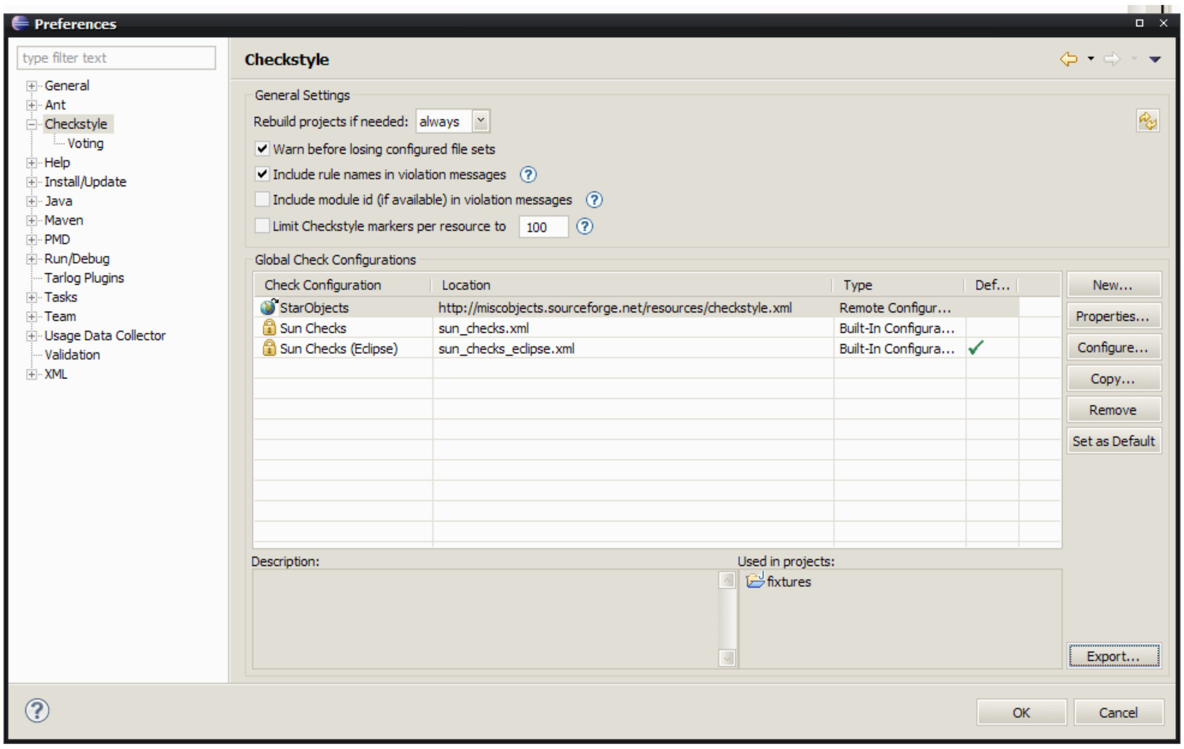
```
<!-- quality checks: checkstyle -->
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-checkstyle-plugin</artifactId>
    <configuration>
        <configLocation>
          http://starobjects.sourceforge.net/m2-site/main/standards/resources/checkstyle.xml
        </configLocation>
        ...
    </configuration>
</plugin>
```

This plugin is not bound to any Maven lifecycle phase, and is not intended to be run other than as a report within `mvn site`. In particular, note that the configuration is only defined in the `<reporting>` section, so it isn't possible to run using `mvn checkstyle:checkstyle`. For more immediate feedback, use the Eclipse plugin, below.

Eclipse Plugin

The eclipse-cs plugin allows Checkstyle violations to be flagged as warnings or errors within the Problems view of the Eclipse IDE. To associate *eclipse-cs* with the Checkstyle config file, use Windows > Preferences, and specify the config file as http://starobjects.sourceforge.net/m2-site/main/standards/resources/checkstyle.xml:

TODO: the screenshot is out of date



Checkstyle

General Settings

Rebuild projects if needed: always

✓ Warn before losing configured file sets

✓ Include rule names in violation messages

☐ Include module id (if available) in violation messages

☐ Limit Checkstyle markers per resource to 100

Global Check Configurations

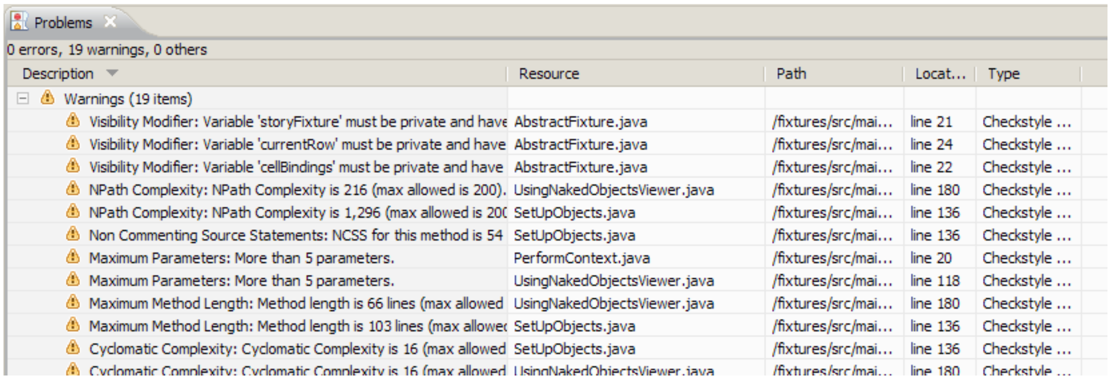
Check Configuration	Location	Type	Def...
StarObjects	http://miscobjects.sourceforge.net/resources/checkstyle.xml	Remote Configur...	
Sun Checks	sun_checks.xml	Built-In Configura...	
Sun Checks (Eclipse)	sun_checks_eclipse.xml	Built-In Configura...	✓

Description:

Used in projects: fixtures

According to eclipse-cs' documentation, it is meant to integrate with m2eclipse and transparently pick up any Maven configuration of mvn-checkstyle-plugin. This doesn't seem to work for me, though.

You may then need to enable CheckStyle for each project as required, using the context menu in Package Explorer. CheckStyle violations show up in the Problems view:



Problems

0 errors, 19 warnings, 0 others

Description	Resource	Path	Locat...	Type
Warnings (19 items)				
Visibility Modifier: Variable 'storyFixture' must be private and have	AbstractFixture.java	/fixtures/src/mai...	line 21	Checkstyle ...
Visibility Modifier: Variable 'currentRow' must be private and have	AbstractFixture.java	/fixtures/src/mai...	line 24	Checkstyle ...
Visibility Modifier: Variable 'cellBindings' must be private and have	AbstractFixture.java	/fixtures/src/mai...	line 22	Checkstyle ...
NPath Complexity: NPath Complexity is 216 (max allowed is 200).	UsingNakedObjectsViewer.java	/fixtures/src/mai...	line 180	Checkstyle ...
NPath Complexity: NPath Complexity is 1,296 (max allowed is 20(	SetUpObjects.java	/fixtures/src/mai...	line 136	Checkstyle ...
Non Commenting Source Statements: NCSS for this method is 54	SetUpObjects.java	/fixtures/src/mai...	line 136	Checkstyle ...
Maximum Parameters: More than 5 parameters.	PerformContext.java	/fixtures/src/mai...	line 20	Checkstyle ...
Maximum Parameters: More than 5 parameters.	UsingNakedObjectsViewer.java	/fixtures/src/mai...	line 118	Checkstyle ...
Maximum Method Length: Method length is 66 lines (max allowed	UsingNakedObjectsViewer.java	/fixtures/src/mai...	line 180	Checkstyle ...
Maximum Method Length: Method length is 103 lines (max allowe(	SetUpObjects.java	/fixtures/src/mai...	line 136	Checkstyle ...
Cyclomatic Complexity: Cyclomatic Complexity is 16 (max allowed	SetUpObjects.java	/fixtures/src/mai...	line 136	Checkstyle ...
Cyclomatic Complexity: Cyclomatic Complexity is 16 (max allowed	UsingNakedObjectsViewer.java	/fixtures/src/mai...	line 180	Checkstyle ...

These violations are dynamic updated, so fixing any problem should automatically remove the violation from the problems view. A quick fix short cut (ctrl+1) is available for some of these.

4.2. PMD

PMD is another static analysis tool that checks for problematic code (possible bugs, dead code, suboptimal code and so on). As for Checkstyle, all the sister projects share a common PMD definition file, this one adapted from Apache group's own PMD definitions.

PMD integrates with both Maven and with Eclipse.

Maven Plugin

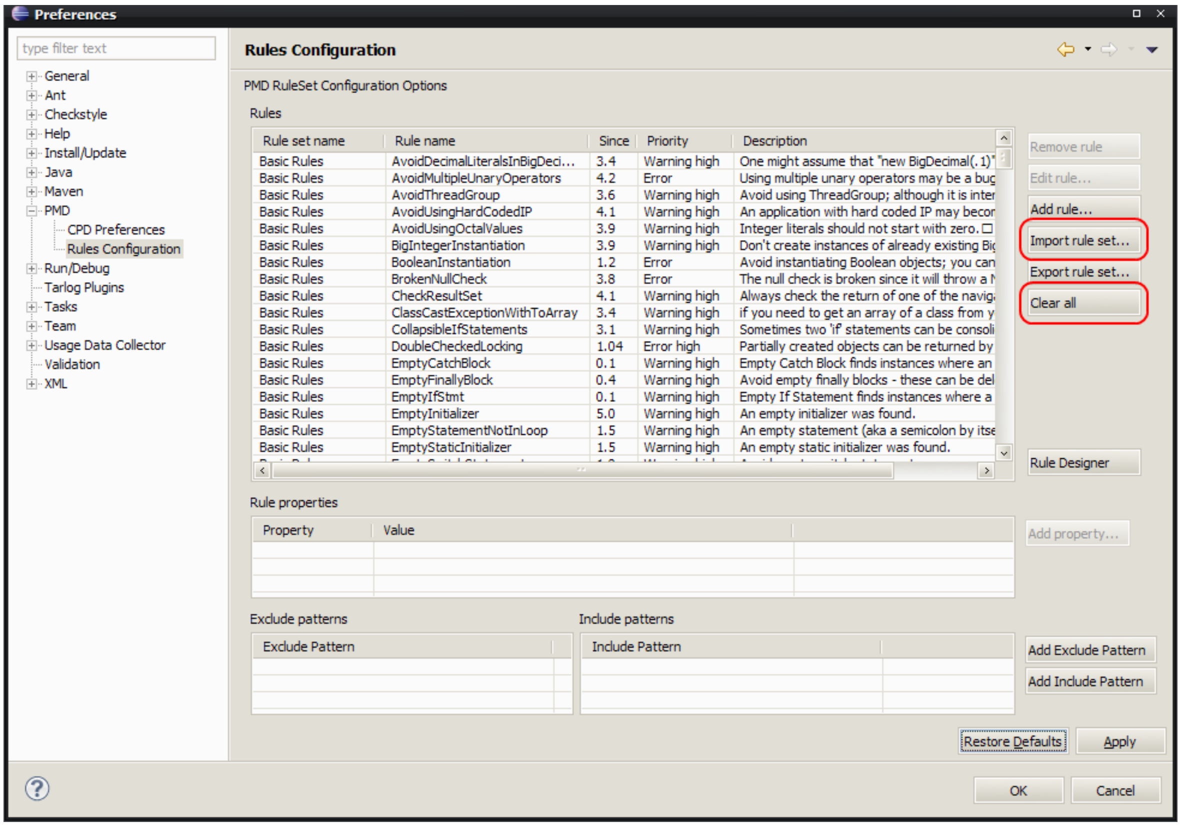
The corporate POM (see Part IV, “Building and Deploying the Corporate Artifacts”) automatically includes the Maven PMD plugin:

```
<!-- quality checks: pmd -->
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-pmd-plugin</artifactId>
    <configuration>
        <targetJdk>${compileTarget}</targetJdk>
        <rulesets>
            <ruleset>
              http://starobjects.sourceforge.net/m2-site/main/standards/resources/pmd.xml
            </ruleset>
        </rulesets>
        ...
    </configuration>
</plugin>
```

As for Checkstyle, this plugin is not bound to any Maven lifecycle phase, and is not intended to be run other than as a report within `mvn site`. In particular, note that the configuration is only defined in the `<reporting>` section, so it isn't possible to run using `mvn pmd:pmd`. For more immediate feedback, use the Eclipse plugin, below.

Eclipse Plugin

The PMD for eclipse plugin allows PMD violations to be flagged as warnings or errors within the Problems view of the Eclipse IDE. The plugin also provides a custom "PMD" perspective which also lists all violations. To associate *PMD for Eclipse* with the PMD config file, use Windows > Preferences:



Rules Configuration

PMD RuleSet Configuration Options

Rules

Rule set name	Rule name	Since	Priority	Description
Basic Rules	AvoidDecimalLiteralsInBigDeci...	3.4	Warning high	One might assume that "new BigDecimal(.1)"
Basic Rules	AvoidMultipleUnaryOperators	4.2	Error	Using multiple unary operators may be a buc
Basic Rules	AvoidThreadGroup	3.6	Warning high	Avoid using ThreadGroup; although it is inter
Basic Rules	AvoidUsingHardCodedIP	4.1	Warning high	An application with hard coded IP may becor
Basic Rules	AvoidUsingOctalValues	3.9	Warning high	Integer literals should not start with zero. ☐
Basic Rules	BigIntegerInstantiation	3.9	Warning high	Don't create instances of already existing Bi
Basic Rules	BooleanInstantiation	1.2	Error	Avoid instantiating Boolean objects; you can
Basic Rules	BrokenNullCheck	3.8	Error	The null check is broken since it will throw a N
Basic Rules	CheckResultSet	4.1	Warning high	Always check the return of one of the navig
Basic Rules	ClassCastExceptionWithToArray	3.4	Warning high	if you need to get an array of a class from y
Basic Rules	CollapsibleIfStatements	3.1	Warning high	Sometimes two 'if' statements can be consoli
Basic Rules	DoubleCheckedLocking	1.04	Error high	Partially created objects can be returned by
Basic Rules	EmptyCatchBlock	0.1	Warning high	Empty Catch Block finds instances where an
Basic Rules	EmptyFinallyBlock	0.4	Warning high	Avoid empty finally blocks - these can be del
Basic Rules	EmptyIfStmt	0.1	Warning high	Empty If Statement finds instances where a
Basic Rules	EmptyInitializer	5.0	Warning high	An empty initializer was found.
Basic Rules	EmptyStatementNotInLoop	1.5	Warning high	An empty statement (aka a semicolon by itse
Basic Rules	EmptyStaticInitializer	1.5	Warning high	An empty static initializer was found.

Rule properties

Property	Value

Exclude patterns

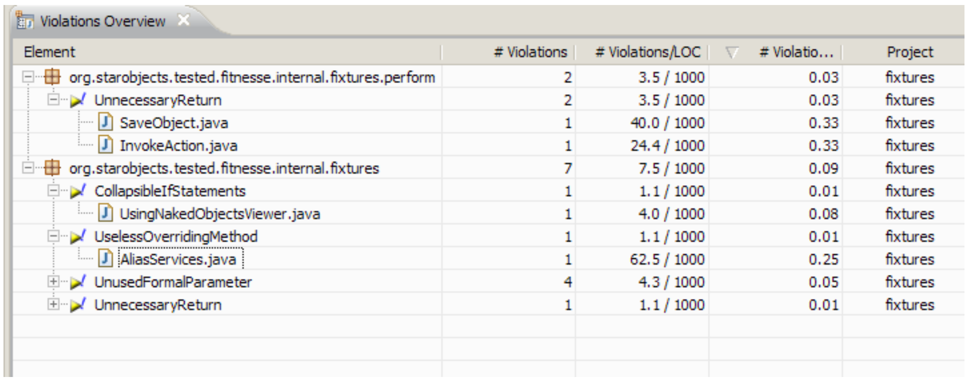
Exclude Pattern

Include patterns

Include Pattern

First, use "Clear All" to remove the default rule set. Then, use "Import rule set" and specify the pmd.xml file. Unlike the Checkstyle plugin, this must be a local file so downloaded from the Star Objects website; the file to use is http://starobjects.sourceforge.net/m2-site/main/standards/resources/pmd.xml.

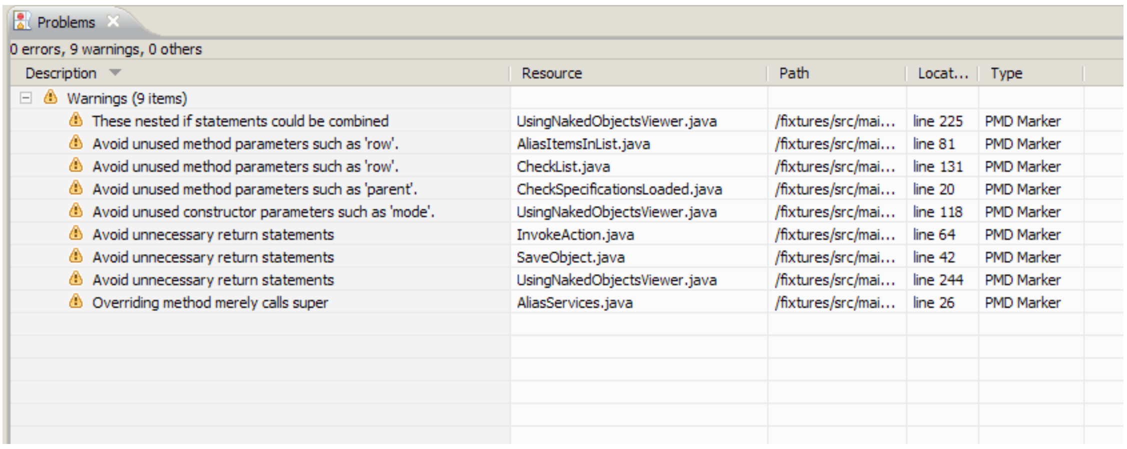
Also unlike the Checkstyle plugin, PMD violations are not continually updated against the code (it is not implemented as an Eclipse builder). To perform a check, you must use the context menu in Package Explorer and then use PMD > Check Code with PMD. This will switch into the PMD perspective; the Violations view will indicate any code that needs attention:



Violations Overview

Element	# Violations	# Violations/LOC	# Violatio...	Project
org.starobjects.tested.fitnesse.internal.fixtures.perform	2	3.5 / 1000	0.03	fixtures
UnnecessaryReturn	2	3.5 / 1000	0.03	fixtures
SaveObject.java	1	40.0 / 1000	0.33	fixtures
InvokeAction.java	1	24.4 / 1000	0.33	fixtures
org.starobjects.tested.fitnesse.internal.fixtures	7	7.5 / 1000	0.09	fixtures
CollapsibleIfStatements	1	1.1 / 1000	0.01	fixtures
UsingNakedObjectsViewer.java	1	4.0 / 1000	0.08	fixtures
UselessOverridingMethod	1	1.1 / 1000	0.01	fixtures
AliasServices.java	1	62.5 / 1000	0.25	fixtures
UnusedFormalParameter	4	4.3 / 1000	0.05	fixtures
UnnecessaryReturn	1	1.1 / 1000	0.01	fixtures

Alternatively, you can switch back to the Java view; any PMD violations show up in the Problems view:



Problems

0 errors, 9 warnings, 0 others

Description	Resource	Path	Locat...	Type
Warnings (9 items)				
These nested if statements could be combined	UsingNakedObjectsViewer.java	/fixtures/src/mai...	line 225	PMD Marker
Avoid unused method parameters such as 'row'.	AliasItemsInList.java	/fixtures/src/mai...	line 81	PMD Marker
Avoid unused method parameters such as 'row'.	CheckList.java	/fixtures/src/mai...	line 131	PMD Marker
Avoid unused method parameters such as 'parent'.	CheckSpecificationsLoaded.java	/fixtures/src/mai...	line 20	PMD Marker
Avoid unused constructor parameters such as 'mode'.	UsingNakedObjectsViewer.java	/fixtures/src/mai...	line 118	PMD Marker
Avoid unnecessary return statements	InvokeAction.java	/fixtures/src/mai...	line 64	PMD Marker
Avoid unnecessary return statements	SaveObject.java	/fixtures/src/mai...	line 42	PMD Marker
Avoid unnecessary return statements	UsingNakedObjectsViewer.java	/fixtures/src/mai...	line 244	PMD Marker
Overriding method merely calls super	AliasServices.java	/fixtures/src/mai...	line 26	PMD Marker

Once violations are fixed, the Check must be performed again to refresh both of these views.

Note, to prevent PMD from switching to its own perspective, use Windows > Preferences > PMD and then disable "Show PMD Perspective when checking code".

4.3. Code Coverage (Cobertura and Emma)

Code coverage of unit testing is provided using Cobertura for Maven, and using Emma for Eclipse. Although there is an Eclipse plugin for Cobertura, it has not been maintained and no longer runs on the latest versions of Eclipse. Conversely, although there is a Maven plugin for Emma, the Cobertura plugin gives reports that are more easily understood.

Maven Plugin (Cobertura)

The corporate POM (see Part IV, “Building and Deploying the Corporate Artifacts”) automatically includes the Maven Cobertura plugin:

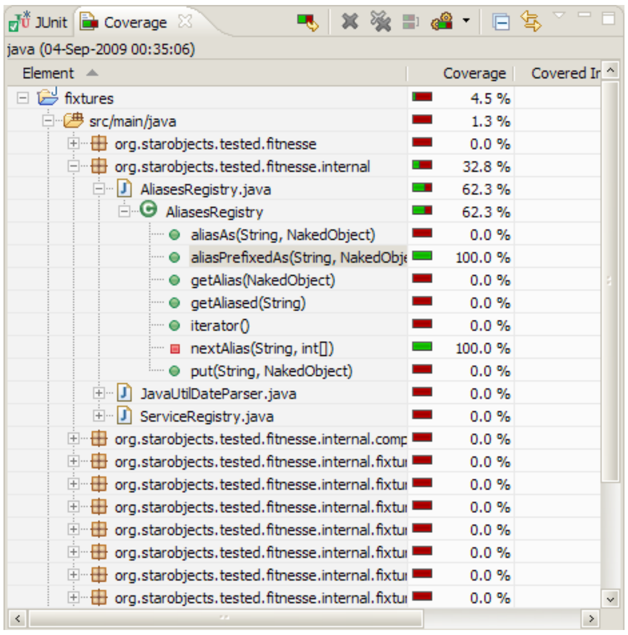
```
<!-- quality checks: pmd -->
<plugin>
    <groupId>org.codehaus.mojo</groupId>
    <artifactId>cobertura-maven-plugin</artifactId>
    <version>2.3</version>
    <inherited>true</inherited>
</plugin>
```

As for Checkstyle and PMD, this plugin is not bound to any Maven lifecycle phase, and is not intended to be run other than as a report within `mvn site`. For more immediate feedback, use the Emma Eclipse plugin, below.

Eclipse Plugin (Emma)

The EclEmma Eclipse plugin offers transparent code coverage directly within the Eclipse IDE. Rather than run the tests using Run > Run As > JUnit Test, instead use Coverage > Coverage As > JUnit Test. This will instrument the code and then provide a coverage view and highlights in the editor to show which code has been exercised.

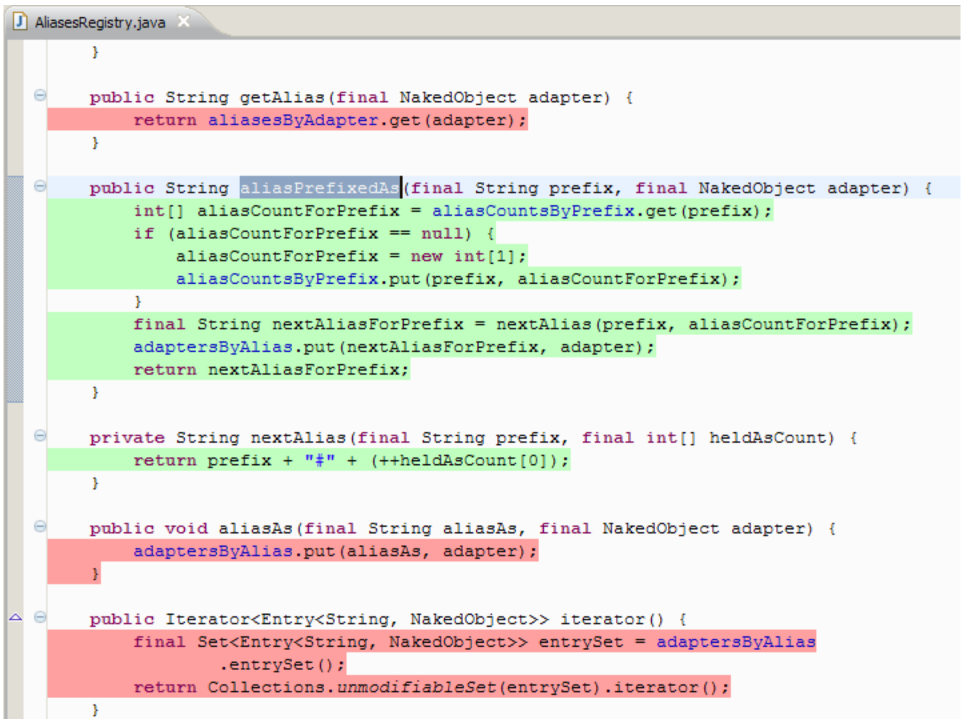
The coverage view looks like:



java (04-Sep-2009 00:35:06)

Element	Coverage	Covered Ir
fixtures	4.5 %	
src/main/java	1.3 %	
org.starobjects.tested.fitnesse	0.0 %	
org.starobjects.tested.fitnesse.internal	32.8 %	
AliasesRegistry.java	62.3 %	
AliasesRegistry	62.3 %	
aliasAs(String, NakedObject)	0.0 %	
aliasPrefixedAs(String, NakedObje	100.0 %	
getAlias(NakedObject)	0.0 %	
getAliased(String)	0.0 %	
iterator()	0.0 %	
nextAlias(String, int[])	100.0 %	
put(String, NakedObject)	0.0 %	
JavaUtilDateParser.java	0.0 %	
ServiceRegistry.java	0.0 %	
org.starobjects.tested.fitnesse.internal.comp	0.0 %	
org.starobjects.tested.fitnesse.internal.fixtu	0.0 %	
org.starobjects.tested.fitnesse.internal.fixtu	0.0 %	
org.starobjects.tested.fitnesse.internal.fixtu	0.0 %	
org.starobjects.tested.fitnesse.internal.fixtu	0.0 %	
org.starobjects.tested.fitnesse.internal.fixtu	0.0 %	
org.starobjects.tested.fitnesse.internal.fixtu	0.0 %	
org.starobjects.tested.fitnesse.internal.fixtu	0.0 %	

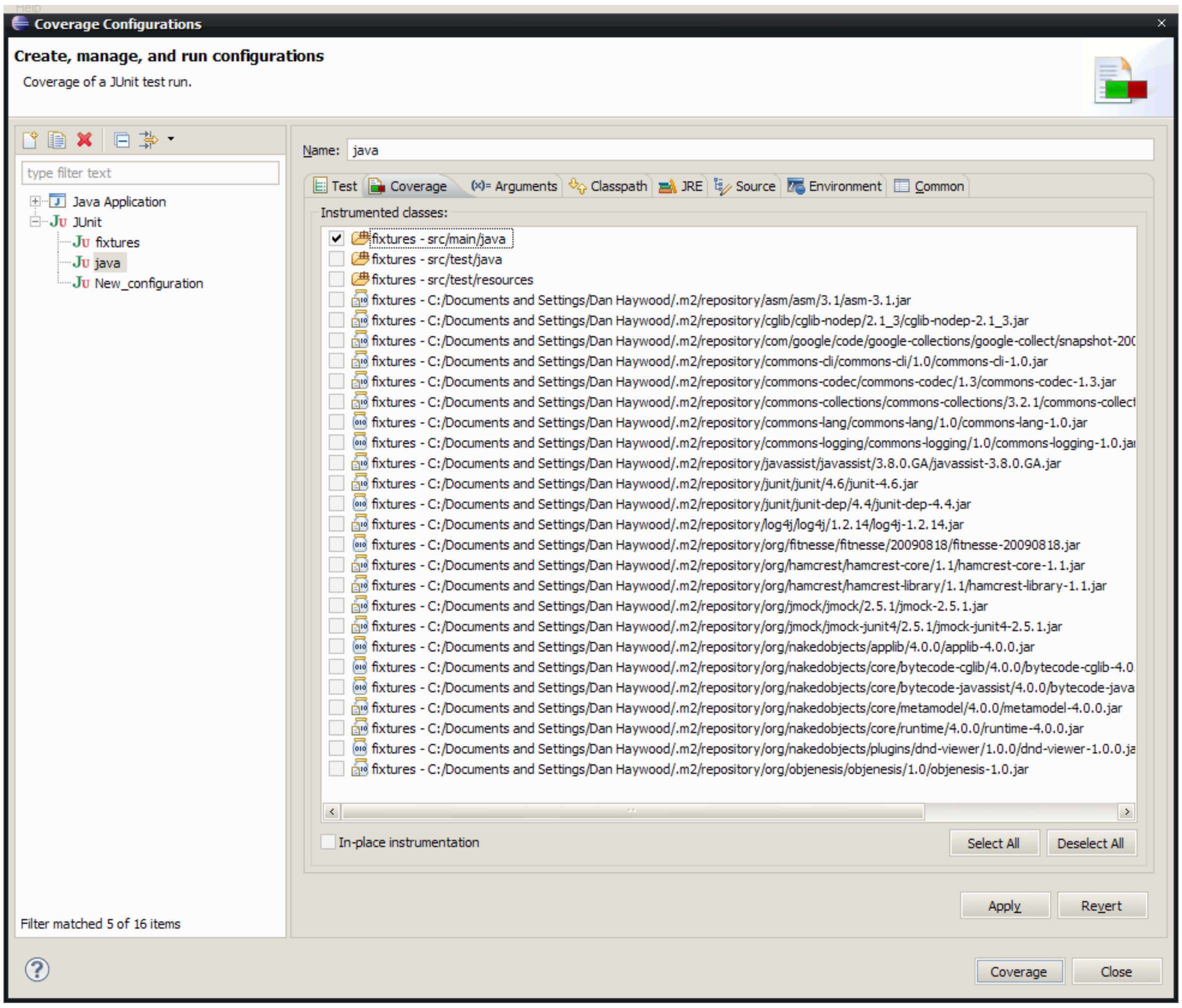
The highlighted editor looks like:


AliasesRegistry.java

```
    }

    public String getAlias(final NakedObject adapter) {
        return aliasesByAdapter.get(adapter);
    }

    public String aliasPrefixedAs(final String prefix, final NakedObject adapter) {
        int[] aliasCountForPrefix = aliasCountsByPrefix.get(prefix);
        if (aliasCountForPrefix == null) {
            aliasCountForPrefix = new int[1];
            aliasCountsByPrefix.put(prefix, aliasCountForPrefix);
        }
        final String nextAliasForPrefix = nextAlias(prefix, aliasCountForPrefix);
        adaptersByAlias.put(nextAliasForPrefix, adapter);
        return nextAliasForPrefix;
    }

    private String nextAlias(final String prefix, final int[] heldAsCount) {
        return prefix + "#" + (++heldAsCount[0]);
    }

    public void aliasAs(final String aliasAs, final NakedObject adapter) {
        adaptersByAlias.put(aliasAs, adapter);
    }

    public Iterator<Entry<String, NakedObject>> iterator() {
        final Set<Entry<String, NakedObject>> entrySet = adaptersByAlias
                .entrySet();
        return Collections.unmodifiableSet(entrySet).iterator();
    }
```

Use the menu item on the Coverage View to select between manage the history of coverage runs (selecting none / deleting all removes the highlights on the editor).

To change what code is instrumented, use Coverage > Coverage ... :



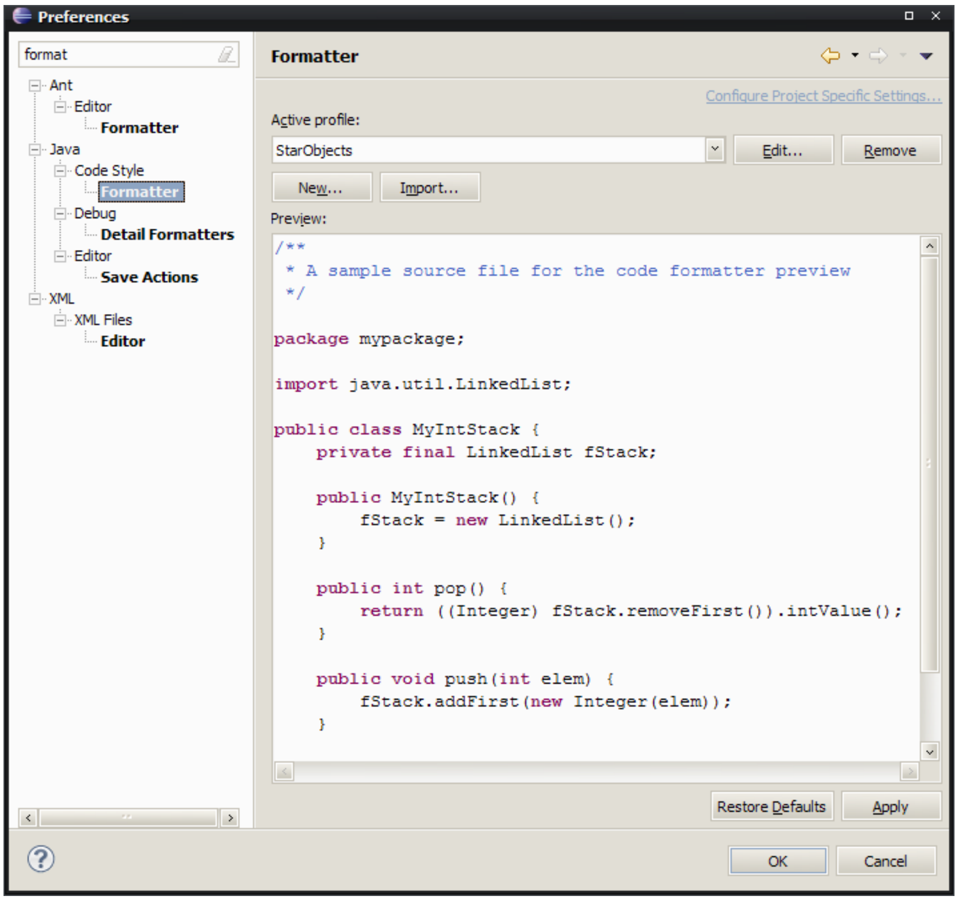
```
[✓] fixtures - src/main/java
[ ] fixtures - src/test/java
[ ] fixtures - src/test/resources
[ ] fixtures - C:/Documents and Settings/Dan Haywood/.m2/repository/asm/asm/3.1/asm-3.1.jar
[ ] fixtures - C:/Documents and Settings/Dan Haywood/.m2/repository/cglib/cglib-nodep/2.1_3/cglib-nodep-2.1_3.jar
[ ] fixtures - C:/Documents and Settings/Dan Haywood/.m2/repository/com/google/code/google-collections/google-collect/snapshot-200
[ ] fixtures - C:/Documents and Settings/Dan Haywood/.m2/repository/commons-cli/commons-cli/1.0/commons-cli-1.0.jar
[ ] fixtures - C:/Documents and Settings/Dan Haywood/.m2/repository/commons-codec/commons-codec/1.3/commons-codec-1.3.jar
[ ] fixtures - C:/Documents and Settings/Dan Haywood/.m2/repository/commons-collections/commons-collections/3.2.1/commons-collect
[ ] fixtures - C:/Documents and Settings/Dan Haywood/.m2/repository/commons-lang/commons-lang/1.0/commons-lang-1.0.jar
[ ] fixtures - C:/Documents and Settings/Dan Haywood/.m2/repository/commons-logging/commons-logging/1.0/commons-logging-1.0.ja
[ ] fixtures - C:/Documents and Settings/Dan Haywood/.m2/repository/javassist/javassist/3.8.0.GA/javassist-3.8.0.GA.jar
[ ] fixtures - C:/Documents and Settings/Dan Haywood/.m2/repository/junit/junit/4.6/junit-4.6.jar
[ ] fixtures - C:/Documents and Settings/Dan Haywood/.m2/repository/junit/junit-dep/4.4/junit-dep-4.4.jar
[ ] fixtures - C:/Documents and Settings/Dan Haywood/.m2/repository/log4j/log4j/1.2.14/log4j-1.2.14.jar
[ ] fixtures - C:/Documents and Settings/Dan Haywood/.m2/repository/org/fitnesse/fitnesse/20090818/fitnesse-20090818.jar
[ ] fixtures - C:/Documents and Settings/Dan Haywood/.m2/repository/org/hamcrest/hamcrest-core/1.1/hamcrest-core-1.1.jar
[ ] fixtures - C:/Documents and Settings/Dan Haywood/.m2/repository/org/hamcrest/hamcrest-library/1.1/hamcrest-library-1.1.jar
[ ] fixtures - C:/Documents and Settings/Dan Haywood/.m2/repository/org/jmock/jmock/2.5.1/jmock-2.5.1.jar
[ ] fixtures - C:/Documents and Settings/Dan Haywood/.m2/repository/org/jmock/jmock-junit4/2.5.1/jmock-junit4-2.5.1.jar
[ ] fixtures - C:/Documents and Settings/Dan Haywood/.m2/repository/org/nakedobjects/applib/4.0.0/applib-4.0.0.jar
[ ] fixtures - C:/Documents and Settings/Dan Haywood/.m2/repository/org/nakedobjects/core/bytecode-cglib/4.0.0/bytecode-cglib-4.0
[ ] fixtures - C:/Documents and Settings/Dan Haywood/.m2/repository/org/nakedobjects/core/bytecode-javassist/4.0.0/bytecode-java
[ ] fixtures - C:/Documents and Settings/Dan Haywood/.m2/repository/org/nakedobjects/core/metamodel/4.0.0/metamodel-4.0.0.jar
[ ] fixtures - C:/Documents and Settings/Dan Haywood/.m2/repository/org/nakedobjects/core/runtime/4.0.0/runtime-4.0.0.jar
[ ] fixtures - C:/Documents and Settings/Dan Haywood/.m2/repository/org/nakedobjects/plugins/dnd-viewer/1.0.0/dnd-viewer-1.0.0.ja
[ ] fixtures - C:/Documents and Settings/Dan Haywood/.m2/repository/org/objenesis/objenesis/1.0/objenesis-1.0.jar
```

4.4. Eclipse Code Style

Eclipse provides a number of interrelated features to help write code consistent with defined standards.

Formatter

The Eclipse formatter config file (accessible here) defines a set of formatting standards consistent with the Checkstyle checks (see Section 4.1, “Checkstyle”). The config file should be imported using Windows > Preferences > Java > Code Style > Formatter:

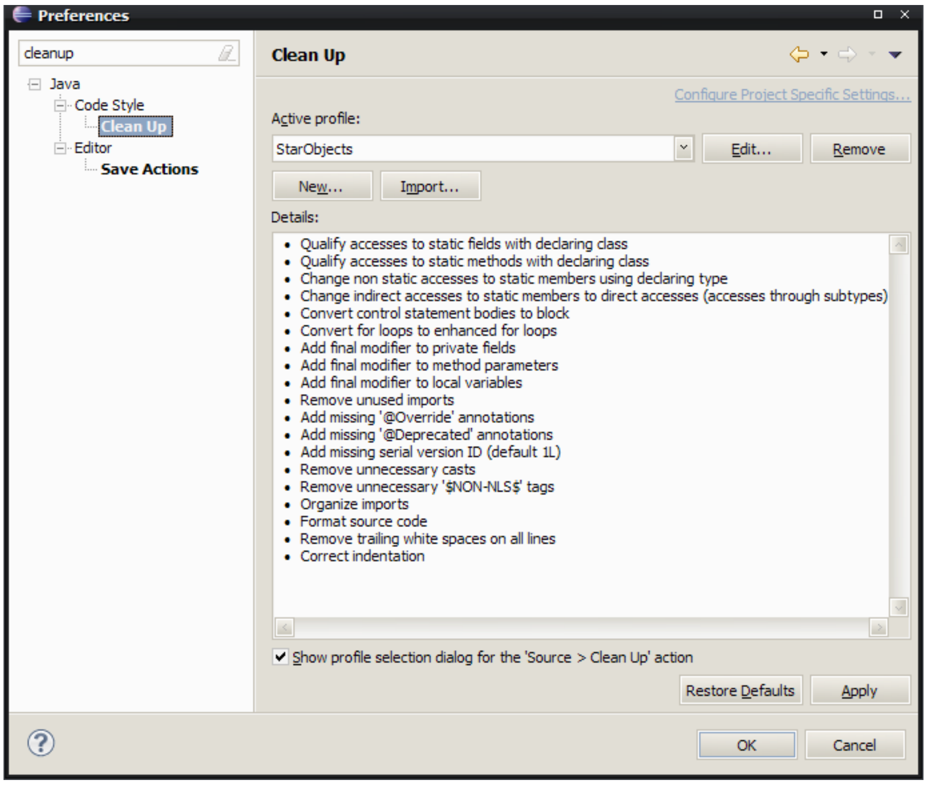


```
/**
 * A sample source file for the code formatter preview
 */

package mypackage;

import java.util.LinkedList;

public class MyIntStack {
    private final LinkedList fStack;

    public MyIntStack() {
        fStack = new LinkedList();
    }

    public int pop() {
        return ((Integer) fStack.removeFirst()).intValue();
    }

    public void push(int elem) {
        fStack.addFirst(new Integer(elem));
    }
```

To run the formatter, use Source > Format. Alternatively, you can use Cleanup, below, which performs formatting and a number of other checks too.

Cleanup

The Eclipse cleanup config file (accessible here) defines a set of operations aimed at removing simple problems with the code, again consistent with the Checkstyle checks (see Section 4.1, “Checkstyle”). These include running the formatter, above. The config file should be imported using Windows > Preferences > Java > Code Style > Cleanup:



- Qualify accesses to static fields with declaring class
- Qualify accesses to static methods with declaring class
- Change non static accesses to static members using declaring type
- Change indirect accesses to static members to direct accesses (accesses through subtypes)
- Convert control statement bodies to block
- Convert for loops to enhanced for loops
- Add final modifier to private fields
- Add final modifier to method parameters
- Add final modifier to local variables
- Remove unused imports
- Add missing '@Override' annotations
- Add missing '@Deprecated' annotations
- Add missing serial version ID (default 1L)
- Remove unnecessary casts
- Remove unnecessary '$NON-NLS$' tags
- Organize imports
- Format source code
- Remove trailing white spaces on all lines
- Correct indentation

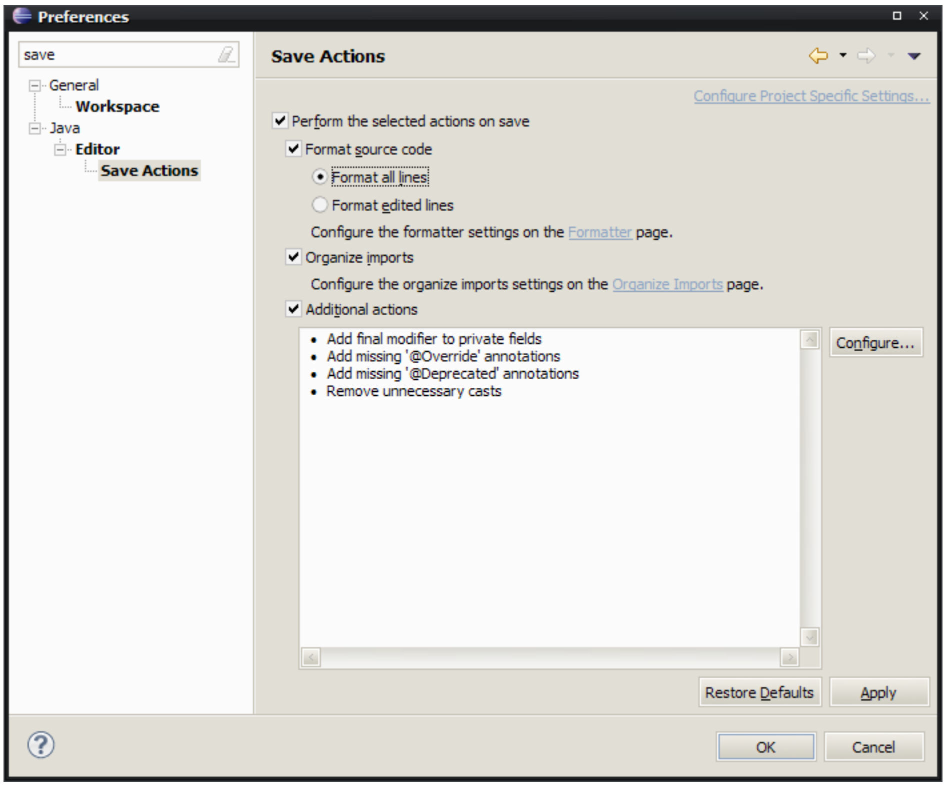
✓ Show profile selection dialog for the 'Source > Clean Up' action

The cleanup wizard can be run on an adhoc basis using Source > Cleanup.

Save Actions

While the formatter and cleanup can be run manually, it is also possible to run the formatter whenever code is saved.

Navigate to Windows > Preferences > Save > Actions and enable:



[✓] Perform the selected actions on save

[✓] Format source code

● Format all lines

○ Format edited lines

Configure the formatter settings on the Formatter page.

[✓] Organize imports

Configure the organize imports settings on the Organize Imports page.

[✓] Additional actions

- Add final modifier to private fields
- Add missing '@Override' annotations
- Add missing '@Deprecated' annotations
- Remove unnecessary casts

From the dialog it is also possible to automatically perform the same actions as the cleanup (the Configure button). However, it doesn't seem possible to reuse the configuration definition file. To avoid violating the DRY principle, so the recommendation is just to enable the simple formatting on save and not waste time manually configuring cleanup actions on save too.

Chapter 5

Templates and Utilities

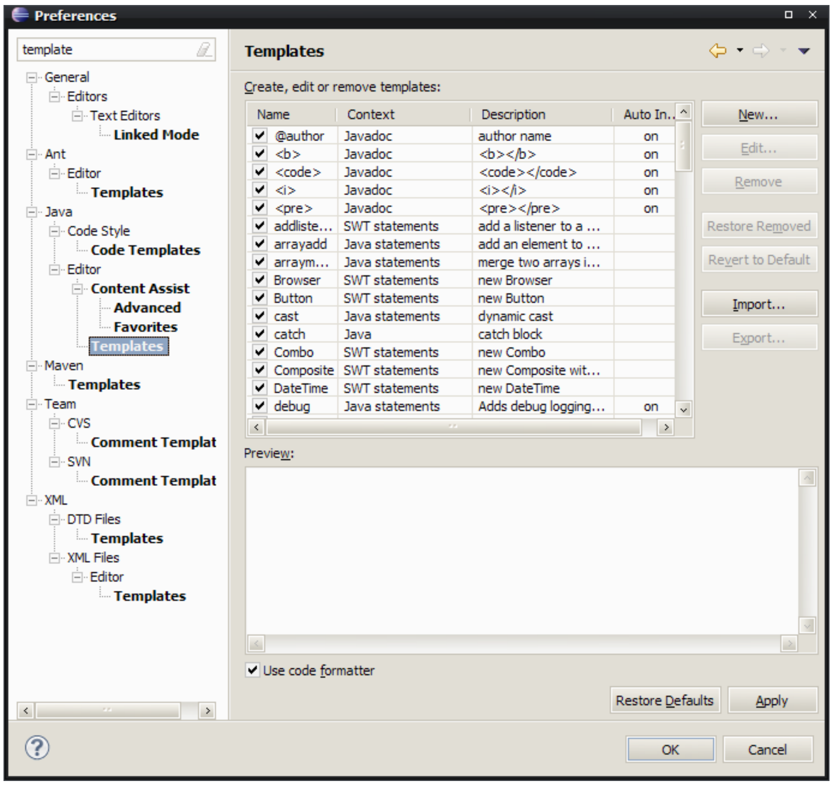
This chapter describes how to set up templates, utilities and any other productivity aids. Follow the steps in this chapter if you intend to contribute back to the sister projects. There's no need to perform these steps if you are just building the sister projects from source.

All of the template files described here are checked into Subversion, (at main/standards/src/main/resources). To ease distribution they are uploaded to the Star Objects sourceforge website. The templates are not versioned (other than by Subversion itself).

5.1. Eclipse Java Editor Templates

Like most IDEs, Eclipse allows templates to be defined for common code snippets. Star Objects provides a number of templates to assist with testing.

To import templates, go to Windows > Preferences > Java > Editor > Templates:



Create, edit or remove templates:

	Name	Context	Description	Auto In...
✓	@author	Javadoc	author name	on
✓	<b>	Javadoc	<b></b>	on
✓	<code>	Javadoc	<code></code>	on
✓	<i>	Javadoc	<i></i>	on
✓	<pre>	Javadoc	<pre></pre>	on
✓	addliste...	SWT statements	add a listener to a ...	
✓	arrayadd	Java statements	add an element to ...	
✓	arraym...	Java statements	merge two arrays i...	
✓	Browser	SWT statements	new Browser	
✓	Button	SWT statements	new Button	
✓	cast	Java statements	dynamic cast	
✓	catch	Java	catch block	
✓	Combo	SWT statements	new Combo	
✓	Composite	SWT statements	new Composite wit...	
✓	DateTime	SWT statements	new DateTime	
✓	debug	Java statements	Adds debug logging...	on

✓ Use code formatter

JUnit 4 Templates

The JUnit4 templates provide a selection of templates to ease the writing of JUnit4-based unit tests.

The definition file is available at http://starobjects.sourceforge.net/m2-site/main/standards/resources/eclipse-junit4-templates.xml.

The templates provided are:

- *jubefore* - creates `setUp` method annotated with `@Before`
- *juafter* - create `tearDown` method annotated with `@After`
- *jutest* - create method annotated with `@Test`
- *juassert* - create `assertThat` assertion

JMock 2 Templates

The JMock 2 templates provide a selection of templates to ease the writing of unit test that perform mocking using JMock 2.

The definition file is available http://starobjects.sourceforge.net/m2-site/main/standards/resources/eclipse-jmock2-templates.xml.

The templates provided are:

- *jmrunwith* - `@RunWith(JMock.class)`
- *jmcontext* - initialize Mockery `context` object
- *jmmock* - create a mock using `context.mock(Foor.class)`
- *jmexpectations* - create a set of expectations using `context.checking(...)`

Chapter 6
Using the Corporate POM

This chapter describes how the code should use the corporate POM as a parent. It is applicable for contributors to any of the sister projects. Contributors to Star Objects who want to build and deploy the corporate POM itself should refer to Chapter 12, The Corporate POM.

It's common practice to use a "corporate" POM (sometimes also inaccurately known as a super POM) to act as a common parent for a group of related projects. As such, the Star Objects provides a corporate POM - `org.starobjects.star:corporate` - for the sister projects that defines common dependency versions and configuration for build plugins and reporting plugins. This corporate POM also inherits from `org.nakedobjects:release:4.0.x` so implicitly defines the dependencies for Naked Objects itself.

Because the corporate POM is only a single pom artifact and is independent of any given project, it is numbered 1, 2, 3,... There is also no use of the snapshot mechanism. This is consistent with the approach taken by Apache themselves (at the time of writing Apache's own corporate POM was ~14).

This chapter assumes that the corporate POM has been released (as described in Chapter 12, *The Corporate POM*).

6.1. Defining the Corporate POM

The corporate POM is `org.starobjects.star:corporate`. The `pom.xml` in the main module (see the section called “The 'main' module”) should define it as a parent:

```
<parent>
    <groupId>org.starobjects.star</groupId>
    <artifactId>corporate</artifactId>
    <version>1</version>
</parent>
```

Typically the version should be the highest available. You can browse the Star Objects repository (see Section 6.2, “Defining the Corporate Repository” below) to see which is the highest.

6.2. Defining the Corporate Repository

The corporate POM is always available from Star Objects' own Maven repository, hosted at http://starobjects.sourceforge.net/m2-repo. In order to access the corporate POM, a bootstrap reference to this repository is required:

```
<repositories>
  <!-- bootstrap parent corporate POM -->
  <repository>
    <id>starobjects-release</id>
    <url>http://starobjects.sourceforge.net/m2-repo/release/</url>
    <snapshots>
      <enabled>false</enabled>
    </snapshots>
    <releases>
      <enabled>true</enabled>
    </releases>
  </repository>
</repositories>
```

Note, the corporate POM should also be available from the Maven central repo.

Chapter 7

Writing DocBook Documentation

This chapter describes how to write and then build DocBook documentation. It is applicable for contributors who wish to update the documentation and check the result of their changes.

All of the sister projects provide supporting documentation, typically a developers' guide (`developers-guide.xml`) and a users' guide (`users-guide.xml`). Star Objects itself has only a developers guide, namely this document.

All of these documents are written in DocBook, and are built using Maven, using build plugins defined by the corporate POM (see Part IV, “Building and Deploying the Corporate Artifacts”).

This chapter explains how to write and built this documentation, using Star Objects' own documentation as a guide.

7.1. Source Code Repository

Generally speaking the documentation for all sister projects lives in that project's sourceforge repository under `trunk/main/documentation`. For example, the documentation for the Star Objects project resides in the Star Objects repository under trunk/main/documentation.

As always, use Subversion to check out. For example:

```
svn co https://starobjects.svn.sourceforge.net/svnroot/starobjects/trunk/main/documentation ~/
starobjects/trunk/main/documentation
```

7.2. Editing the Documentation (using XMLMind)

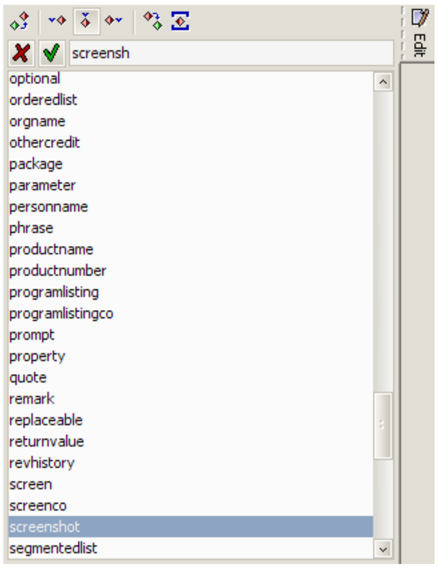
XMLMind XML Editor

The documentation are written in XML using the DocBook dialect. As such you can edit the text with any text editor. However, you may find it easier to use an editor; the one I use is from XMLMind. The personal edition is free for use on open source projects.

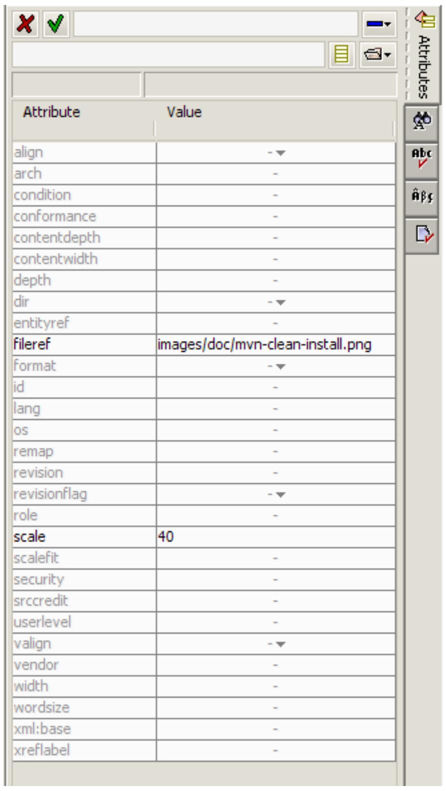
Images

Images should be saved as `.png` files, under the `images` subdirectory (relative the directory holding `xxx-guide.xml`).

Images should be embedded into the documentation using a `screenshot/mediaobject/imageobject/imagedata` tag. In XMLMind, use Edit>Insert and select screenshot:



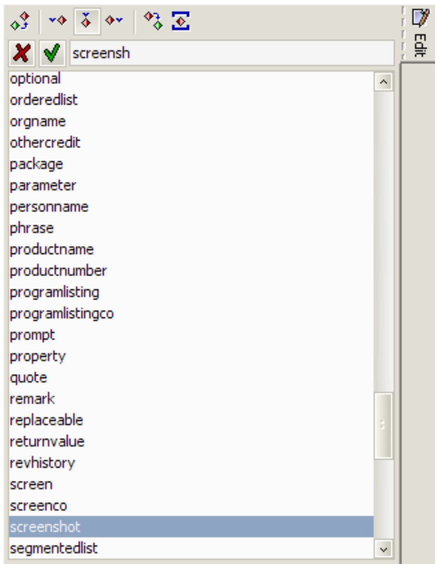
The outer `imagedata` tag should have an appropriate `scale` attribute; a value of 50 (note: *not* 50%) works reasonably well:



Attribute	Value
align	-
arch	-
condition	-
conformance	-
contentdepth	-
contentwidth	-
depth	-
dir	-
entityref	-
fileref	images/doc/mvn-clean-install.png
format	-
id	-
lang	-
os	-
remap	-
revision	-
revisionflag	-
role	-
scale	40
scalefit	-
security	-
srccredit	-
userlevel	-
valign	-
vendor	-
width	-
wordsize	-
xml:base	-
xreflabel	-

Tables

In XMLMind, use Edit > Insert, then select one of table, table(head_column), table(head_row), table(head_row_column):



screensh

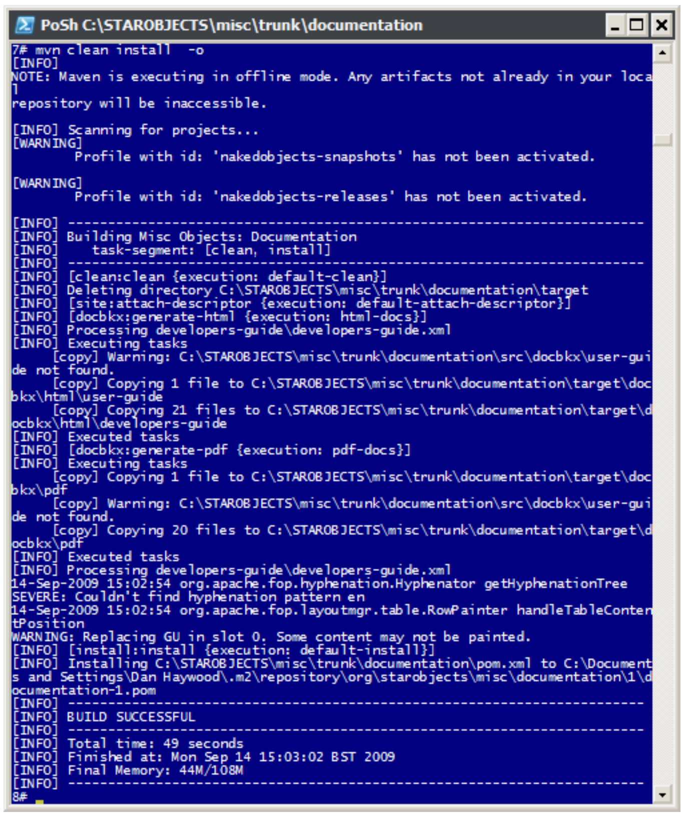
optional
orderedlist
orgname
othercredit
package
parameter
personname
phrase
productname
productnumber
programlisting
programlistingco
prompt
property
quote
remark
replaceable
returnvalue
revhistory
screen
screenco
screenshot
segmentedlist

This will create a 2x2 table body with a header row and/or column if requested. Then use DocBook > Column > Insert or DocBook > Row > Insert to adjust the number of columns and rows as required.

7.3. Build the Documentation

The documentation for all sister projects can be built with Maven in the usual way. For example, to build Star Objects' own documentation, use:

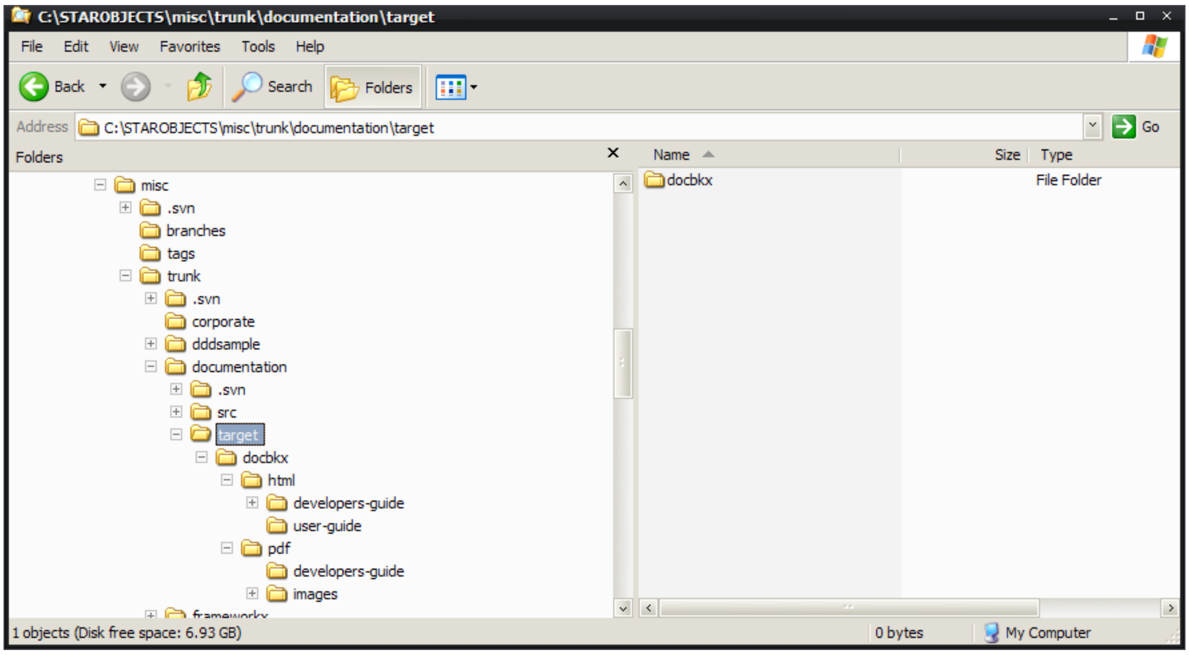
```
cd ~/starobjects/trunk/main/documentation
mvn clean install
```



PoSh C:\STAROBJECTS\misc\trunk\documentation

```
7# mvn clean install  -o
[INFO]
NOTE: Maven is executing in offline mode. Any artifacts not already in your loca
l
repository will be inaccessible.

[INFO] Scanning for projects...
[WARNING]
        Profile with id: 'nakedobjects-snapshots' has not been activated.

[WARNING]
        Profile with id: 'nakedobjects-releases' has not been activated.

[INFO] ------------------------------------------------------------------------
[INFO] Building Misc Objects: Documentation
[INFO]    task-segment: [clean, install]
[INFO] ------------------------------------------------------------------------
[INFO] [clean:clean {execution: default-clean}]
[INFO] Deleting directory C:\STAROBJECTS\misc\trunk\documentation\target
[INFO] [site:attach-descriptor {execution: default-attach-descriptor}]
[INFO] [docbkx:generate-html {execution: html-docs}]
[INFO] Processing developers-guide\developers-guide.xml
[INFO] Executing tasks
     [copy] Warning: C:\STAROBJECTS\misc\trunk\documentation\src\docbkx\user-gui
de not found.
     [copy] Copying 1 file to C:\STAROBJECTS\misc\trunk\documentation\target\doc
bkx\html\user-guide
     [copy] Copying 21 files to C:\STAROBJECTS\misc\trunk\documentation\target\d
ocbkx\html\developers-guide
[INFO] Executed tasks
[INFO] [docbkx:generate-pdf {execution: pdf-docs}]
[INFO] Executing tasks
     [copy] Copying 1 file to C:\STAROBJECTS\misc\trunk\documentation\target\doc
bkx\pdf
     [copy] Warning: C:\STAROBJECTS\misc\trunk\documentation\src\docbkx\user-gui
de not found.
     [copy] Copying 20 files to C:\STAROBJECTS\misc\trunk\documentation\target\d
ocbkx\pdf
[INFO] Executed tasks
[INFO] Processing developers-guide\developers-guide.xml
14-Sep-2009 15:02:54 org.apache.fop.hyphenation.Hyphenator getHyphenationTree
SEVERE: Couldn't find hyphenation pattern en
14-Sep-2009 15:02:54 org.apache.fop.layoutmgr.table.RowPainter handleTableConten
tPosition
WARNING: Replacing GU in slot 0. Some content may not be painted.
[INFO] [install:install {execution: default-install}]
[INFO] Installing C:\STAROBJECTS\misc\trunk\documentation\pom.xml to C:\Document
s and Settings\Dan Haywood\.m2\repository\org\starobjects\misc\documentation\1\d
ocumentation-1.pom
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESSFUL
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 49 seconds
[INFO] Finished at: Mon Sep 14 15:03:02 BST 2009
[INFO] Final Memory: 44M/108M
[INFO] ------------------------------------------------------------------------
8#
```

This does build an pom artifact, with an artifact Id of documentation in the local repository (for example, the Star Objects project's own documentation artifact is `org.starobjects.star:documentation`). However, this isn't what we're interested in. Instead, what we care about is the generated HTML website and PDF documents in `target/docbkx`:



C:\STAROBJECTS\misc\trunk\documentation\target

```
misc
  .svn
  branches
  tags
  trunk
    .svn
    corporate
    dddsample
    documentation
      .svn
      src
      target
        docbkx
          html
            developers-guide
            user-guide
          pdf
            developers-guide
            images
    frameworkx
```

docbkx File Folder

1 objects (Disk free space: 6.93 GB) 0 bytes My Computer

7.4. Deploying the Documentation

The generated documentation is deployed as part of the site, basically by copying the generated artifacts from the `target/docbkx` directory into `target/site/docbkx` directory. See the section called “Menu items linking to Documentation (parent modules only)” for more details.

Chapter 8

Writing Site Documentation

This chapter describes how to write and to check site documentation for any of the sister projects. Follow the steps in this chapter if you intend to contribute back to the sister projects and will be writing such documentation. There's no need to perform these steps if you are just building the sister projects from source.

Maven provides the ability to automatically create a website holding various reports, such as Checkstyle, PMD, test results and code coverage.

The reporting plugins in the corporate POM (see Chapter 6, *Using the Corporate POM*) go a long way to defining the contents of these sites; since every sister project uses the corporate POM that also means that the sites will have the same general contents.

Maven sites can also be customized and supplemented with additional details. This chapter describes how to perform this customization and where to place this documentation. It also describes how the sister project site should be skinned using a standard Maven skin, common for all sister projects. This chapter assumes that the Maven skin has been released (as described in Chapter 13, *The Maven Skin*).

8.1. Overview

Customization for Maven sites is accomplished by:

- customizing the menu bar and general look-n-feel (see Section 8.3, "Customizing the Site Descriptor (site.xml)")

  This is done by editing the site descriptor for each module, namely, `src/site/site.xml`. The exact content of the site descriptor varies, generally parent modules (and standalone modules) will have more content than child modules. This is in part because child modules inherit certain information from the parent.

- writing additional site content (see Section 8.4, "Writing Additional Site Content")

The additional content lives under `src/site/xxx`, where `xxx` represents the file format. These files can be linked to from the menu bar using `site.xml`. Any document that corresponds to `index.html` (for example `src/site/apt/index.apt`) need not be explicitly linked to because it will be the default for the site.

- specifying the list of reports to produce (see Section 8.5, "Generated Reports")

  This is done in the `pom.xml`. For sister projects the configuration is inherited from the corporate POM.

The following sections provide further detail on these steps.

For details on actually building, checking and deploying the site, see Chapter 11, *Deploying the Site*.

8.2. Source Code Repository

Generally speaking the documentation for all sister projects lives in that project's sourceforge repository under `trunk/main/documentation`. For example, the documentation for the *Star Objects* project resides in the *Star Objects* repository under trunk/documentation.

As always, use Subversion to check out. For example:

```
svn co https://starobjects.svn.sourceforge.net/svnroot/starobjects/trunk/main/documentation ~/
starobjects/trunk/main/documentation
```

8.3. Customizing the Site Descriptor (site.xml)

Generally speaking sister projects should have site documentation alongside the code artifacts for each Maven module (so that there is a `src/site` directory alongside `src/main/java` and `src/test/java`).

The parent module is the top-level module that contains the main `index.html` entry page along with a hierarchy of subsites corresponding to the submodules defined in the `<modules>` element of the POM, for example:

```
<modules>
    <module>fixtures</module>
    <module>documentation</module>
</modules>
```

The child modules are then the modules referenced by the parent module.

Some sister projects may have separate standalone modules that just contain site content (typically with no code artifacts of their own). These can be thought of as parent modules with no children modules.

The customization required depends on whether the site is a parent/standalone module or a child module.

Parent Module and Standalone Modules

Project Name

The project name should use `${project.name}`, being taken from the `pom.xml`.

```
<project name="${project.name}">
```

```
   ...
</project>
```

The name will appear in any child modules sites linking back to the parent module (see the section called “Body and Menu Items”). It must be short enough to fit in the space provided.

Version Position and Banner

The version should be positioned to the right, and the logo should use a version of `hal-logo-for-maven-site.jpg` customised for the sister project (there is a `hal-logo-for-maven-site.pdn` Paint.NET master available in the Star Objects umbrella project):

```
<project name="${project.name}">
  <version position="right"/>

  <bannerLeft>
    <name>HAL</name>
    <src>images/hal-logo-for-maven-site.jpg</src>
    <href>.</href>
  </bannerLeft>

  ...
</project>
```

Powered By Logo

The `<poweredBy>` element (which adds a logo at the bottom of the menu bar) should be specified to use the `NO-powered-by-logo.png`:

```
<project name="${project.name}">
  ...
  <poweredBy>
     <logo name="Naked Objects" href="http://nakedobjects.org"
           img="images/NO-powered-by-logo.png"/>
  </poweredBy>
  ...
</project>
```

Note that the theme expects that the `.png` image should be scalable to 100 x 62 (see `maven-theme.css` in Star Objects' `trunk/skin/src/main/resources/css`).

Skin

The skin should be specified as the corporate Maven skin:

```
<project name="${project.name}">
  ...
  <skin>
    <groupId>org.starobjects.star</groupId>
    <artifactId>skin</artifactId>
    <version>1</version>
  </skin>
  ...
</project>
```

Note that section Chapter 13, *The Maven Skin* of this document describes how this skin is deployed. Sister projects should be able to take it for granted that this skin is available.

Body and Menu Items

The `<body>` element defines menu items (and optionally links)

```
<project name="${project.name}">
  ...
  <body>
    ...
  </body>
</project>
```

Link items (parent modules only)

(For parent modules), link item (appearing in the top border) should reference Naked Objects and other sister projects:

```
  <body>
     <links>
       <item name="Naked Objects" href="http://www.nakedobjects.org/"/>
       <item name="Sister Projects" href="http://starobjects.sourceforge.net"/>
      </links>
    ...
  </body>
```

Menu item linking to Child Modules (parent modules only)

(For parent modules), a menu item should reference child modules, providing navigability to the subsites of these modules:

```
  <body>
    ...
    <menu ref="modules" inherit="top"/>
    ...
  </body>
```

Menu item linking to Module-specific Content

If there is any specific content (such as overview pages, screencasts, FAQ pages and so on) then they should be referenced here. At the time of writing none of the sister projects had this requirement, but the Naked Objects site, also built with Maven, does show how:

```
  <body>
    ...
    <menu name="Developing with Naked Objects">
      <item name="Forums" href="http://sourceforge.net/projects/nakedobjects/forums/" />
      <item name="Blog/News" href="http://blog.nakedobjects.org/" />
      <item name="Manual" href="manual.html" collapse="true">
        <item name="Tutorial" href="tutorial.html" />
        <item name="Application manual" href="application.html" />
        <item name="Filename conventions" href="file-nameconvention.html" />
        <item name="Installing NOF" href="installing-nof.html" />
      </item>
    </menu>
    ...
  </body>
```

Menu item linking to Reports (if source code)

Neither parent sites nor standalone sites are expected to include source code. If they do, though, then a menu item to reference generated reports should be added:

```
  <body>
    ...
    <menu ref="reports"/>
    ...
  </body>
```

Menu items linking to Documentation (parent modules only)

Many parent modules will have a documentation (sub)module which is used to create DocBook documentation (as discussed in Chapter 7, *Writing DocBook Documentation*). The following links allow the generated documentation to be referenced directly from the parent site:

```
  <body>
    ...
    <menu name="Documentation">
      <item name="User Guide (PDF)"
        href="documentation/docbkx/pdf/user-guide.pdf"/>
      <item name="User Guide (HTML)"
        href="documentation/docbkx/html/user-guide/user-guide.html"/>
      <item name="Developers Guide (PDF)"
        href="documentation/docbkx/pdf/developers-guide.pdf"/>
      <item name="Developers Guide (HTML)"
        href="documentation/docbkx/html/developers-guide/developers-guide.html"/>
    </menu>
    ...
  </body>
```

These links pick up generated documentation that is copied from each the documentation project's `target/docbkx` directory into its `target/site/docbkx` directory. The copying itself is defined in the corporate POM, as part of the `<postProcess>` configuration of the docbkx-maven-plugin plugin:

```
<plugin>
  <groupId>com.agilejava.docbkx</groupId>
  <artifactId>docbkx-maven-plugin</artifactId>
  <version>2.0.8</version>
  ...
  <executions>
    <execution>
      <id>html-docs</id>
      ...
      <configuration>
        ...
        <postProcess>
          <copy todir="target/site/docbkx/html" failonerror="false">
             <fileset dir="target/docbkx/html">
               <include name="**/*"/>
             </fileset>
          </copy>
        </postProcess>
      </configuration>
    </execution>

    ...
  </executions>
</plugin>
```

There's a similar `<postProcess>` element for PDF documents too.

Note that some sister projects may only have user guide and/or documentation guide. For example, Star Objects umbrella project only provides a developers guide (this guide!). In such cases, only add the links required.

Menu items linking to Project Resources

Project resources provide pointers to resources hosted elsewhere (typically SourceForge). For example, Tested Objects has:

```
  <body>
```

```
    ...
    <menu name="Project Resources">
      <item name="SF Trac Wiki" href="http://sourceforge.net/apps/trac/testedobjects"/>
      <item name="SF Project Page" href="http://sourceforge.net/projects/testedobjects"/>
    </menu>
    ...
  </body>
```

The "Maven Repo" link perhaps needs a little more explanation. Each of the sister projects has its own staging Maven repository on SourceForge. The "Maven Repo" link allows this repository to be browsed.

Menu items linking to Sister Projects

All sister projects should have a set of menu items pointing to other sister projects (home page, and the Maven repo):

```
  <body>
    ...
    <menu name="Sister Projects">
      <item name="Home" href="http://starobjects.sourceforge.net"/>
      <item name="Maven Repo" href="http://sourceforge.net/projects/starobjects/m2-repo"/>
    </menu>
    ...
  </body>
```

Menu items linking to any other resources

Other resources provide pointers to related projects, blogs and so forth, also hosted elsewhere. The standard set for all sister projects are:

```
  <body>
    ...
    <menu name="Other Resources">
      <item name="Dan Haywood's blog" href="http://danhaywood.com"/>
      <item name="Naked Objects" href="http://www.nakedobjects.org/"/>
      <item name="Naked Objects for .NET" href="http://www.nakedobjects.net/"/>
      <item name="Naked Objects blog" href="http://blog.nakedobjects.org/"/>
      <item name="Scimpi" href="http://www.scimpi.org/" />
    </menu>
  </body>
```

Child Modules

Child modules inherit site definitions from the parent, but override the `<body>` element to customize the contents of the menu items. Pretty much everything else (such as the logo and skin definition) is unchanged.

Project Name

The project name should use `${project.name}`, being taken from the `pom.xml`.

```
<project name="${project.name}">
   ...
</project>
```

The name will appear in any child sites linking back to the parent module (see the section called “Parent Module and Standalone Modules”), so it must be short enough to fit in the space provided.

Body and Menu Items

The `<body>` element defines menu items (and optionally links)

```
<project name="${project.name}">
  ...
  <body>
    ...
  </body>
</project>
```

Menu link referencing the Parent Module

The child modules should reference the parent module, providing navigability back from the child's subsite to the parent's site:

```
  <body>
    <menu ref="parent"/>
    ...
  </body>
```

Menu item linking to Module-specific Content

If there is any specific content for the child module then they should be referenced here. At the time of writing none of the sister projects had this requirement, but the Naked Objects site, also built with Maven, does show how:

```
  <body>
    ...
    <menu name="Developing with Naked Objects">
      <item name="Forums" href="http://sourceforge.net/projects/nakedobjects/forums/" />
      <item name="Blog/News" href="http://blog.nakedobjects.org/" />
      <item name="Manual" href="manual.html" collapse="true">
        <item name="Tutorial" href="tutorial.html" />
        <item name="Application manual" href="application.html" />
        <item name="Filename conventions" href="file-nameconvention.html" />
        <item name="Installing NOF" href="installing-nof.html" />
      </item>
    </menu>
    ...
  </body>
```

Links to Generated Reports

All child modules are expected to contain source code. They should therefore include a reference to generated reports from that source code:

```
  <body>
    ...
    <menu ref="reports"/>
  </body>
```

Documentation Links (documentation child module only)

Since all sister projects should have a documentation module, this module should also link to the documentation. The text for this is similar to that on the parent module, but one level deeper in the directory (ie remove the "documentation/" from the path):

```
  <body>
    ...
    <menu name="Documentation">
      <item name="User Guide (PDF)"
        href="docbkx/pdf/user-guide.pdf"/>
      <item name="User Guide (HTML)"
        href="docbkx/html/user-guide/user-guide.html"/>
      <item name="Developers Guide (PDF)"
```

```
        href="docbkx/pdf/developers-guide.pdf"/>
      <item name="Developers Guide (HTML)"
        href="docbkx/html/developers-guide/developers-guide.html"/>
    </menu>
    ...
  </body>
```

8.4. Writing Additional Site Content

Additional site content

- documents go in `src/site/xxx`, where `xxx` is the file format.

  For example APT documents live under `src/site/apt`.
- images and other resources go in src/site/resources and are referenced relative to this directory.

  For example images typically live under `src/site/resources/images`.

As mentioned above, any document that corresponds to `index.html` (for example `src/site/apt/index.apt`) need not be explicitly linked to because it will be the default for the site.

File Formats

Maven sites can include documentation in any of a number of file formats:

Table 8.1. File formats supported by Maven site

Format	Type	Location	Reference
APT	Wiki-like format	src/site/apt/xxx.apt	Maven Doxia site (APT reference)
DocBook	Full power of DocBook	src/site/docbook/xxx.xml	DocBook site (quick ref)
FML	FAQ Markup Language	src/site/fml/xxx.xml	Maven Doxia site (FML reference)
XDoc	Simplified DocBook, used in Maven 1.	src/site/xdoc/xxx.xml	Maven Doxia site (XDoc reference)

Of these, APT is the lowest entry, and is the generally recommended format; see the section called “APT Quick Start”.

APT Quick Start

APT reference is available online at http://maven.apache.org/doxia/references/apt-format.html, but the following are some of the main formatting tips.

Sections and Sub-sections

Sections are not indented, paragraphs are.

```
My section title (not indented).
```

```
  My paragraph first line (indented by 2 spaces).  There is no need for remaining
sentences in the paragraph to be indented.  A blank line terminates the paragraph.
```

Subsections can be defined using leading asterisks (`*`) to indicate the subsection level indents:

```
Section title

* Sub-section title

** Sub-sub-section title

*** Sub-sub-sub-section title
```

Fonts

In addition to regular font, we can specify italics, bold or monospaced:

```
  <italicised text>
  <<bold text>>
  <<<monospaced text>>>
```

Lists

List items are indented, and begin with an asterisk (`*`)

```
  * List item 1.

  * List item 2.

    Paragraph contained in list item 2.

    * Sub-list item 1.

    * Sub-list item 2.

  * List item 3.
```

To force the end of a list, use the `[]` pseudo-element:

```
  * List item 3.

  []

  This text is not in the list
```

Links and Figures

To create a hyperlink, use:

```
  Link to {{http://www.pixware.fr}}.
  or
  Link to {{{http://www.pixware.fr}Pixware home page}}.
```

To create an anchor, use:

```
  {Anchor}. This text is anchored.
  and then
  Link to {{anchor}}.
  or
  Link to {{{anchor}showing alternate text}}
```

Figures are specified by

```
  [images/foo/bar.png] Figure caption
```

Code Blocks (verbatim text)

To quote a code block, use 3 dashes (`---`) before and after:

```
----------------------------------------
public class FooBar {
  ...
}
----------------------------------------
```

To put into a box, use a plus symbol (`+--`)

Other Code Elements

In addition to the above, APT supports tables, horizontal rules (===), page breaks, comments and special characters. See the Maven Doxia site for further details.

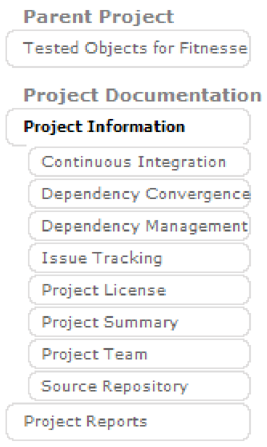
8.5. Generated Reports

The reports are defined by the `<reporting>` section of the corporate POM. These only appear in a module's site if there is a `<menu ref="reports"/>` link; generally this *is* the case for child modules and is *not* the case for parent modules.

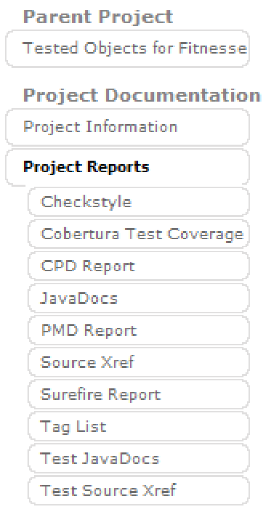
At the time of writing, the reports defined were:

```
<reporting>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-project-info-reports-plugin</artifactId>
      ...
    </plugin>

    <plugin>
       <groupId>org.codehaus.mojo</groupId>
       <artifactId>cobertura-maven-plugin</artifactId>
    </plugin>

    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-surefire-report-plugin</artifactId>
    </plugin>

    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-checkstyle-plugin</artifactId>
      ...
    </plugin>

    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-pmd-plugin</artifactId>
      ...
    </plugin>

    <plugin>
       <groupId>org.codehaus.mojo</groupId>
       <artifactId>taglist-maven-plugin</artifactId>
    </plugin>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-jxr-plugin</artifactId>
      ...
    </plugin>

    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-javadoc-plugin</artifactId>
      ...
    </plugin>
  </plugins>
</reporting>
```

The maven-project-info-reports-plugin plugin generates the submenu under the "Project Information" menu item:



The remaining plugins each generate a submenu item under the "Project Reports" menu item:



8.6. Checking the Site by Deploying it Locally

To check any changes to the site it is necessary to build and deploy it locally.

Prerequisites

See Appendix A, *Deployment Prerequisites*, which describes prerequisite configuration needed prior to performing any deployment.

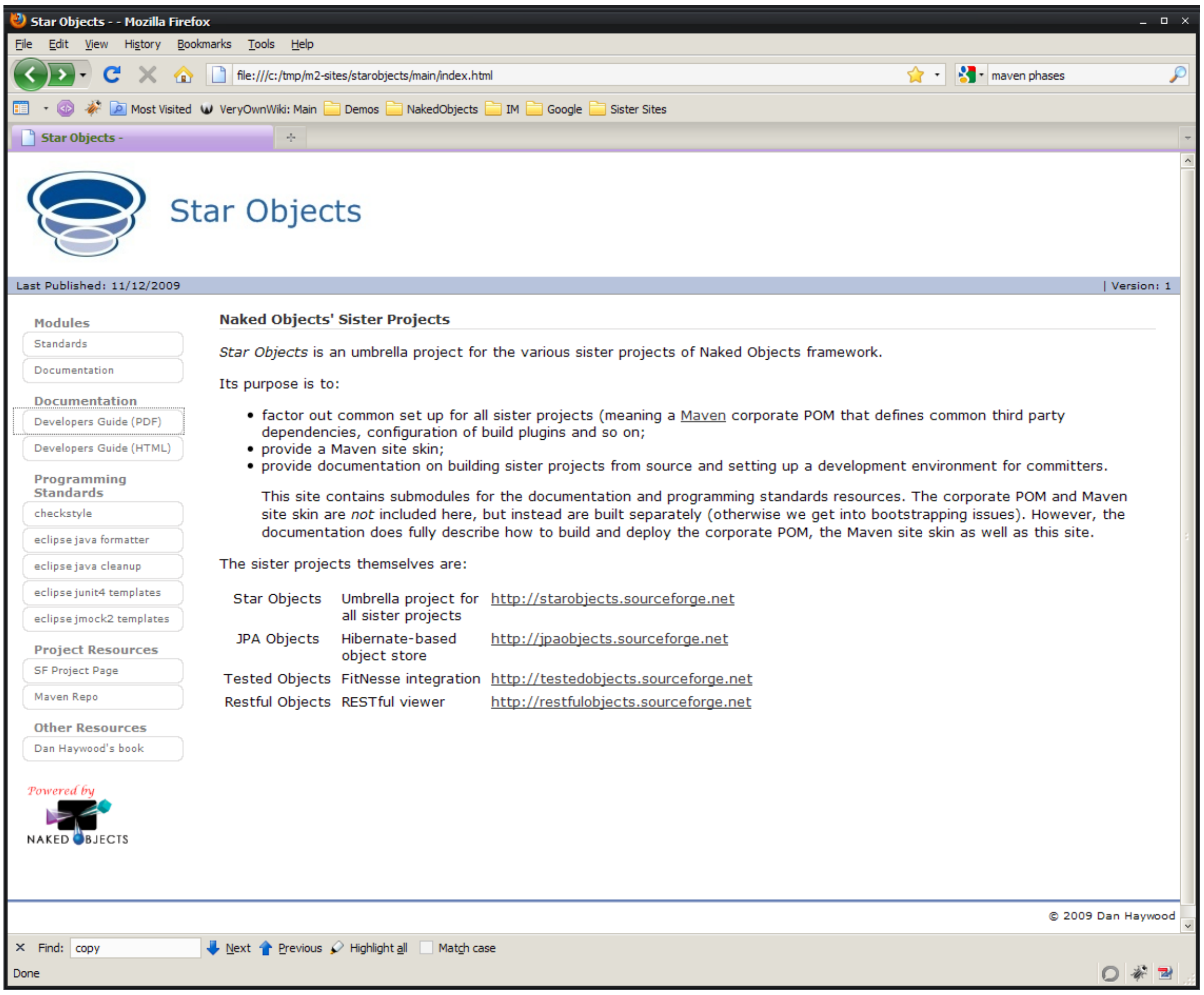
Performing the Local Deploy

To build a site and deploy it locally, use:

```
mvn site-deploy -D dist=local
```

The generated site can be browsed by navigating to the location specified in the corporate POM, namely `/tmp/m2-sites/${distMgmtArtifactId}`. For example, Star Objects' own site is deployed to `/tmp/m2-sites/starobjects`.

The screenshot below shows what a typical generated site looks like:


Star Objects - - Mozilla Firefox
File Edit View History Bookmarks Tools Help
file:///c:/tmp/m2-sites/starobjects/main/index.html
maven phases
Most Visited VeryOwnWiki: Main Demos NakedObjects IM Google Sister Sites
Star Objects -
Find: copy Next Previous Highlight all Match case
Done

Star Objects

Last Published: 11/12/2009 | Version: 1

Modules
Standards
Documentation

Documentation
Developers Guide (PDF)
Developers Guide (HTML)

Programming Standards
checkstyle
eclipse java formatter
eclipse java cleanup
eclipse junit4 templates
eclipse jmock2 templates

Project Resources
SF Project Page
Maven Repo

Other Resources
Dan Haywood's book

Powered by NAKED OBJECTS

Naked Objects' Sister Projects

Star Objects is an umbrella project for the various sister projects of Naked Objects framework.

Its purpose is to:

- factor out common set up for all sister projects (meaning a Maven corporate POM that defines common third party dependencies, configuration of build plugins and so on;
- provide a Maven site skin;
- provide documentation on building sister projects from source and setting up a development environment for committers.

This site contains submodules for the documentation and programming standards resources. The corporate POM and Maven site skin are *not* included here, but instead are built separately (otherwise we get into bootstrapping issues). However, the documentation does fully describe how to build and deploy the corporate POM, the Maven site skin as well as this site.

The sister projects themselves are:

Star Objects	Umbrella project for all sister projects	http://starobjects.sourceforge.net
JPA Objects	Hibernate-based object store	http://jpaobjects.sourceforge.net
Tested Objects	FitNesse integration	http://testedobjects.sourceforge.net
Restful Objects	RESTful viewer	http://restfulobjects.sourceforge.net

© 2009 Dan Haywood

Part III

Release Process

This part of the guide how to release the code or site. It will only be of interest to those with admin privileges.

Chapter 9
Deploying a Code Snapshot

This chapter describes how to release the main module into the snapshot repository. It will only be of interest to contributors with admin privileges to release code.

A snapshot of the code artifacts can be deployed at any time. This chapter describes the steps involved, which are quite straightforward. For details on deploying a code release proper, see Chapter 10, *Deploying a Code Release*.

9.1. Prerequisites

If deploying remotely, check the server settings are defined (see Appendix A, *Deployment Prerequisites*).

9.2. Deploying a Snapshot Manually

Deploying a snapshot means first deploying the main module, and then deploying the support module. Check with the sister projects if there are other modules to deploy.

For local deploys, the `pom.xml` module files deploy to subdirectories under `/tmp`. See Section A.1, "How profile-based distribution management has been designed" for further details.

Deploying the main module

First, build the module:

```
cd ~/xxxobjects/trunk/main
mvn clean install
```

Then, ensure it can be deployed locally:

```
mvn deploy -D dist=local
```

Then, deploy it remotely:

```
mvn deploy -D dist=remote
```

Deploying the support module

The steps for deploying the support module are essentially identical to deploying the main module; just replace 'main' with 'support'.

Chapter 10
Deploying a Code Release

This chapter describes how to perform a release, and to deploy that release remotely. It will only be of interest to contributors with admin privileges to release and deploy code.

The release process tags the codebase under `trunk` into `tags`, updating the versions in all POMs in the process removing the "-SNAPSHOT" suffix. After the release is finished, it may then be deployed . Note that snapshots do not go through this process and can be deployed at any time (see Chapter 9, *Deploying a Code Snapshot*).

The release is performed in two parts: first the main artifacts, then the support artifacts.

10.1. Pre-release Check List

First, ensure there are no oustanding code changes.

Then, perform a smoke test:

- As per Chapter 3, *Building Projects from Source*, build the parent, the main artifacts and the support artifacts from source
- make sure that the archetype runs correctly
- make sure that the documentation is built correctly
- make sure that the site builds correctly

  To check the site builds it is actually necessary to perform a local deploy. See Section 8.6, “Checking the Site by Deploying it Locally” for details.d

Finally, if the release is to be deployed remotely (as is typically the case), check the server settings are defined (see Appendix A, *Deployment Prerequisites*).

10.2. Releasing the Main Module

First we release the main module with submodules.

Release Prepare (Dry Run)

First perform a dry run:

```
cd ~/xxxobjects/trunk/main
mvn release:prepare -D dryRun=true
```

This will walk you through a dry run and you can view the temporary POMs it creates to verify you did everything correctly.

For example, when on 1.0-beta-3-SNAPSHOT the release plugin correctly guesses the release as 1.0-beta-3 and the next snapshot to be 1.0-beta-4-SNAPSHOT. You may need to override the release tag guess, specifically from "main-1.0-beta-3" to just "1.0-beta-3". There's no need for that "main" prefix; it is already specified in the path (and in mvn-release-plugin's tagBase configuration).

As a result of this dry run you'll see the following files:

- `release.properties` - copy of the properties to be used
- `pom.xml.releaseBackup` - the `pom.xml` prior to any changes; identical to `pom.xml` since only done a dryRun
- `pom.xml.tag` - the `pom.xml` as it will look when tagged.
- `pom.xml.next` - the `pom.xml` as it will look for the next iteration.

Release Prepare

When ready, perform the prepare for real:

```
cd ~/xxxobjects/trunk/main
mvn release:clean release:prepare
```

This will again ask for the values release values, defaulting from `release.properties` file:

This will:

- remotely tag the project
- increment the trunk's version, and commit

Deploying the Release

There are two different ways to deploy a release, either using the mvn-release-plugin or manually.

- The mvn-release-plugin's perform goal reads the information in `release.properties`, then checks out the newly created tag, does a `mvn deploy`, and does a `mvn site:deploy`.
- Alternatively the above steps can be performed manually

In this section we concentrate only on the manual approach; it provides full with visibility of the tagged release before finally deploying it.

Pull Down Tag & Build

First, pull down the new tag:

```
cd ~/xxxobjects/tags/main
svn update 1.0-beta-3
```

or whatever the release was just tagged.

Then, make sure that the code builds:

```
mvn clean install
```

Perform any last-minute verification.

Deploy Module

The steps for deploying the release are essentially the same as deploying a snapshot. Essentially the steps are:

- check deployment prerequisites (see Appendix A, *Deployment Prerequisites*)
- switch to the documentation submodule:

  ```
  cd documentation
  ```
- rehearse the deployment by deploying locally, use:

  ```
  mvn deploy -D dist=local
  ```
- when happy, deploy the release remotely:

  ```
  mvn deploy -D dist=remote
  ```

See Chapter 9, *Deploying a Code Snapshot* for more details if required

10.3. Support Module

The steps for deploying the support module are essentially identical to deploying the main module; just replace 'main' with 'support'.

10.4. Deploy Site

You will probably want to deploy the site remotely. See Chapter 11, *Deploying the Site* for details.

Chapter 11

Deploying the Site

This chapter describes how to deploy the site, as well as documentation. It is applicable to those administrators with permissions to upload changes to the new site, and will generally be part of a core release (see Chapter 10, Deploying a Code Release). For contributors who may have made changes and just want to check them, see Section 8.6, "Checking the Site by Deploying it Locally".

Deploying the site also deploys the documentation; as described in Section 8.3, "Customizing the Site Descriptor (site.xml)", the docbkx-maven-plugin also copies the PDF and HTML it generates into the site's target directory.

11.1. Prerequisites

First, check that the locally deployed site is good; see Section 8.6, "Checking the Site by Deploying it Locally"for details.

Second, ensure that any deployment prerequisites have been met, as per Appendix A, *Deployment Prerequisites*. (Basically, you may need to add an entry into `~/.m2/settings.xml` for the server hosting the site).

11.2. Deploying Remotely to Sourceforge

Since we are deploying to sourceforge, we must first create an SSH shell using ssh or putty (if on Windows). Full details are described in http://maven.apache.org/plugins/maven-site-plugin/examples/site-deploy-to-sourceforge.net.html, but what it amounts to is:

- ensure you are in the main submodule:

  ```
  cd ~/xxxobjects/trunk/main
  ```

- if on *NIX, start a local shell and in it run:

```
  ssh -t USER,PROJECT@shell.sourceforge.net create
```

- if on Windows, open up Putty and set the following settings:
  - Session > Host Name : shell.sourceforge.net
  - Session > Connection Type: SSH
  - Connection > SSH > TTY: ensure that "Don't allocate a pseudo-terminal" is NOT checked
  - Connection > SSH: Remote command: create
  - Connection > Data: Auto-login username: USER,PROJECT

Once this has been done, then you should be able to deploy the site remotely:

```
mvn site-deploy -D dist=remote
```

Part IV
Building and Deploying the Corporate Artifacts

The Star Objects project holds a number of artifacts which are referenced by other sister projects. Specfically, these are:

- the corporate POM, which defines common dependency versions and configuration for build plugins and reporting plugins, and
- a maven site skin which provides a standard look-n-feel for Maven sites generated by sister projects
- the 'main' project, which in turn defines:
  - standards - as referenced in Part II, “Contributing Changes”
  - this documentation
  - a Maven website aggregating the above (as well as deploying them for easy access via the web)

These part of the document describes how to build and deploy these "corporate" artifacts. It is only applicable to those with admin permission to release code for the Star Objects project itself. Users and contributors of sister projects should look to Chapter 6, *Using the Corporate POM* and Chapter 8, *Writing Site Documentation*.

Chapter 12 The Corporate POM

This chapter describes how to release the corporate POM. It is only applicable if you have been given admin permissions to release code for the Star Objects project. If you are contributing to other sister projects then you probably want to look at Chapter 6, Using the Corporate POM.

This chapter describes how to build and deploy the corporate POM.

12.1. Source Code Repository

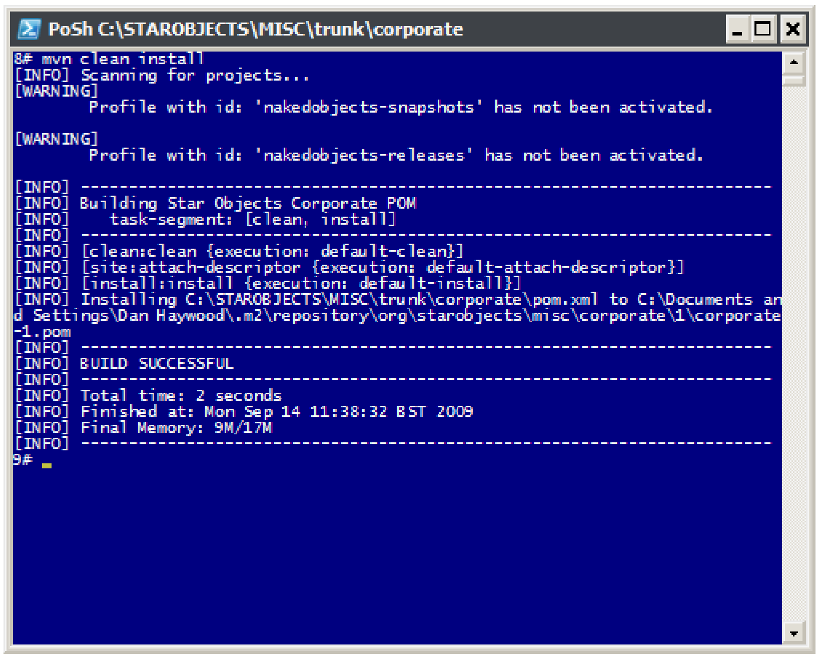
The corporate POM resides in the Star Objects sourceforge repository under trunk/corporate. Use Subversion to check out:

```
svn co https://starobjects.svn.sourceforge.net/svnroot/starobjects/trunk/corporate ~/
starobjects/trunk/corporate
```

12.2. Build the Corporate POM

Build the corporate POM using:

```
cd ~/starobjects/trunk/corporate
mvn clean install
```



```
8# mvn clean install
[INFO] Scanning for projects...
[WARNING]
        Profile with id: 'nakedobjects-snapshots' has not been activated.

[WARNING]
        Profile with id: 'nakedobjects-releases' has not been activated.

[INFO] ------------------------------------------------------------------------
[INFO] Building Star Objects Corporate POM
[INFO]    task-segment: [clean, install]
[INFO] ------------------------------------------------------------------------
[INFO] [clean:clean {execution: default-clean}]
[INFO] [site:attach-descriptor {execution: default-attach-descriptor}]
[INFO] [install:install {execution: default-install}]
[INFO] Installing C:\STAROBJECTS\MISC\trunk\corporate\pom.xml to C:\Documents an
d Settings\Dan Haywood\.m2\repository\org\starobjects\misc\corporate\1\corporate
-1.pom
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESSFUL
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 2 seconds
[INFO] Finished at: Mon Sep 14 11:38:32 BST 2009
[INFO] Final Memory: 9M/17M
[INFO] ------------------------------------------------------------------------
9# _
```

This builds org.starobjects.star:corporate.

12.3. Deploy the Corporate POM

Prerequisites

See Section A.2, “Prerequisites for Deploying Corporate Artifacts (from the umbrella project)”.

Deploy

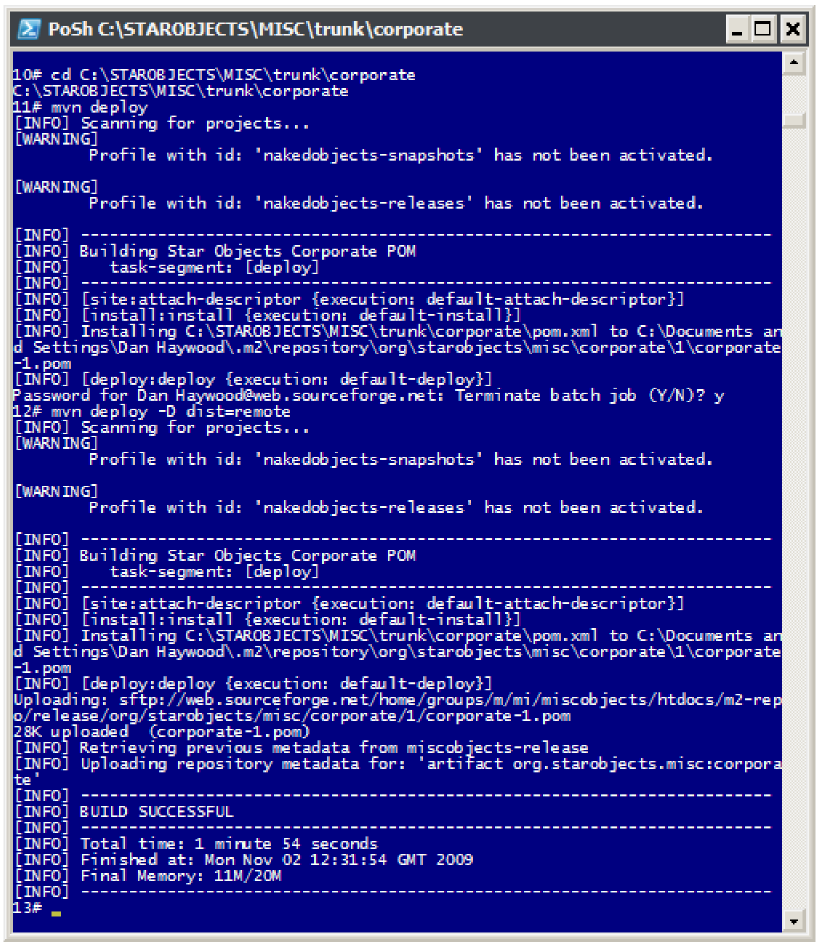
To rehearse the deployment locally, use:

```
cd ~/starobjects/trunk/corporate
mvn deploy -D dist=local
```

This will deploy the corporate POM to `/tmp/m2-repo/starobjects-release`. See Section A.1, “How profile-based distribution management has been designed” for further details.

To deploy remotely, use:

```
cd ~/starobjects/trunk/corporate
mvn deploy -D dist=remote
```



```
10# cd C:\STAROBJECTS\MISC\trunk\corporate
C:\STAROBJECTS\MISC\trunk\corporate
11# mvn deploy
[INFO] Scanning for projects...
[WARNING]
        Profile with id: 'nakedobjects-snapshots' has not been activated.

[WARNING]
        Profile with id: 'nakedobjects-releases' has not been activated.

[INFO] ------------------------------------------------------------------------
[INFO] Building Star Objects Corporate POM
[INFO]    task-segment: [deploy]
[INFO] ------------------------------------------------------------------------
[INFO] [site:attach-descriptor {execution: default-attach-descriptor}]
[INFO] [install:install {execution: default-install}]
[INFO] Installing C:\STAROBJECTS\MISC\trunk\corporate\pom.xml to C:\Documents an
d Settings\Dan Haywood\.m2\repository\org\starobjects\misc\corporate\1\corporate
-1.pom
[INFO] [deploy:deploy {execution: default-deploy}]
Password for Dan Haywood@web.sourceforge.net: Terminate batch job (Y/N)? y
12# mvn deploy -D dist=remote
[INFO] Scanning for projects...
[WARNING]
        Profile with id: 'nakedobjects-snapshots' has not been activated.

[WARNING]
        Profile with id: 'nakedobjects-releases' has not been activated.

[INFO] ------------------------------------------------------------------------
[INFO] Building Star Objects Corporate POM
[INFO]    task-segment: [deploy]
[INFO] ------------------------------------------------------------------------
[INFO] [site:attach-descriptor {execution: default-attach-descriptor}]
[INFO] [install:install {execution: default-install}]
[INFO] Installing C:\STAROBJECTS\MISC\trunk\corporate\pom.xml to C:\Documents an
d Settings\Dan Haywood\.m2\repository\org\starobjects\misc\corporate\1\corporate
-1.pom
[INFO] [deploy:deploy {execution: default-deploy}]
Uploading: sftp://web.sourceforge.net/home/groups/m/mi/miscobjects/htdocs/m2-rep
o/release/org/starobjects/misc/corporate/1/corporate-1.pom
28K uploaded  (corporate-1.pom)
[INFO] Retrieving previous metadata from miscobjects-release
[INFO] Uploading repository metadata for: 'artifact org.starobjects.misc:corpora
te'
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESSFUL
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 1 minute 54 seconds
[INFO] Finished at: Mon Nov 02 12:31:54 GMT 2009
[INFO] Final Memory: 11M/20M
[INFO] ------------------------------------------------------------------------
13# _
```

This deploys the corporate POM to the server referenced in `<distributionManagement>` and specified in your local `settings.xml`.

12.4. Using the Corporate POM

See Chapter 6, *Using the Corporate POM* for details of how the sister projects should actually use (reference) the corporate POM.

Chapter 13
The Maven Skin

This chapter describes how to release the Maven skin. It is only applicable if you have been given admin permissions to release code for the Star Objects project. If you are contributing site documentation to other sister projects then you probably want to look at Chapter 8, Writing Site Documentation (which will reference this site skin from its released location).

Maven provides the ability to automatically create a website holding various reports, such as Checkstyle, PMD, test results and code coverage. Furthermore this site can be skinned using an artifact which provides a set of CSS styles and supporting icons.

So that there is a common look-n-feel for sister projects, we use a single Maven skin. Like the corporate POM, this is managed under the Star Objects project, and is released to the Star Objects Maven repository on the web. Generally then the sister projects simply pick up the skin from this repository.

This part of the document describes how to build this skin. It is applicable only for those (such as administrators) who will be modifying and re-releasing the skin itself.

13.1. Source Code Repository

The Maven skin resides in the *Star Objects* sourceforge repository under trunk/skin. Use Subversion to check out:

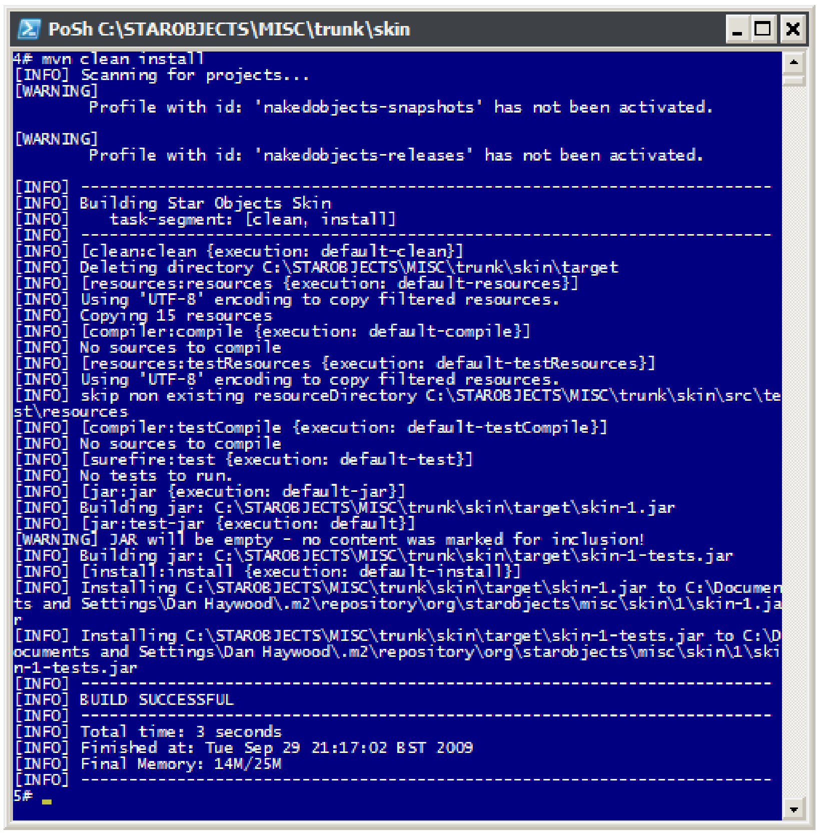
```
svn co https://starobjects.svn.sourceforge.net/svnroot/starobjects/trunk/skin ~/starobjects/
trunk/skin
```

13.2. Build the Skin

Build the skin using:

```
cd ~/starobjects/trunk/skin
```

```
mvn clean install
```



```
4# mvn clean install
[INFO] Scanning for projects...
[WARNING]
        Profile with id: 'nakedobjects-snapshots' has not been activated.

[WARNING]
        Profile with id: 'nakedobjects-releases' has not been activated.

[INFO] ------------------------------------------------------------------------
[INFO] Building Star Objects Skin
[INFO]    task-segment: [clean, install]
[INFO] ------------------------------------------------------------------------
[INFO] [clean:clean {execution: default-clean}]
[INFO] Deleting directory C:\STAROBJECTS\MISC\trunk\skin\target
[INFO] [resources:resources {execution: default-resources}]
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] Copying 15 resources
[INFO] [compiler:compile {execution: default-compile}]
[INFO] No sources to compile
[INFO] [resources:testResources {execution: default-testResources}]
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] skip non existing resourceDirectory C:\STAROBJECTS\MISC\trunk\skin\src\te
st\resources
[INFO] [compiler:testCompile {execution: default-testCompile}]
[INFO] No sources to compile
[INFO] [surefire:test {execution: default-test}]
[INFO] No tests to run.
[INFO] [jar:jar {execution: default-jar}]
[INFO] Building jar: C:\STAROBJECTS\MISC\trunk\skin\target\skin-1.jar
[INFO] [jar:test-jar {execution: default}]
[WARNING] JAR will be empty - no content was marked for inclusion!
[INFO] Building jar: C:\STAROBJECTS\MISC\trunk\skin\target\skin-1-tests.jar
[INFO] [install:install {execution: default-install}]
[INFO] Installing C:\STAROBJECTS\MISC\trunk\skin\target\skin-1.jar to C:\Documen
ts and Settings\Dan Haywood\.m2\repository\org\starobjects\misc\skin\1\skin-1.ja
r
[INFO] Installing C:\STAROBJECTS\MISC\trunk\skin\target\skin-1-tests.jar to C:\D
ocuments and Settings\Dan Haywood\.m2\repository\org\starobjects\misc\skin\1\ski
n-1-tests.jar
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESSFUL
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 3 seconds
[INFO] Finished at: Tue Sep 29 21:17:02 BST 2009
[INFO] Final Memory: 14M/25M
[INFO] ------------------------------------------------------------------------
5#
```

This builds org.starobjects.star:skin.

13.3. Deploying the Skin

Prerequisites

See Section A.2, “Prerequisites for Deploying Corporate Artifacts (from the umbrella project)”.

Deploy

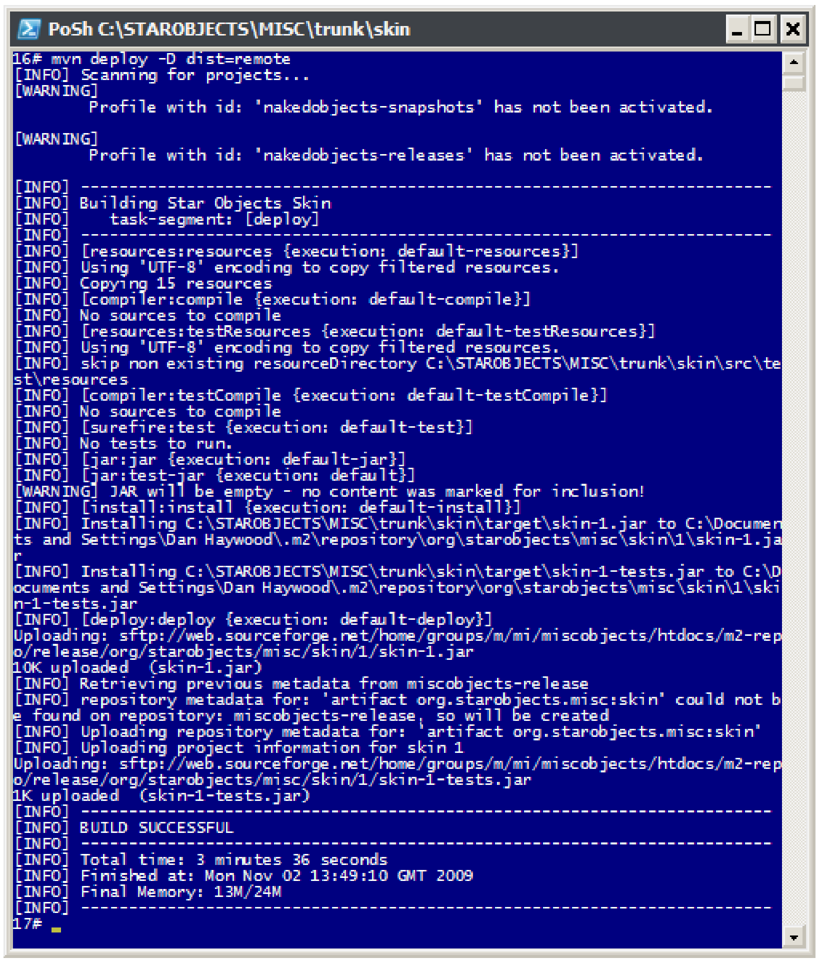
To rehearse the deployment locally, use:

```
cd ~/starobjects/trunk/skin
mvn deploy -D dist=local
```

This will deploy the Maven skin to `/tmp/m2-repo`; see Section A.1, “How profile-based distribution management has been designed” for further details.

To deploy remotely, use:

```
cd ~/starobjects/trunk/skin
mvn deploy -D dist=remote
```



PoSh C:\STAROBJECTS\MISC\trunk\skin

```
16# mvn deploy -D dist=remote
[INFO] Scanning for projects...
[WARNING]
        Profile with id: 'nakedobjects-snapshots' has not been activated.

[WARNING]
        Profile with id: 'nakedobjects-releases' has not been activated.

[INFO] ------------------------------------------------------------------------
[INFO] Building Star Objects Skin
[INFO]    task-segment: [deploy]
[INFO] ------------------------------------------------------------------------
[INFO] [resources:resources {execution: default-resources}]
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] Copying 15 resources
[INFO] [compiler:compile {execution: default-compile}]
[INFO] No sources to compile
[INFO] [resources:testResources {execution: default-testResources}]
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] skip non existing resourceDirectory C:\STAROBJECTS\MISC\trunk\skin\src\te
st\resources
[INFO] [compiler:testCompile {execution: default-testCompile}]
[INFO] No sources to compile
[INFO] [surefire:test {execution: default-test}]
[INFO] No tests to run.
[INFO] [jar:jar {execution: default-jar}]
[INFO] [jar:test-jar {execution: default}]
[WARNING] JAR will be empty - no content was marked for inclusion!
[INFO] [install:install {execution: default-install}]
[INFO] Installing C:\STAROBJECTS\MISC\trunk\skin\target\skin-1.jar to C:\Documen
ts and Settings\Dan Haywood\.m2\repository\org\starobjects\misc\skin\1\skin-1.ja
r
[INFO] Installing C:\STAROBJECTS\MISC\trunk\skin\target\skin-1-tests.jar to C:\D
ocuments and Settings\Dan Haywood\.m2\repository\org\starobjects\misc\skin\1\ski
n-1-tests.jar
[INFO] [deploy:deploy {execution: default-deploy}]
Uploading: sftp://web.sourceforge.net/home/groups/m/mi/miscobjects/htdocs/m2-rep
o/release/org/starobjects/misc/skin/1/skin-1.jar
10K uploaded  (skin-1.jar)
[INFO] Retrieving previous metadata from miscobjects-release
[INFO] repository metadata for: 'artifact org.starobjects.misc:skin' could not b
e found on repository: miscobjects-release, so will be created
[INFO] Uploading repository metadata for: 'artifact org.starobjects.misc:skin'
[INFO] Uploading project information for skin 1
Uploading: sftp://web.sourceforge.net/home/groups/m/mi/miscobjects/htdocs/m2-rep
o/release/org/starobjects/misc/skin/1/skin-1-tests.jar
1K uploaded  (skin-1-tests.jar)
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESSFUL
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 3 minutes 36 seconds
[INFO] Finished at: Mon Nov 02 13:49:10 GMT 2009
[INFO] Final Memory: 13M/24M
[INFO] ------------------------------------------------------------------------
17# _
```

This deploys the Maven skin to the server referenced in `<distributionManagement>` and specified in `settings.xml`.

13.4. Using the Skin

See Chapter 8, *Writing Site Documentation* for details of how the site documentation actually uses (references) the skin.

Chapter 14

The 'Main' Module (Docs & Site)

The overall Maven site, along with documentation and the standards resources (for checkstyle, pmd etc) are built from the 'main' parent module. This module depends on both the corporate POM and the Maven skin, so they must have been deployed first.

There is no code in this main module, so building it and deploying it are straightforward. The POMs for the documentation and standards copy over their generated artifacts so that they are uploaded to the site automatically. The documentation is copied using the described earlier (see the section called “Menu items linking to Documentation (parent modules only)”), the standards resource files rely on some bespoke POM copying.

14.1. Source Code Repository

The 'main' module resides in the *Star Objects* sourceforge repository under trunk/main. Use Subversion to check out:

```
svn co https://starobjects.svn.sourceforge.net/svnroot/starobjects/trunk/main ~/starobjects/
trunk/main
```

14.2. Building the Documentation & Site

Build the documentation using:

```
mvn clean install
```

The site will be deployed to `/tmp/m2-sites/starobjects`.

You may want to combine this with deploying locally (see below) using:

```
mvn clean install site-deploy -D dist=local
```

14.3. Deploying the Site

The steps for deploying the site are the same as for any other sister project. Basically, it boils down to checking the site is okay locally, the deploying using -D dist=remote, with a separate **ssh** session created to sourceforge. Full details are given in Chapter 11, *Deploying the Site*.

Part V
Appendices

This part of the developers guide contain reference appendices.

Appendix A. Deployment Prerequisites

This appendix describes prerequisites for deploying artifacts from both this, the corporate "umbrella" project, and also from other sister projects. It is only applicable if you have been given admin permissions to release code for the Star Objects project.

A.1. How profile-based distribution management has been designed

The corporate POM defines two `<profile>`s that allow deployments to be rehearsed by deploying locally, and then when ready deployed remotely. These contain `<distributionManagement>` tags activated by a property key. This allows us to rehearse a deployment (by deploying locally) of the corporate artifacts before actually doing the deployment proper (deploying remotely).

To deploy locally, we add:

```
-D dist=local
```

And to deploy remotely, we instead add:

```
-D dist=remote
```

All code artifacts, for all sister projects, are deployed to the repositories represented by the starobjects-snapshot or starobjects-release server definitions. However, each sister project has its own site.

Every POM defining artifacts being site-deployed must specify (or inherit) a number of properties. For example, starobjects' own settings are:

```
<properties>
    ...
  <distMgmtArtifactId>starobjects</distMgmtArtifactId>
  <distMgmtArtifactName>Star Objects</distMgmtArtifactName>
  <distMgmtArtifactSfDir>s/st/starobjects</distMgmtArtifactSfDir>
</properties>
```

Similarly, *Tested Objects* has:

```
<properties>
    ...
  <distMgmtArtifactId>testedobjects</distMgmtArtifactId>
  <distMgmtArtifactName>Tested Objects</distMgmtArtifactName>
  <distMgmtArtifactSfDir>t/te/testedobjects</distMgmtArtifactSfDir>
</properties>
```

Other sister projects similarly need to setup these properties.

These properties are then used in `<distributionManagement>`. For a local deploy, we use the following profile:

```
<profile>
  <id>dist-local</id>
  <!-- USE -D dist=local TO SELECT -->
  <activation>
    <property>
      <name>dist</name>
      <value>local</value>
    </property>
  </activation>
  <distributionManagement>
    <snapshotRepository>
      <id>starobjects-snapshot</id>
      <name>Star Objects Local Snapshot Repository</name>
      <url>file:///tmp/m2-repo/snapshot</url>
      <uniqueVersion>false</uniqueVersion>
    </snapshotRepository>
    <repository>
      <id>starobjects-release</id>
      <name>Star Objects Local Release Repository</name>
      <url>file:///tmp/m2-repo/release</url>
    </repository>
    <site>
      <id>${distMgmtArtifactId}-site</id>
      <name>${distMgmtArtifactName} Site</name>
      <url>file:///tmp/m2-sites/${distMgmtArtifactId}</url>
    </site>
  </distributionManagement>
</profile>
```

Any deployment using this profile will put into */tmp*.

Remote deployments on the other hand SFTP/SCP files up to sourceforge:

```
<profile>
  <id>dist-remote</id>
  <!-- USE -D dist=remote TO SELECT -->
  <activation>
    <property>
      <name>dist</name>
      <value>remote</value>
    </property>
  </activation>
  <distributionManagement>
    <snapshotRepository>
      <id>starobjects-snapshot</id>
      <name>Star Objects Snapshot Repository</name>
      <url>
        sftp://web.sourceforge.net/home/groups/s/st/starobjects/htdocs/m2-repo/snapshot
      </url>
      <uniqueVersion>false</uniqueVersion>
    </snapshotRepository>
    <repository>
      <id>snapshot-release</id>
      <name>Star Objects Release Repository</name>
      <url>
        sftp://web.sourceforge.net/home/groups/s/st/starobjects/htdocs/m2-repo/release
      </url>
    </repository>
    <site>
      <id>${distMgmtArtifactId}-site</id>
      <name>${distMgmtArtifactName} Site</name>
      <url>scp://shell.sourceforge.net/home/groups/${distMgmtArtifactSfDir}/htdocs/m2-site</
url>
    </site>
  </distributionManagement>
</profile>
```

For a local deployment to work the directories under /tmp should be exist:

- `/tmp/m2-repo/snapshot` and `/tmp/m2-repo/release` for the repositories
- `/tmp/m2-sites` for the various sites.

For a remote deployment to work requires an entry in the `settings.xml` for the named server id. The umbrella project and all sister projects share the same server Ids for deploying code releases (as per Chapter 10, *Deploying a Code Release*) but every umbrella project will define its own server for deploying its site. In effect this means that the settings.xml will contain:

- starobjects-snapshot

  for both the umbrella project and all sister projects' deployments of code snapshots
- starobjects-release

  for both the umbrella project and all sister projects' deployments of code releases
- starobjects-site

  for the umbrella project's site
- *xxx*objects-site

  for each sister project *xxx*object's site

This is spelt out below

A.2. Prerequisites for Deploying Corporate Artifacts (from the umbrella project)

All corporate artifacts from the umbrella project - corporate POM, Maven skin etc - are deployed to a release repository so that they can be picked up by sister projects (see Section 6.2, “Defining the Corporate Repository” for referencing the corporate POM, and in the section called “Skin” for referencing the Maven skin).

When deploying remotely we need to provide the username and password for the following repositories in the `~/.m2/settings.xml` in order to automate the sftp:

```
<servers>
  <server>
    <id>starobjects-snapshot</id>
    <username>xxx</username>
    <password>xxx</password>
  </server>
  <server>
    <id>starobjects-release</id>
    <username>xxx</username>
    <password>xxx</password>
  </server>
  <server>
    <id>starobjects-site</id>
    <username>xxx</username>
    <password>xxx</password>
  </server>
</servers>
```

A.3. Prequisites for Deploying Sister Project Artifacts

There are similar prerequisites when deploying any of the sister projects that reference the corporate POM.

Every POM defining artifacts being deployed must specify (or inherit) a number of properties. For example, for *Tested Objects* we have:

```
<properties>
    ...
  <distMgmtArtifactId>testedobjects</distMgmtArtifactId>
  <distMgmtArtifactName>Tested Objects</distMgmtArtifactName>
  <distMgmtArtifactSfDir>t/te/testedobjects</distMgmtArtifactSfDir>
</properties>
```

To perform a remote site deploy for *Tested Objects* therefore also required the following is also needed in `~/m2/settings.xml`:

```
<servers>
  <server>
    <id>testedobjects-site</id>
    <username>xxx</username>
    <password>xxx</password>
  </server>
</servers>
```

Other sister projects likewise should follow the same pattern.