

Tested Objects 1.0 Users' Guide FitNesse Integration for Naked Objects 4.0.x Version 0.1

Copyright © 2009 Dan Haywood

Permission is granted to make and distribute verbatim copies of this manual provided that the copyright notice and this permission notice are preserved on all copies.

Preface	v
1. Introduction	1
1.1. Scenario Testing (aka Agile Acceptance Testing)	1
1.2. Introduction to FitNesse	1
1.3. How Tested Objects' Integration Works	2
2. Using the FitNesse Archetype	5
2.1. Prerequisites	5
2.2. Running the Archetype	5
2.3. Starting FitNesse Wiki 10	0
2.4. Setting up to Run Tests 1	1
2.5. Running the Tests 12	2
3. Hints and Tips 1	5
3.1. Separate In-Progress Stories from the Backlog 1	5
3.2. Use a Story Page to Collect A Set of Scenario Tests 15	5
3.3. Organize Completed Stories by Component 1	5
3.4. Structure your test using Given/When/Then 10	6
3.5. Using the RunViewer fixture 10	6
3.6. Factor out common "Given"s 1	6
3.7. Use a Declarative Style for Page Names 1	7
3.8. Run against a real database 1	7
3.9. Set up Continuous Integration 1	7
A. Reference (Online User Guide) 19	9
A.1. Bootstrapping 1	9
A.2. SetUp 22	2
A.3. User Interaction	5
A.4. Debugging 3	0
A.5. Tear Down	2

Preface

<u>Tested Objects</u> is a sister project for the <u>Naked Objects</u> framework, providing integration with <u>FitNesse</u> to enable agile acceptance (or scenario) testing. The integration bundles the FitNesse wiki server and provides a set of generic Fitnesse fixtures that interact with the domain model in the same manner that a Naked Objects viewer does. It also provides a Maven archetype to get you started quickly.

This user guide describes how to use Tested Objects to test your Naked Objects domain applications through FitNesse. Much of this guidance relates to using the Maven archetype; for ease of reference the archetype creates an online user guide within the FitNesse wiki. If you are interested in building Tested Objects from source code (perhaps with a view to contributing to or extending the capabilities of Tested Objects itself) then please see the developers' guide.

Tested Objects is hosted on <u>SourceForge</u>, and is licensed under <u>Apache Software License v2</u>. Naked Objects is also hosted on SourceForge, and is also licensed under Apache Software License v2.

Chapter 1 Introduction

1.1. Scenario Testing (aka Agile Acceptance Testing)

Prior to agile development, requirements gathering for systems was traditionally performed by business analysts discussing requirements with the business, and expressing those requirements in documentation, such as Word specs and perhaps spreadsheets. The acceptance criteria for such requirements were often only sketched out, if at all; it would normally fall to the system testers to write acceptance tests for the requirements, through a mixture of consulting the original (by now out-of-date) requirements documentation and (as often as not) reverse-engineering the implementation.

Scenario testing combines the requirements capture and the acceptance test criteria in a single form, through scenarios. As before, these requirements are in a form that a non-technical domain expert from the business can understand. What differs though is that these scenarios can be used directly exercise the system, and so also represent the acceptance tests for the scenario. Moreover, the results of these tests are rendered in such a way that the business can understand, and thus can help determine if the code is at fault or the test. Once implemented, the acceptance tests also act as a regression suite for the system.

Scenario tests tend to act against a complete system, or sometimes at a subsystem-level. At any rate at a granularity that makes sense to a non-technical businesss person. Compare this to unit testing which exercises the behaviour / method of a single class.

Scenario testing is more usually called "agile acceptance testing", but I find that term rather clumsy. Other names also exist, including behaviour-driven development, and example-driven development. Here we use our own term "scenario testing".

1.2. Introduction to FitNesse

<u>FitNesse</u> is an framework to enable scenario testing. It builds upon the earlier <u>FIT</u> framework, which executes tests expressed in terms of tables. FitNesse sits on top of FIT^1 , allowing the tables to be written within a Wiki, and spawning off FIT to run the tests. The results of the tests are shown as annotated tests, as shown below.

¹In fact, FitNesse as of end 2008 has a replacement test execution architecture, called <u>SLIM</u>.

🗋 Suite Results: ClaimsA	ppSuite 🗙 🕀						Google 👝 🖻 🗙		
(< → C ☆ h	ttp://localhost:9	090/ClaimsAppSui	ite?suite				 D- &- 		
	ClaimsAppSuite								
Suite	Test Page	s: 4 right, 0	wrong, 0 ignore	ed, 0 excep	otions As	sertions: 51	right, 0 wrong, 0 ignored, 0 exceptions		
Edit					TEST SUN	IMARIES			
Properties	FIT:FIT.FITS	ERVER							
Refactor	18 right, 0 v	vrong, 0 ignore	d, 0 exceptions C	laimsSuite.G	ivenEmployee	WithApproverS	uite.WhenCreateClaimSuite.ThenDefaultsOkTest		
Where Used	17 right, 0 w 8 right, 0 wr	vrong, 0 ignore ong. 0 ignored	d, 0 exceptions <u>C</u> . 0 exceptions En	<u>laimsSuite.G</u> nploveesSuite	<u>ivenNewlyCre</u> GivenEmplo	eatedClaimSuite veesSuite.Wher	<u>.WhenModifyAndSaveClaimSuite.ThenSavedOkTest</u>		
Search	8 right, 0 wr	ong, 0 ignored	, 0 exceptions En	nployeesSuite	e.GivenEmplo	yeesSuite.Wher	ListAllSuite.ThenCertainReturnedTest		
Files					TEST O	UTPUT			
Versions				TEST S	YSTEM: FIT	FIT.FITSERV	'ER		
Pasant Ohennes	Claims	Suite.Give	nEmployeeV	VithAppr	overSuite	.WhenCrea	ateClaimSuite.ThenDefaultsOkTest		
Recent Changes	-						<u>Top</u>		
User Guide	Set Up:	ClaimsAppSuite	<u>SetUp (edit)</u>				Expand All Collapse All		
Test History	Included p	oage: <u><givenen< u=""></givenen<></u>	nployeeWithAppro	overSuite.Th	<u>eGiven (edit</u>)	Expand All Collapse All		
	Included p	oage: <u><whencr< u=""></whencr<></u>	eateClaimSuite.T	<u>heWhen (edi</u>	<u>t)</u>		Expand All Collapse All		
	then								
	using naked objects viewer								
	on object	alias result as	perform	on member	that it	value			
	tomsClaim1		check object		is not saved				
	tomsClaim1		check property	Date	contains	02-Mar-2007			
localhost:9090/ClaimsAppSuit	te, ClaimsSuite, GivenE	mployeeWithApprover	rSuite,W	Description	ic amoto				

This makes it easy for non-technical business users to both author new tests, and to view their execution. It also creates an efficient feedback loop; a FitNesse test will "keep on going" even if it hits a failure. Thus the developer can identify several issues and fix them in a single pass.

Another way to think of FitNesse is as a replacement presentation layer, hitting the underlying domain model in the same way that the regular UI would.

If using FitNesse on a "regular" project then the developer writes glue code that in effect take the values out of the wiki page, and use them to interact with the system. They then return a success/failure response which FitNesse then uses to annotate the results page. FitNesse calls this glue code a "fixture". There is some overlap with Naked Objects' own use of that term. However, whereas a Naked Objects fixture is only used to setup the initial state of a test, a FitNesse fixture not only does that but it also is used to execute the test proper.

It is relatively straightforward to integrate FitNesse into a continuous integration environment; see Section 3.9, "Set up Continuous Integration" for further details.

1.3. How Tested Objects' Integration Works

Although you could test a Naked Objects application using vanilla FitNesse, this would entail you having to write all the FitNesse fixtures to interact with the domain objects. You would also need to come up with a representation for the tables.

Tested Objects' FitNesse integration is designed to let you use FitNesse to test your Naked Objects application without all this hassle. We can use the same Naked Objects metamodel that is used for the auto-generation of its generic OOUIs to create a set of generic fixtures that interact with the domain object similarly. Using Tested Objects is therefore just a matter of using these fixtures out-of-the-box.

There are many such fixtures, but the one you will use the most often is the UsingNakedObjectsViewer fixture (see the section called "UsingNakedObjectsViewer / UsingNakedObjectsViewerForSetup"). The screenshot below shows it in use for checking assertions at the end of a test (the "Then").

ClaimsAppSuite.Claim	sSuite.GivenAppro	overSuite.WhenCre	ateClaimSuite.ThenDe	faults0kTest - M	ozilla Firefox				_ O X
()) C ×	http:	//localhost:9090/Claim	sAppSuite.ClaimsSuite.Give	enApproverSuite.W	henCreateClaimSuit	e.ThenDefaultsOkTest	☆ •	Soogle	\sim
ClaimsAppSuite.Clai	msSuite.GivenAp.	. +							-
ClaimsAppSuite. <u>ClaimsSuite</u> . <u>GivenApproverSuite</u> . <u>WhenCreateClaimSuite</u> . ThenDefaultsOkTest [add child]									
Test									,
Edit	Set Up: .C	laimsAppSuite	.SetUp <u>(edit)</u>					Expand All	Collapse All
Properties	Included p	age: <u><givenap< u=""></givenap<></u>	proverSuite.The	<u>Given (edit)</u>				Expand All	<u>Collapse All</u>
Refactor	 Included p 	age: <u><whencr< u=""></whencr<></u>	eateClaimSuite. I	<u>heWhen</u> <u>(edi</u>	<u>t)</u>			Expand All	<u>Collapse All</u>
Where Used	then								
Search	using naked	objects viewe	r						
Files	on object	alias result as	perform	on member	that it	value			
Versions	tomsClaim1		check object		is not saved				
Recent Changes			check property	Date	contains	02-Mar-2007			
				Description	is empty				
User Guide				Status	contains	New			
Test History				Claimant		tomEmployee			
				Approver		tomsApprover			
			check collection	Items	is empty				
	► Tear Down: <u>.ClaimsAppSuite.TearDown (edit)</u> Expand All Collapse All								
http://localhost:9090/Claims/	AppSuite.ClaimsSuite	.GivenApproverSuite.	WhenCreateClaimSuite.The	enDefaultsOkTest?t	est				* E

To explain this: the "tomsClaim1" is an alias to an object that has been created or obtained previously. The "perform" column lists the verb to do, in this case all checks. Other things that can be done include invoke actions, set properties and so forth. The "on member" column specifies the property, collection or action of the domain object being interacted with.

The following screenshot shows how the test is annotated after a successful run:

🥙 Test Results: ClaimsAppSuite.ClaimsSuite.GivenApproverSuite.WhenCreateClaimSuite.ThenDefaultsOkTest - Mozilla Firefox – 🗆 🗙										
Elle Edit View History										
	C X M I http://ocahost:9090/ClaimsAppSuite.ClaimsSuite.GivenApproverSuite.WhenCreateClaimSuite.ThenDefaultsOkTest?test X · S · Google									
Test Results: Claims	sAppSuite.Claims							~		
(they	<u>ClaimsAppSui</u>	laimsAppSuite. <u>ClaimsSuite</u> . <u>GivenApproverSuite</u> .								
	TEST RESULT	S [history]						Output		
Test	Assertions	s: 18 right, 0	wrong, 0 ignor	ed, 0 exce	ptions			<u>Captured</u>		
Edit										
Dreparties	Set Up:	laimsAppSuite	.SetUp (edit)				<u>Expand All</u>	Collapse All		
Properties	Included p	age: <u><givenap< u=""></givenap<></u>	proverSuite.The	<u>Given (edit)</u>			Expand All	Collapse All		
Refactor	Included p	age: <u><whencr< u=""></whencr<></u>	eateClaimSuite.T	heWhen <u>(edi</u>	<u>t)</u>		Expand All	Collapse All		
Where Used										
Search	then									
Files	using naked	objects viewe	r							
Versions	on object	alias result as	perform	on member	that it	value				
Decent Changes	tomsClaim1		check object		is not saved					
Recent Changes	tomsClaim1		check property	Date	contains	02-Mar-2007				
User Guide	tomsClaim1		check property	Description	is empty					
Test History	tomsClaim1		check property	Status	contains	New				
	tomsClaim1		check property	Claimant	contains	tomEmployee				
	tomsClaim1		check property	Approver	contains	tomsApprover				
	tomsClaim1		check collection	Items	is empty					
	Tear Down	n: <u>.ClaimsAppS</u>	<u>uite.TearDown (e</u>	<u>dit)</u>			Expand All	<u>Collapse All</u>		
Done								* E		

The end-user should be able to follow this test and could, if they want, run through the same set of steps manually themselves. Or, of course, they could just look at the test results.

Chapter 2 Using the FitNesse Archetype

Like Naked Objects, Tested Objects provides a Maven archetype to get you up and running with quickly. Tested Objects' FitNesse archetype runs against any Naked Objects application, and provides a new xxx-fitnesse project (where xxx is the root artifact Id). This project bundles FitNesse, contains the FitNesse wiki pages that contain the tests (and also a user guide), and allows the FitNesse wiki server to be run.

Out-of-the-box the archetype will generate a set of tests for the example claims application in the nakedobjects-4.0.x-for-maven.zip download¹, under examples/claims). These are great to try out *Tested Objects* for the first time, and they also provide some ideas for you to structure your own test suite.

The xxx-fitnesse project generated by the archetype is intended to be included as a module of the parent pom (xxx/pom.xml) --- in Maven parlance, it is a partial archetype. Providing that your run the archetype in the correct directory, Maven will automatically include the newly generated project in the parent project.

2.1. Prerequisites

To use the archetype you'll need to install <u>Maven</u> (and there's a good chance you've done this already if you're using the Naked Objects Maven archetype).

Optionally (and recommended) you can install the <u>m2eclipse</u> plugin for <u>Eclipse</u> IDE (again, there's a good chance you've done this too).

2.2. Running the Archetype

You can either run the archetype from the command line (and then import into Eclipse if using m2eclipse), or run the archetype using m2eclipse straight off.

¹See <u>http://sourceforge.net/projects/nakedobjects</u>

Running the archetype from the command line

Navigate to the parent directory of the project (that holds the parent pom.xml). The directions here are for the examples/claims project.

TODO: Naked Objects 4.0.0 released examples/claims as org.nakedobjects.distribution:examples-claims. In 4.0.1 (the plan is) for it to be released as org.nakedobjects.examples:claims. In the screenshots that follow I've hacked the 4.0.0 pom to change the groupId and artifactId. However, I've provided the command that kicks the whole thing for both 4.0.0 and 4.0.1

To run the archetype from the command line (against NOF 4.0.0):

```
$ cd $NO_HOME/examples-claims
$ mvn archetype:generate \
    -D archetypeCatalog=http://starobjects.sourceforge.net/m2-repo/snapshot \
    -D archetypeGroupId=org.starobjects.tested.fitnesse \
    -D archetypeArtifactId=archetype \
    -D archetypeVersion=1.0-beta-3-SNAPSHOT \
```

- -D groupId=org.nakedobjects.distribution \
- -D artifactId=claims-fitnesse \
- -D parentArtifactId=claims \
- -D package=org.nakedobjects.examples.claims.fitnesse \
- -D version=4.0.0

or against NOF 4.0.1:

```
$ cd $NO_HOME/examples-claims
```

- \$ mvn archetype:generate \
 - -D archetypeCatalog=http://starobjects.sourceforge.net/m2-repo/snapshot \
 - -D archetypeGroupId=org.starobjects.tested.fitnesse \
 - -D archetypeArtifactId=archetype \
 - -D archetypeVersion=1.0-beta-3-SNAPSHOT \
 - -D groupId=org.nakedobjects.examples \
 - -D artifactId=claims-fitnesse \

```
-D parentArtifactId=claims \
```

- -D package=org.nakedobjects.examples.claims.fitnesse \
- -D version=4.0.1

The groupId:parentArtifactId:version tuple should correspond to the parent pom (so, for 4.0.0, that is org.nakedobjects.distribution:claims:4.0.0, and for 4.0.1 it is org.nakedobjects.examples:claims:4.0.1). Note that the parentArtifactId is not one of the standard Maven properties, instead it is an additional property mandates by the Tested Objects' archetype. (The archetype uses this property to derive the names of some the other projects to include in the dependencies).

The artifactId then is the artifact of the FitNesse project being generated, so is typically *parentname*-fitnesse (so here: claims-fitnesse).

The output is shown below:



This will generate a claims-fitnesse directory, and update the parent pom.xml file.

We can then import into project into Eclipse, using File > Import > Maven

Import	
Select Maven Projects	Ľ
Select an import source:	
Pyper Here HereAl Image: Archive File Image: Archive File	
(? < Back Next > Einish	Cancel

⇒ Import Maven projects	
Maven Projects Select Maven projects	
Root Directory: Stilava/nakedobjects-4.0.0/examples/dams	Browse
Projects: fpom.xml - org.nakedobjects.examples:daims:4.0.0;pom dom/pom.xml - org.nakedobjects.examples:daims-dom:4.0.0;jar fxture/pom.xml - org.nakedobjects.examples:daims-dom:4.0.0;jar commandine/pom.xml - org.nakedobjects.examples:daims-dom:4.0.0;jar webapp/pom.xml - org.nakedobjects.examples:daims-daims-dom:4.0.0;jar webapp/pom.xml - org.nakedobjects.examples:daims-daims-dom:4.0.0;jar webapp/pom.xml - org.nakedobjects.examples:daims-daims-dom:4.0.0;jar webapp/pom.xml - org.nakedobjects.examples:daims-daims-daims-fitnesse:4.0.0;jar daims-fitnesse/pom.xml - org.nakedobjects.examples:daims-fitnesse:4.0.0;jar	Select <u>A</u> II <u>D</u> eselect AII <u>R</u> efresh
Add project(s) to working set	New
Adyanced	
	Cancel

Running the archetype from m2eclipse

As an alternative to running the archetype from the command line, you can instead run the archetype using m2eclipse.

However, m2eclipse does not (seem to) understand partial archetypes, and so the project cannot be generated in a subfolder of the parent project. Instead, I recommend you create the project alongside the parent (eg in examples/claims-fitnesse).

TODO: These notes are slightly incomplete; it is also necessary to specify the archetype catalog in the Windows>Preferences. However, m2eclipse 0.9.8 (the latest 'stable' version at the time of writing) fails to pick up snapshot archetypes from remote catalogs. The workaround is either to use the command line once (this will copy the archetype into your local repository) or alternatively achieve the same thing by building the FitNesse code from scratch, as described in the developers' guide.

So, use File > New > Project, then Maven > Maven Project to bring up the wizard:



Specify the directory. Note that this cannot be a directory that contains a pom.xml file.

🗢 New Maven Project	
New Maven project	
Enter a location for the project.	IM
Create a simple project (skip archetype selection)	
Use default Workspace location	
Location:	Browse
Add project(s) to working set	
Working set:	<u>∨</u> <u>N</u> ew
► Ad <u>v</u> anced	
Comparison (Comparison (C	Cancel

Select the archetype. (Note that the screenshot here shows the archetype in the "Default Local" catalog ... that's because I've built the archetype from source.).

🗢 New I	Maven Project		
New Ma Select ar	ven project Archetype		M
C <u>a</u> talog: Filter:	Default Local		Con <u>fig</u> ure
Group I	ld robjects.tested	Artifact Id fitnesse-integration	Version 1.0-beta-3-SNAPSHOT
Tested (✓ <u>S</u> how ► Ad <u>v</u> an	Dbjects Fitnesse Inte the last version of A Iced	egration (Archetype) rchetype only	ین Add Archetype
?		< Back Next >	Einish Cancel

Specify the groupId, artifactId (eg claims-fitnesse), version and package. Also specify the parentArtifactId as the parent project's artifactId (eg claims):

🗢 New Ma	iven Project								
New Mave Specify Ard	en project hetype paramet	ers			M				
Group Id:	org.nakedobje	org.nakedobjects.examples							
Artifact Id:	claims-fitnesse								
Version:	4.0.0								
Package:	org.nakedobje	cts.examples.claims.fitne	sse						
Properties:									
Name		Value			<u>A</u> dd				
testedObj rootArtifad	ectsFitNesse :tId	1.0-beta-3-SNAPSHOT claims			Remove				
Advance	d								
	-								
?		< <u>B</u> ack	<u>N</u> ext >	Einish	Cancel				

Hit Finish and you will be in more-or-less the same place as if you had generated the archetype from the command line and then imported. The only thing different is that the new fitnesse project will not be in the parent project's modules.

Therefore, navigate to the parent project's <modules> region, and add:

```
<modules>
<module>dom</module>
<module>fixture</module>
<module>service</module>
<module>commandline</module>
<module>webapp</module>
<module>.../claims-fitnesse</module>
</modules>
```

The instructions in the wiki do assume that you have done this.

2.3. Starting FitNesse Wiki

Once the fitnesse project has been imported you can start the wiki. Navigate to ide/eclipse/launch and there will be a launch configuration:



Start by right clicking and selecting run as. The FitNesse wiki should start on port 9090:



2.4. Setting up to Run Tests

Using your favorite browser, navigate to <u>http://localhost:9090/FrontPage</u>; you should see the introductory page:



Before we can run the generated tests there are a couple of things we need to take care of (both are documented on FrontPage). First, we need to copy the application into a location so that FitNesse can find it. This is done by simply running mvn clean install on the project:



(Note, if for any reason you didn't include the xxx-fitnesse project into the parent project, you'll need to run mvn clean install for the xxx-fitnesse project as well).

Secondly, the pages generated by the archetype use the nakedobjects.properties to bootstrap Naked Objects. By default, this is assumed to be in .../commandline/config/nakedobjects.properties. If this isn't the case, then edit the BootStrapNakedObjects page as required:



2.5. Running the Tests

If you are trying out the archetype against the examples/claims application then you'll probably want to run the existing tests to check everything works.

The tests are organized into a single application suite, ClaimsAppSuite. This in turn defines two subsuites:



FitNesse will automatically invoke all tests in the hierarchy, therefore just press the Suite button on the left:

Suite Results: Claims	AppSuite × 🕀									
(← → C ☆	http://localhost:9	090/ClaimsAppSui	te?suite						D •	۶.
(All and a second seco) ClaimsAppSuite Suite Results Inistory]									
Suite	Test Page	s: 4 right, 0	wrong, 0 ignore	ed, 0 excep	otions As	sertions: 51	right, 0 wrong, 0 ignored, 0 exce	eptions	4	
Edit					TEST SUN	IMARIES				
Properties	FIT:FIT.FITS	ERVER								
Refactor	18 right, 0 v	vrong, 0 ignore	d, 0 exceptions C	laimsSuite.G	ivenEmployee	WithApproverS	uite.WhenCreateClaimSuite.ThenDefa	<u>ultsOkTest</u>		
Where Used	17 right, 0 w 8 right, 0 wr	vrong, 0 ignored	d, 0 exceptions <u>C</u> , 0 exceptions En	laimsSuite.G	ivenNewlyCre .GivenEmplo	atedClaimSuite	.WhenModifyAndSaveClaimSuite.Then ListAllSuite.ThenAllReturnedTest	SavedOkTes	<u>at</u>	
Search	8 right, 0 wr	ong, 0 ignored	, 0 exceptions En	nployeesSuite	.GivenEmplo	yeesSuite.Wher	ListAllSuite.ThenCertainReturnedTest			
Files					TEST O	JTPUT				
Versions				TEST S	YSTEM: FIT	:FIT.FITSERV	/ER			
Recent Changes	Claims	Suite.Give	nEmployeeV	VithAppr	overSuite	.WhenCrea	ateClaimSuite.ThenDefau	lts0kTe	st	
User Guide	Set Up: .0	laimsAppSuite	SetUp (edit)				Expand All	Collapse	Top All	
Test History	Included p	age: <u><givenen< u=""></givenen<></u>	nploveeWithAppro	overSuite.Th	eGiven (edit		Expand All	Collapse	All	
Test history	Included p	age: <u><whencr< u=""></whencr<></u>	eateClaimSuite.T	heWhen (edi	<u>t)</u>		Expand All	<u>Collapse</u>	All	
	then									
	using naked objects viewer									
	on object	alias result as	perform	on member	that it	value				
	tomsClaim1		check object		is not saved					
	tomsClaim1		check property	Date	contains	02-Mar-2007				

If you've run the archetype against your own project though, then you can either delete these ClaimsAppSuites, or leave them around for reference. To ensure that they don't accidentally run, use the Properties button (on the left hand side) and deselect the Suite and Test properties.

Chapter 3 Hints and Tips

This chapter contains a collection of hints, tips and suggestions for writing your own tests.

For further guidance, check out Gojko Adzic's book, Bridging the Communication Gap.

3.1. Separate In-Progress Stories from the Backlog

If you are using an agile methodology then you will be implementing a number of stories per iteration; the remainder will be in a backlog. When you select a story for implementation, create a new page for it in a "CurrentIteration" suite. The objective for the team is therefore to get the entire CurrentIteration suite green.

Other stories that you may have identified but not selected for the iteration can remain in a Backlog suite.

3.2. Use a Story Page to Collect A Set of Scenario Tests

Part of estimating the size of a story includes identifying the acceptance criteria. These can be created as children of the story page as placeholders, so that the story page becomes a suite. The child scenario tests can be fleshed out as required with plain text during the estimation meeting, and with actual FitNesse tests once the iteration starts. The FitNesse <u>!contents</u> instruction will then list all the acceptance criteria for the story.

For the story page itself, the "as a ... I want ... so that... " template is a good way to summarize the intent of the story.

3.3. Organize Completed Stories by Component

Once you have completed an iteration and implements its stories, move those stories out to the relevant component that the story relates to. The scenario tests for stories ultimately *are* the documentation of the

behaviour of the system. A year on you won't remember (and won't care) which iteration you implemented a story, you'll be searching for it by the component whose behaviour you want to understand.

3.4. Structure your test using Given/When/Then

A standard template for organizing structuring tests is given/when/then¹:

- given ... the system is in this particular state
- when ... this interesting thing happens
- then ... these are the consequences

This structure is readily understood by non-technical business users, and helps them (and the team) focus on the point of the test.

In terms of mechanics, one approach is to put the "given" into the setup page for a test, with the "when" and the "then" in separate pages. Alternatively, as the archetype does (see Chapter 2, *Using the FitNesse Archetype*), you could separate out the "given", the "when" and the "then" into a hierarchy of pages.

- Separating out the "given" from the rest of the test makes it easy to include that given in other tests (discussed further in Section 3.6, "Factor out common "Given"s"), and it also allows us to run up the viewer to inspect the setup (see Section 3.5, "Using the RunViewer fixture").
- Separating out the "then" from the "when" makes it easy to identify the individual post conditions. A new requirement might mean only the addition of a new post condition. A downside is that the "given" and "when" will be run for each post-condition, leading to longer test turn-around times.

3.5. Using the RunViewer fixture

One reason that the archetype (Chapter 2, *Using the FitNesse Archetype*) separates out the "given", "when" and "then" is so that the "given" - which is often the hardest part to get setup - can be verified independently from the rest of the test.

To do this, we can use the RunViewer fixture (see the section called "RunViewer"). This will run up the drag-n-drop viewer at the specified point in the test; a visual equivalent of System.out.println(), really. We can therefore take the Given page and add a RunViewer fixture at the end.

Note that to do this you must temporarily mark the Given page as a test page.

3.6. Factor out common "Given"s

Just like code, tests need to be actively managed, because if the tests become hard to maintain, they'll end up being deleted. In fact, we probably should take even more care with the tests than the code if they represent the primary documentation of the behaviour of the system.

In terms of size, the "given" is far larger than either the "when" or the "then", and therefore this is the area where tests can quickly become unmaintainable. So instead, factor out your givens into separate pages,

¹As first described, I believe, by Dan North in a blog post, <u>Introducing BDD</u>.

and then use FitNesse's <u>linclude</u> directive to assemble the pages you need (as done by the archetype, Chapter 2, *Using the FitNesse Archetype*).

The names of these pages should also follow a declarative style, see Section 3.7, "Use a Declarative Style for Page Names".

3.7. Use a Declarative Style for Page Names

When factoring out "given"s (see Section 3.6, "Factor out common "Given"s"), or indeed when writing the "when"s and the "then"s, use a declarative style for the pages. The page should describe what it does, not how it does it.

For example, a good page would be "SetUpCountries". It's clear that this will set up all Country reference data classes. This could be included into a "SetUpReferenceData" page. For transaction data, we could have a page "JoeBloggsCustomer"; another one again could be "JoeBloggsFiveOrders".

3.8. Run against a real database

The Tested Objects' integration exercises the Naked Objects domain model as configured in nakedobjects.properties (as per the SetUpConfigDirectory fixture, ???). Out-of-the-box, of course, Naked Objects uses an in-memory object store, and so there are no issues running one scenario test against another.

You can if you want though configure Naked Objects to go against a real database, for example by using the Hibernate-based object store provided by the <u>JPA Objects</u> sister project. In this case you will need to ensure that you reset the database at the end; Tested Objects doesn't provide any fixtures to help you though, so you are on your own here. An alternative might be to use JPA Objects against HSQLDB configured to for in-memory use. This would let you verify your database mappings, but without no need to tear anything down.

3.9. Set up Continuous Integration

Since Tested Objects is a Maven application, it is easy enough to configure it to run under a CI server, such as <u>Hudson</u>. If you google around you should be able find <u>a way</u> to publish the FitNesse test results through Hudson.

Appendix A. Reference (Online User Guide)

For convenience when writing tests the FitNesse wiki pages created by the Maven archetype (see Chapter 2, *Using the FitNesse Archetype*) also include an online user guide:

Coogle Coogle X										
< → C ☆	http://localho	st:9090/FitNesse.UserGuide	► □ - <i>₽</i> -							
(Contraction of the second se	FitNesse. UserGuide									
Edit Properties Refactor	FORMATTIN See <u>WikiForma</u>	FORMATTING See <u>WikiFormatting</u> for examples of supported wiki markup.								
Where Used	FIXTURE T	ABLES								
Search	Туре	Table Name	Purpose							
Files	Bootstrapping	Used for bootstrapping the is included in the test suite	test framework itself. These are typically referenced in <u>.BootstrapNakedObjects</u> which 's setup page.							
Versions		StoryFixture	Sets up the workflow story test. Should appear first.							
User Guide		SetConfigDirectory	Specifies the config directory containing nakedobjects.properties							
		EnableExploration	Enables exploration actions if required							
		InitNakedObjects	Use to initialize Naked Objects runtime.							
	Setup	Used to initialize a particula	ar story's setup. Typically referenced in the test suite or story's own setup page							
		<u>Datels</u>	Sets the clock to a specific date and time							
		LogonAs or SwitchUser	Logs on as a specific user							
		<u>AliasServices</u>	Aliases services so that actions can be invoked upon them							
		<u>SetUpObjects</u>	Initializes objects. Can use for either reference data or operational data							
	User interaction	Appear in the main body of	the test							
		<u>UsingNakedObjectsViewer</u>	Simulates interacting with the domain objects as if through a viewer. Also use to assert on objects' state, and to alias objects							
		<u>CheckList</u>	Check items in list, either precisely or just for presence							
		<u>AliasItemsInList</u>	Provide an alias to items in list, which are presumed to exist (see also <u>CheckList</u>)							
	Debugging	Debugging and diagnostics.	Useful for checking setup is correct, for example							
		<u>DebugServices</u>	Lists service class names, as picked up from configuration. Useful with <u>AliasServices</u>							
		<u>DebugClock</u>	Reads current value of the clock							
		DebugObjectStore Dumps contents of the object store								
		CheckSpecificationsLoaded	Verifies that listed NakedObjectSpecifications have been loaded into the metamodel							

Rather than repeat the text here, this reference guide just consists of screenshotsof the various pages.

A.1. Bootstrapping

The bootstrapping fixtures are used to bootstrap the test framework itself. These are typically referenced in a "BootstrapNakedObjects" page, included in the test's setup page. One option is to use the FitNesse <u>SetUp</u> page.

StoryFixture

Sets up the workflow story test. Boilerplate, should always be the first FitNesse fixture included in a page.



SetConfigDirectory

Specifies the config directory containing nakedobjects.properties. Called after StoryFixture (see the section called "StoryFixture"), and before InitNakedObjects (see the section called "InitNakedObjects"). The DebugServices fixture (the section called "DebugServices") can be used to debug the set of services specified (after Naked Objects has been initialized).



EnableExploration

Enables exploration actions if required. Should be called before InitNakedObjects (see the section called "InitNakedObjects").

Note that when using the DnD or HTML viewers, exploration mode means that there is no need to logon. For FitNesse tests though you should specify who to login as, see the section called "LogonAs".

FitNesse.Enable	Exploration × 🗭
€ → C 2	http://localhost:9090/FitNesse.EnableExploration
(Contraction of the second se	EinableExploration
Edit	Description
Properties	Enables exploration actions if required. Part of the bootstranning of the test framework itself, buically referenced
Refactor	in <u>BootstrapNakedObjects</u> , prior to calling <u>InitNakedObjects</u> .
Where Used	Anciumnite
Search	ARGUMEN I S
Files	• none
Versions	COLUMNS
Recent Changes	• none
User Guide	
	Example Usage
	enable exploration

InitNakedObjects

Initializes the Naked Objects runtime using the services specified through the UseConfigDirectory fixture (see the section called "SetConfigDirectory").

The CheckSpecificationsLoaded fixture (the section called "CheckSpecificationsLoaded") can be used to check which classes have been located from the services as a result of initialization.

FitNesse.InitNak	edObjects × +
€ → C ₹	☆ http://localhost:9090/FitNesse.InitNakedObjects
(Contraction of the second se	EitNesse. InitNakedObjects
Edit Properties	DESCRIPTION Use to initialize Naked Objects runtime. Part of the bootstrapping of the test framework itself, typically referenced in .BootstrapNakedObjects included in the test suite's setup page.
Where Used Search	
Files Versions	COLUMNS
User Guide	• none
	EXAMPLE USAGE

The setup fixtures are used to specify the running application for a particular story's setup. Specifically, this means setting up the services that define the application, the effective date and the effective user. It also allows the setup of arbitrary objects (typically reference/static data objects; for transactional objects see the section called "UsingNakedObjectsViewer / UsingNakedObjectsViewerForSetup").

Datels

Sets the clock to a specific date and time. This installs the FixtureClock as the implementation of the Clock singleton (in the applib). If this fixture is not called, then the default system clock is used, which gets the time from the host computer. The DebugClock fixture (the section called "DebugClock") can be used to verify the clock state.



LogonAs

Logs on as a specific user.

Unlike Naked Objects' own LogonFixture, the login specified is not remembered to the end of the setup. In order to run tests as a particular login it should therefore appear towards the end of the setup.

FitNesse.Logon/	As × +
€ → C s	http://localhost:9090/FitNesse.LogonAs
	Fittesse. LogonAs
Edit	DESCRIPTION
Refactor	Logs on as a specific user. Part of the initialization for a particular story's setup, and typically referenced in the test suite or story's own setup page. Unlike Naked Objects' own LogonFixture, the login specified is <i>not</i> remembered to the end of the
Where Used	setup. In order to run tests as a particular login it should therefore appear towards the end of the setup.
Search	Variants
Files	SwitchUser o is effectively a synonym.
Recent Changes	Arguments
User Guide	 login name as returned DomainObjectContainer#getUser() role name(s) can specify up to four.
	COLUMNS
	• none
	Example Usage
	logon as fsmith
	with roles
	logon as fsmith with roles rpt_mgr_role,admin_role

AliasServices

Specifies an alias to services in order to invoke actions upon them. Note that the services are *not* defined by this fixture; for that see SetConfigDirectory fixture, section the section called "SetConfigDirectory". See also DebugServices (the section called "DebugServices") to verify the services that have been identified.

FitNesse.AliasSer	rvices ×	Google 👝 🔲 🗶
€ → C ₹	http://localhost:9090/FitNesse.AliasServices	► 🗗 🖈
	FitNesse. AliasServices	
Edit Properties Refactor	DESCRIPTION Specifies an alias to services in order to invoke actions upon them. No fixture, instead they are read from the config directory.	ote that the services are <i>not</i> defined by this
Where Used Search Files	ARGUMENTS • none	
Versions Recent Changes User Guide	COLUMNS • class name • the fully qualified name of the service implementation • alias= • assigns an alias to the service instance • can also use 'alias as'	
	EXAMPLE USAGE	
	alias services	
	class name	alias=
	com.mycompany.myapp.service.claim.ClaimRepositoryInMemory	claims
	com.mycompany.myapp.service.claim.EmployeeRepositoryInMemory	daims

SetUpObjects

Initializes objects. Typically used for immutable reference/standing data objects). Can also be to setup used for transaction/operational data objects (though UsingNakedObjectsViewerForSetup, the section called "UsingNakedObjectsViewer / UsingNakedObjectsViewerForSetup", is preferable). The DebugObjectStore fixture (the section called "DebugObjectStore") can be used to check the state of objects created.

FitNesse.SetUpO	bjects ×	₽\			(Google 👝 💷	<u> </u>			
< → C ☆	http://localho	st:9090,	/FitNesse.SetUp	Objects		▶ ⊡-	£-			
	FitNesse. SetUpObjects									
Edit Properties	DESCRIPTION									
Refactor	Initializes objec story's setup. Ty	Initializes objects. Can use for either reference data or operational data. Used to initialize a particular story's setup. Typically referenced in the test suite or story's own setup page.								
Where Used	VARIATIONS	Variations								
Files	 <u>SetUpObject</u>, the 'transie occasionally 	<u>SetUpOl</u> nť varia / useful f	bjects, <u>SetUpTra</u> tions do not pers for complex setu	nsientObject, SetUpTransie sist the instantiated object p where there is no action	entObjects that can be invoked					
Versions Recent Changes	∘ can be pers	isted lat	er through its al	ias						
User Guide	ARGUMENTS	dass pa	ma of object to	instantiate						
	• Tutty quatried	CldSS 11d	ine or object to	linstantiate						
	COLUMNS • property names of the domain object o not all properties need to be specified • alias= o assigns an alias to each instance (saves having to lookup the object in the test proper) o can be left blank, aliases will be automatically assigned o if no alias column is provided, one will be appended and aliases automatically assigned o can also use 'alias as'									
	EXAMPLE US	AGE								
	set up objects	com.my	company.foobar	.dom.employee.Employee						
	Name	Approve	er	alias as						
	Fred Smith			Employee:Fred Smith						
	Tom Brown	Employe	e:Fred Smith	Employee: Tom Brown						
	Sam Jones	Employe	e:Fred Smith	Employee:Sam Jones						
				. fachar dan anala						
	set up transien	t object	com.mycompar	ny.roopar.dom.employee.En	npioyee					
	Rill Jackson		Approver	allas as						
	Did Jackson			спірюуее: віц заскзоп						
۲]				III						

A.3. User Interaction

User interaction fixtures appear in the main body of the test, for the "given" (to setup the rest of the state of the system, typically transactional objects), for the "when" (the interaction being tested) or the "then" (assertions on the state after the interaction being tested).

UsingNakedObjectsViewer / UsingNakedObjectsViewerForSetup

Simulates interacting with domain objects as if through a viewer. Interact with objects, check their state, alias referenced or returned objects.

The "ForSetup" version disables checks for visibility and usability, making it easier to reuse functionality for setting up objects prior to a test scenario (the "given"). The DebugObjectStore fixture (the section called "DebugObjectStore") can be used to check the state of objects created.



Example usage:

FitNesse.Using	lakedObjec ×	Ð							
€ → C	http://loca	alhost:9090/I	itNesse.Using	gNakedObject	:sView				
(Carles	Example	Example Usage							
Edit	using nake	using naked objects viewer							
Eun	on object	alias result a	s perform	using me	mber				
Properties	employees	list1	invoke act	ion All Emplo	/ees				
Refactor									
Where Used	using nake	d objects viev	ver for set up						
Search	on object	alias result a	s perform	using me	nber				
Files	employees	list1	invoke act	ion All Emplo	/ees				
Versions Recent Changes	Alternate co	lumn headers	(more forma	l style):	1				
recent onaliges	using nake	d objects view	ver						
User Guide	object	result= inte	raction type	member					
	employees	employees list1 invoke action All Employees							

On properties:

			gnaledobjectonene			
(Contraction)		ES				
Edit Properties	on object	alias as	perform	using member	that it	value
Refactor	object alias		check property	property name	is hidden	1
Where Used	object alias		check property	property name	is visible	
Search Files	object alias		check property	property name	is disabled	
Versions	object alias		check property	property name	is enabled	
Recent Changes	object alias		check property	property name	is empty	
User Guide	object alias		check property	property name	is not empty	
	object alias		check property	property name	contains	value or object alias
	object alias		check property	property name	does not contain	value or object alias
	object alias		check set property	property name	is valid for	value or object alias
	object alias		check set property	property name	is not valid for	value or object alias
	object alias		check clear property	property name	is valid	
	object alias		check clear property	property name	is not valid	
	object alias	alias for default object	get property default	property name		
	object alias	alias for list of choices	get property choices	property name		
	object alias	alias for referenced object	get property	property name		
	object alias	alias for referenced object	set property	property name		value or object alias
	object alias		clear property	property name		

On collections:

FitNesse.UsingNa	akedObjec ×	(†			Goo	gle 👝 🔲	X			
< → C <	► → C ☆ http://localhost:9090/FitNesse.UsingNakedObjectsViewer									
(Sallega)	ON COLLECTIONS									
	on object	alias as	perform	using member	that it	reference				
Edit	object alias		check collection	collection name	is hidden					
Properties	object alias		check collection	collection name	is visible					
Flopenies	object alias		check collection	collection name	is disabled					
Refactor	object alias		check collection	collection name	is enabled					
Where Used	object alias		check collection	collection name	is empty					
Search	object alias		check collection	collection name	is not empty					
	object alias		check collection	collection name	contains	object alias	1			
Files	object alias		check collection	collection name	does not contain	object alias	1			
Versions	object alias		check add to collection	collection name	is valid for	object alias				
Recent Changes	object alias		check add to collection	collection name	is not valid for	object alias				
User Guide	object alias		check remove from collection	collection name	is valid for	object alias				
	object alias		check remove from collection	collection name	is not valid for	object alias				
	object alias	alias for collection	get collection	collection name			1			
	object alias	alias for collection	add to collection	collection name		object alias	1			
	object alias	alias for collection	remove from collection	collection name		object alias	1			
							- -			

On actions:

< → C ₹	http://loc	alhost:9090/FitNesse.UsingNa	akedObjectsViewer			• •					
(telles	ON ACTIONS	NI ACTIONS									
	on object	alias result as	perform	using member	that it	with arguments					
Edit	object alias		check action	action name	is hidden						
Refactor	object alias		check action	action name	is visible						
Where Used	object alias		check action	action name	is disabled						
Search	object alias		check action	action name	is usable						
/ersions	object alias		check action	action name	is valid for	argument list					
Recent Changes	object alias		check action	action name	is not valid for	argument list					
Jser Guide	object alias	alias for parameter default	get action parameter default	action name		param number (0- based)					
	object alias	alias for list of parameter choices	get action parameter choices	action name		param number (0- based)					
	object alias	alias for returned object	invoke action	action name		argument list					

On objects:

Coogle 👝 🗆 X									
← → C ☆ http://localhost:9090/FitNesse.UsingNakedObjectsViewer									
(and a second	ON OBJECTS T	HEMSELVES							
	on object	perform	that it						
Edit	object alias	check object	is valid						
Properties	object alias	check object	is not valid						
Refactor	object alias	check object	is saved						
Where Lised	object alias	check object	is not saved						
Where Osed	object alias	save object							
Search									
Files									

CheckList

Check items in list, either precisely or just for presence, using their title. Lists are either aliased results of actions, or aliased collections within objects.

Typically used in the "Then", though can be helpful as a way of confirming/documenting a "Given".

See also AliasItemsInList (the section called "AliasItemsInList"), which also performs an implicit check (will fail if the objects are not in the list) and aliases them for further use.

FitNesse.CheckL	ist X
€ → C s	http://ocalhost:9090/FitNesse.CheckList
	FitNesse. CheckList
Edit Properties Refactor Where Used Search Files Versions Recent Changes User Guide	DESCRIPTION Check items in list, either precisely or just for presence. Lists are either aliased results of actions, or aliased collections within objects. See also Aliasitemsini List. ARGUMENTS • list alias • qualifier • either 'contains' or 'precisely contains' COLUMNS • Title • Type • optional, concrete type Example Usage And drek list list is contains Title Fred Smith Tom Brown and drek list list is precisely contains Title Fred Smith Tom Brown Sam Jones

AliasItemsInList

Allows an alias to be associated with items in a list. The list items are located by their title, and are presumed to exist. This fixture can therefore also be used as a way of checking for presence of items in a list (similar to CheckList, the section called "CheckList").

Typically used both in the "Given" (to simplify writing the rest of a test).

FitNesse.AliasIte	msInList × 🜩									
€ → C ₹	http://localhost:	9090/FitNesse.	AliasItemsInList					►	0-	p -
	AliasIte	msInLi	st							
Edit	DESCRIPTION									
Properties Refactor	Provide an alias to items in list, which are presumed to exist. See also <u>CheckList</u> .									
Where Used	Arguments									
Search	• list alias	• list alias								
Files	Country									
Versions	COLUMNS									
Recent Changes	 title type 									
User Guide	 o the required class name (can be fully qualified, but does not need to be) o optional; useful for collections of roles where the title given to each role might be that of the parent, and only the type differs alias as can also use 'alias=' 									
	EXAMPLE USAG	θE								
	alias items in list	list1								
	title	alias as								
	Tom Brown	tomEmployee								

A.4. Debugging

Debugging and diagnostics. Useful for checking setup, for example.

DebugServices

Lists service class names, as picked up from configuration. Useful with AliasServices (see the section called "AliasServices").



DebugClock

Reads the current value of the clock. Useful for debugging and diagnostics.

Coogle 🔲 🗓 X		
← → C ☆ http://localhost:9090/FitNesse.DebugClock		
	Eitnesse. DebugClock	
Edit Properties Refactor	DESCRIPTION Reads current value of the clock. Useful for debugging and diagnostics.	
Where Used	Arguments	
Search	• none	
Files Versions Recent Changes	COLUMNS • none	
User Guide		
	debug clock	

DebugObjectStore

Dumps the contents of the object store. Useful for debugging setup (through SetupObjects, the section called "SetUpObjects", and UsingNakedObjectsViewerForSetup, the section called "UsingNakedObjectsViewer/UsingNakedObjectsViewerForSetup").

Coogle 👝 🖾 🗙		
C ☆ http://ocalhost:9090/FitNesse.DebugObjectStore		► □ - / -
(Contraction of the second se	Fithesse. DebugObjectStore	
Edit Properties Refactor	DESCRIPTION Dumps contents of the object store. Useful for debugging and diagnostics.	
Where Used	Arguments	
Search	none	
Files Versions Recent Changes	COLUMNS • none	
User Guide	EXAMPLE USAGE	

CheckSpecificationsLoaded

Verifies that listed NakedObjectSpecifications have been loaded into the metamodel.

Coogle E X			
← → C ☆ http://localhost:9090/FitNesse.CheckSpecificationsLoaded			
	Eitnesse. CheckSpecificationsLoaded		
Edit Properties Refactor	DESCRIPTION Verifies that listed NakedObjectSpecifications have been loaded into the metamodel. Useful for debugging and diagnostics.		
Where Used	Arguments		
Search	• none		
Files Versions	Columns		
Recent Changes	short name o name of class		
User Guide	Example Usage		
	check specifications loaded		
	short name		
	EmployeeRepositoryInMemory		
	ClaimRepositoryInMemory		
	Claimant		
	ClaimItem		
	Employee		

RunViewer

Runs up the DnD viewer with the current state of the objects. This is a great way to inspect the state of the system, for example if a test is failing and you can't see why.

Of course, other alternatives are writing unit tests and debugging with break points. But failing scenario tests usually arise because of an integration issue between two different bits of the system. Being able to have a "poke around" can be invaluable.

A.5. Tear Down

The opposite of setting up...

ShutDownNakedObjects

This fixture shuts down the Naked Objects runtime, releasing memory and so on. A good place to put this is in the test's <u>TearDown</u> page (which could be inherited).