Round Trip Integration Guide

Version 1.0

Industrial Logic, Inc.
http://industriallogic.com

Introduction

Round-Trip Integration is the process of Continuous Integration of Storytests and production code. It empowers Customers/Subject Matter Experts/QA with the ability to write tests in parallel with the developers, and test the latest snapshot of the system at any given time. It also provides developers with an environment where they get the latest Storytests every time they integrate.

This guide introduces Round-Trip Integration with a specific set of technologies. You would need a “Staging Area” which should:

1. be a linux box
2. have cvs for version control
3. have Fitnesse installed

Customer View

Customers are those members of the Project Community who specify stories for the system. They could be:

1. the entity paying for the project
2. subject matter experts
3. QA

They would write story tests on Fitnesse. The moment a story is converted to a test (by setting its Property), the story is dispatched to the code repository. When developers perform their next integration with the repository, they will receive the Storytests. They would work on it and make it pass. Once they’re done, they will check in their code. The Staging Area has scheduled tasks (read cron-jobs) which run at given intervals. These tasks involve:

1. checking out the latest production code
2. building the system
3. deploying it such that Fit tests will run against them

The interval for scheduling depends largely on the complexity of the system. In the past, we have scheduled these tasks every 1 to 5 minutes.

Developer View

Developers are those members of the Project Community who are involved in implementing the system.

They will integrate early and often, every time their unit tests pass. They will work toward making all the Storytests pass. They will receive Storytests within their own IDE when they integrate with the repository. They own the code but not the Stories – so they will not modify the Storytests by themselves.

Implementation

In order to achieve the aforementioned views, a certain discipline has to be followed by both parties. We outline below the technical details of our implementation – which could be used in your workspace. These details are targeted to the developers in the Project Community.

Package Structure

We like to follow the package structure described below:

Directory	Description
src/java	Contains production source code
src/tests/java	Contains unit tests
src/fixtures/html **OR** stories	Contains Storytests
src/fixtures/java	Contains fixtures that are referred to in the Storytests
results	Will contain the results of Storytests after execution
lib	Contains libraries that your code depends on (we like to check this in to the repository as well)

You are free to modify this structure as per your conventions. Remember to tie this in with the FitTestRunner (which we describe next).

Fit Test Runner

In addition to Fit, you will need the Fit Test Runner (our addition to Fit), which will allow you to run Fit tests within your IDE (using Junit). (You can get it from your resources page)

Once you have the fitTestRunner.jar in your lib (and in your classpath), create a StoryTests class in the root package (com.yourcompany.yourproduct). The code for this class looks like:

```java
import junit.framework.*;

public class StoryTests extends TestSuite {
  static public TestSuite suite() {
    runner.FitRunner.specsDirectory = "src/fixtures/html";
    runner.FitRunner.storiesDirectory = ".";
    TestSuite suite = new TestSuite();
    suite.addTest(runner.FitRunner.suite());
    return suite;
  }
}
```

Modify line 5 to specify your own specs directory. Do not modify line 6 (it tells the runner where to look for the specs directory – usually the current project directory). The results of execution are automatically put in the results directory under the project.

Fitnesse

You should have an installation of Fitnesse which has our modifications. You will find this in your resources page. This version of Fitnesse copies a test to a specified directory. Then, it determines if the test is a new addition, or is an update to an existing test. Accordingly, it calls an add or update script – which sends the addition/modification to the repository.

You will need to provide *fitnesse.properties* in the fitnesse installation directory. Here is a sample of its contents:

```
acceptanceTestsDirectory=/usr/local/projects/myproject/live/cvsmodulename/src/f
ixtures/html
addFileToCvs=/usr/local/projects/myproject/scripts/addFileToCvs.sh
updateFileInCvs=/usr/local/projects/myproject/scripts/updateFileInCvs.sh
```

As you might have noticed, the project is deployed in a directory in a manner that makes it easy to back it up.

/usr/local/projects/myproject – is the project root.

The *live* directory contains the checked out cvs project. The first time, you would have to do the checkout from the repository manually. After that, setup a script to update it every x minutes.

The *scripts* directory contains the following scripts:
1. addFileToCvs.sh
2. updateFileInCvs.sh
3. checkOutMyProject.sh

addFileToCvs.sh

This script adds a new test to the repository. Fitnesse will pass in the name of the file to be added to cvs.

```
#!/bin/csh -v
cd "/usr/local/projects/myproject/live/cvsmodule/src/fixtures/html"
cvs add $1
cvs ci -m "Added ${1}"
```

updateFileInCvs.sh

This script updates a given test in the repository (called by Fitnesse upon modification of a test).

```
cd "/usr/local/projects/myproject/live/cvsmodule/src/fixtures/html"
cvs update $1
cvs ci -m "Modified ${1}"
```

checkOutMyProject.sh

This script checks out the latest snapshot of the project, and builds it. It requires you to have build.xml in your project root directory.

```
#!/bin/csh
cd
/usr/local/projects/myproject/live/cvsmodulename
cvs update -d
setenv ANT_HOME /usr/local/ant
setenv JAVA_HOME /usr/java/j2sdk1.4.2/
set path=( $path $ANT_HOME/bin )
ant
```

As you can see, you may modify the scripts and their locations.

To deploy fitnesse, you'd need both fitnesse.jar and fitnesseExtended.jar.
We've packaged this for you so with the shell script to run it.

Setting up Cron Jobs

Setup a cron job to call checkOutMyProject.sh every x minutes.

You are now all set to use Round Trip Integration.

Note:
1. It is a good idea to run the scripts independently to verify that they run correctly.
2. You would also want to create CurrentStoryTest – this will help you to focus on the story that you are working on. [Make a copy of StoryTest, and change the specsDirectory to src/fixtures/html/currentTest. Drop in stories that you are working on. Remember to take them out when you are done.]
3. As a developer, you must use the Fitnesse interface for editing any Storytest, should you need to. Do not use your editor to make changes and put it back in – your changes will not make it to Fitnesse.