

# **CS427 Group Nibbler - Unit Testing With FRUIT**

**Charlie Meyer**

**Anthony Gelsomini**

**Dave Buhl**

**Pat Guelah**

---

## **CS427 Group Nibbler - Unit Testing With FRUIT**

Charlie Meyer

Anthony Gelsomini

Dave Buhl

Pat Guelah

---

---

# Table of Contents

1. Overview .....	1
Definitions .....	1
References .....	1
2. Architecture and Design .....	2
Overview .....	2
Components .....	3
User Interface .....	3
Test Case Generation .....	4
Build and Launch System .....	5
Driver Generation and Output Parsing .....	5
3. Project Postmortem .....	7
Future Plans .....	7
Team Member Reflections .....	7
Dave Buhl .....	7
Anthony Gelsomini .....	7
Pat Guelah .....	7
Charlie Meyer .....	7
A. Software Installation .....	8
Requirements .....	8
Installation .....	8
Eclipse 3.4 Distribution for Plug-in Developers .....	8
CDT and Photran Sources .....	8
Nibbler Project Sources .....	8
Cygwin .....	9
GFortran .....	10
Setting up Environment Variables (Windows) .....	10
Setting Up Environment Variables (Cygwin) .....	10
Ruby Gems .....	10
FRUIT .....	11
Nibbler Code .....	11
B. Software User Guide .....	16
Fortran Project Creation .....	16
Creating a Test Case .....	17
Running FRUIT Tests .....	19
C. Test Suite Usage .....	21
Manual Test Procedures .....	21
Verifying Empty View .....	21
Running The Sample FRUIT Tests .....	23
Automated Test Procedures .....	24
FruitLaunchManagerTest .....	24

---

## List of Figures

2.1. Nibbler UML Diagram .....	3
2.2. PUnit View .....	4
2.3. PUnit View Class Diagram .....	4
A.1. Cygwin Install Screen .....	9
A.2. New Repository Location .....	12
A.3. New Repository Location Window .....	13
A.4. Checkout from SVN .....	14
A.5. Checkout Projects .....	15
B.1. The Include Paths Required for FRUIT .....	16
B.2. Configuring FRUIT libraries .....	17
B.3. Create a FRUIT test case .....	18
B.4. The PUnit View Displaying Results .....	20
C.1. Select the PUnit View .....	22
C.2. The Empty Punit View .....	23
C.3. PUnit View After Manual Test .....	24

---

# Chapter 1. Overview

FRUIT is an unit testing framework designed for the Fortran Programming Language. FRUIT is based up on the xUnit style of unit testing as typified by JUnit. The purpose of the Unit Testing with FRUIT CS427 Final Project is to integrate to the FRUIT testing framework with the Photran Fortran IDE that is implemented as part of Eclipse.

The prototype example implementation developed over the course of this project uses the Eclipse JUnit implementation as a reference implementation of a unit testing framework in Eclipse. Wherever possible the FRUIT UI and usage will be modeled on the JUnit implementation.

## Definitions

1. *FRUIT* -- a unit testing framework for Fortran programs, modeled after the xUnit family.

## References

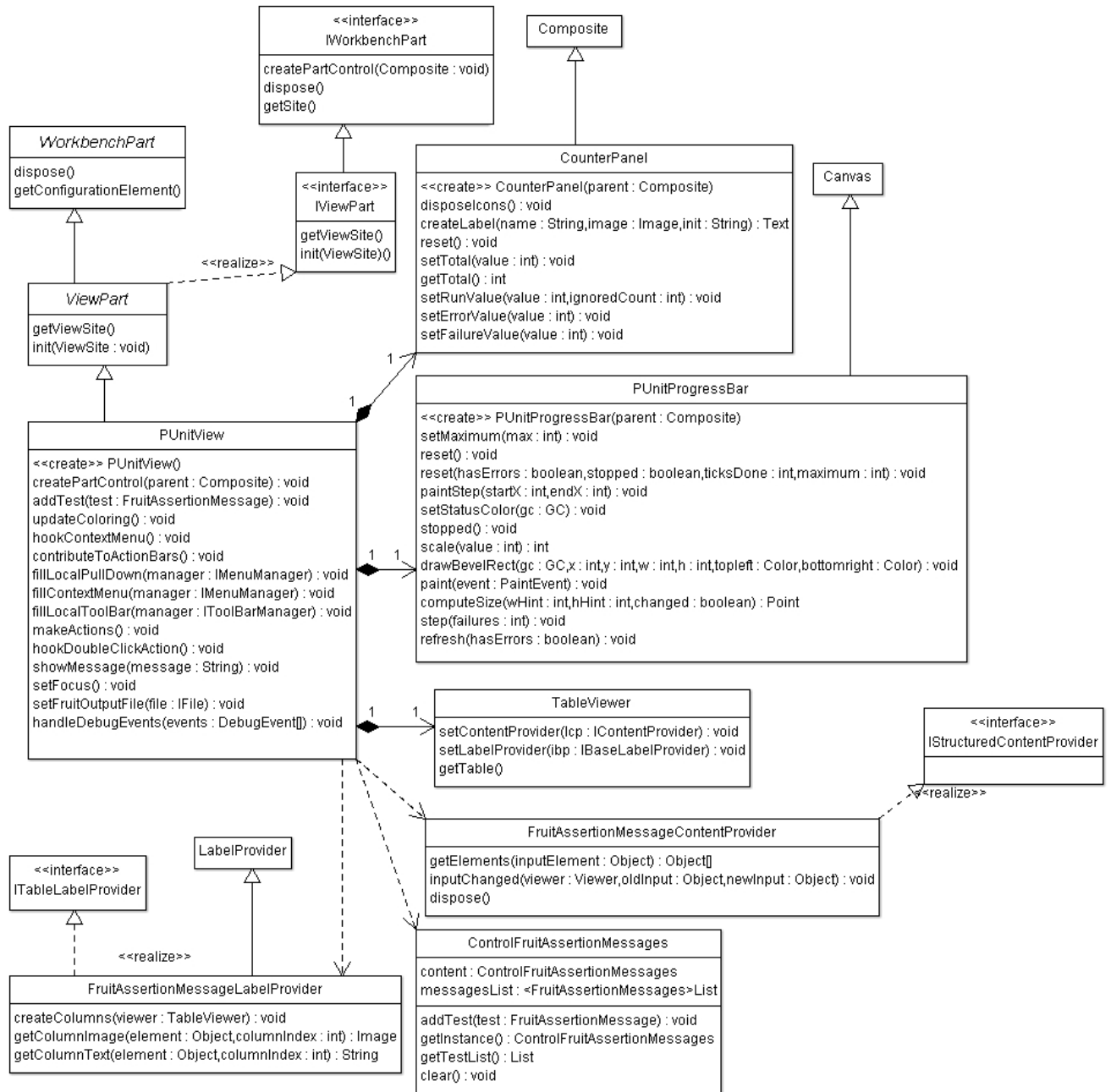
1. *Photran Developer Guide* [[http://dev.eclipse.org/viewcvs/index.cgi/~checkout~/org.eclipse.photran/org.eclipse.photran-dev-docs/dev-guide/dev-guide.pdf?cvsroot=Technology\\_Project](http://dev.eclipse.org/viewcvs/index.cgi/~checkout~/org.eclipse.photran/org.eclipse.photran-dev-docs/dev-guide/dev-guide.pdf?cvsroot=Technology_Project) ]

---

# Chapter 2. Architecture and Design

## Overview

We broke up our project into several distinct units of functionality that were coded independently then integrated together near the end of the development process. The integration of the FRUIT library and driver generation tools into the Photran framework required several complex changes to a build system that expects to generate a single binary executable from a series of Fortran source files. In addition the FRUIT distribution contains some Ruby scripts that generate some driver files necessary to exercise the FRUIT tests.

**Figure 2.1. Nibbler UML Diagram**

## Components

### User Interface

The user interface was designed and modeled after the JUnit view that is shipped with the standard Eclipse distribution. At first, it was decided to take the JUnit view code and simply modify it to hook into the other components that had been created for FRUIT testing.

After some initial discovery, it was determined that it would make more sense to create a completely new view plugin from scratch and port bits of JUnit code to the new view to emulate the look and feel of JUnit. This way, the resulting PUnit view is custom built to read directly from the FRUIT parser component and display results accordingly. The JUnit labels, images, and status bar were all able to be ported easily, but to display the FRUIT parser's output more efficiently, a table was chosen in place of the existing JUnit GUI components. This table displays the output in a friendly, condensed form.

The PUnit view is an extension to `org.eclipse.views`. It provides an informative GUI displaying test summaries. It is designed and modeled after the JUnit view. New views are created in Eclipse by defining a category and the view in the plug-in manifest file. Since the PUnit view is used for Fortran testing, it is added to the Fortran category. The view needs a name, an id, and a class. The view is displayed by running the runtime workbench and going to Window->Show View->Other->Fortran->PUnit

## Figure 2.2. PUnit View

In Eclipse, views must implement the `IViewPart` interface. The abstract class `ViewPart`, which is a subclass of `WorkbenchPart` implements that interface. The PUnit view class, `PunitView` implements the `IViewPart` interface by subclassing the `ViewPart` abstract class. The view is built using the Composite pattern. The view is composed of three major elements defined in the required `CreatePartControl(Composite)` method, which creates the controls comprising the view:

- The counter panel is a panel that displays the counters for the number of Runs, Errors, and failure. The `CounterPanel` Class is a Subclass of `Composite` which is contained in the `Composite` Parent.
- The Progress bar is a JUnit like progress bar, which turns green for test success and red for failures. The `PUnitProgressBar` Class is a subclass of `Canvas` and has a constructor that takes the parent `Composite` of the view.
- A Table viewer that displays the result of the test. Each row represents a test while each column represents an attribute of the test: name, Elapsed time, Test passed, Expected value, Actual value. Each row of the table is colored in red or green depending on if the test passed or failed. The `Tableviewer` must be configured with a content provider; witch implements the `IStructureContentProvider` interface and `Label` provider, witch must implement the `IBaseLabelProvider`.

The result of Fruit tests are represented in `FruitAssertionMessages` objects. The contain provider class `FruitAssertionMessageContentProvider` implements the required `getElements` method, which returns a list of `FruitAssertionMessages` to display. The `Label` provider class `FruitAssertionMessageLabelProvider` is a subclass of `LabelProvider` and implements the `ITableLabelProvider`. This interface extends `IBaseLabelProvider` with method to provide text and image for each column of a given test. This class creates the column of the table with the 5 tests attributes: Test name, Elapsed Time, Test Passed, Expected Value, Actual Value. The output generated by the fruit parser is used to populate the PUnitview. Below is a Class diagram of important classes:

## Figure 2.3. PUnit View Class Diagram

# Test Case Generation

When a user wishes to create a new FRUIT test case two options exist that are supported by the Photran FRUIT system. First the user can create the FRUIT test case completely manually by just creating a new Fortran source file in Photran and following the appropriate rules for creating a FRUIT Test Case.



The Photran FRUIT tools can handle test cases that are created manually in this fashion, and will be automatically detected by the build system when a Build and Launch occurs.

In addition to the manual procedure Photran now provides a wizard based test case generation procedures. This provides a means to create a minimal skeleton test case in a similar means to that provide by the JUnit framework within Eclipse. Under the File->New hierarchy the user will be able to select "New Fruit Test Case". This wizard will prompt for the Fortran file that the test case will test, along with the name of the Fortran module within that file. Upon completion of the wizard, a new skeleton test case file will be created that provides some of the framework required to build a FRUIT test case.

## Build and Launch System

### Building

When the user performs a normal system build Photran will perform the build as normal, the FRUIT code will not affect normal builds except for the fact that the `fruit_test_driver.f90` is excluded from the build.

When a user runs the "Run Fruit Test" context menu entry on the project, the FRUIT build system will take over. The `FruitLaunchManager` module will take over at this point and get Photran into a state to display the results of the FRUIT tests.

First the system will verify that the FRUIT configuration is up to date with all the latest test case files that are in the system. The system will also update the `fruit_test_driver.f90` and `fruit_basket_gen.f90` files which comprise the FRUIT test driver application.

Photran will then instruct the build system to build the FRUIT configuration. The configuration is explicitly built since there appear to be some issues with trusting the Eclipse Launching system to always automatically rebuild it when necessary.

### Execution

Once the FRUIT configuration is successfully built, Photran will verify that a correct Launch Configuration exists to execute the FRUIT test driver correctly. This Launch Configuration consists of a link to the actual application along with a variety of configuration settings. The core changes from a standard Launch Configuration that are used for FRUIT are that the application will not be connected to a console, and the output will be redirected to a file "`fruit_output.log`". This file will contain the output of the FRUIT run that will be parsed and loaded into the PUnit view.

Since aspects of the execution actually run in the background after the context menu handler exits, a way is needed to communicate to the view that a new `fruit_output.log` file is available to be parsed and displayed. `PUnitView` will listen for `DebugEvents` to determine when the FRUIT test application has finished execution and sends the `TERMINATE` event.

## Driver Generation and Output Parsing

### Driver Generation

Before the FRUIT tests can be built and run they first needs some setup code to be built that is specific to each test suite. Since Fortran doesn't support Reflection as is the case with Java, FRUIT cannot directly find these tests cases and execute them without any assistance as is the case with JUnit.

In the FRUIT distribution there exists a collection of Ruby scripts that will parse the Fortran files in a given project and generate the appropriate driver files from the test case source files. For this project this

code needed to be integrated into Photran so that this process could be automatically run whenever the FRUIT test cases need to be rebuilt and run.

In order to accomplish this the Ruby scripts were rewritten in Java and integrated into Photran. When the driver generator is run it will scan through all the all the files in the directory provided to determine which ones contain test cases and will then add these tests to the overall driver.

---

# Chapter 3. Project Postmortem

## Future Plans

Currently CS427 Team Nibbler has no future plans for this project. With additional testing and code cleanup there is a possibility that it could be used within Photran. For this project to actually be used with Photran several design decisions would most likely need to be analyzed to see if the choices that were made by Team Nibbler will actually fit within the model that is provided within Photran. The primary issues that would need to be analyzed are

- Use of Build Configurations to support building FRUIT application
- Better integration of FRUIT distribution into Photran
- Decision to parse fruit output. Integration with the FRUIT project would potentially allow a more stable output format.

## Team Member Reflections

**Dave Buhl**

**Anthony Gelsomini**

**Pat Guelah**

**Charlie Meyer**

---

# Appendix A. Software Installation

The Nibbler FRUIT plugins for Eclipse are designed and tested for use under Windows XP with Cygwin providing the required Unix compatibility layer. The installation instructions provided are targeted at this environment.

## Requirements

- Eclipse 3.4 Distribution for Plug-in Developers
  - CDT Version 5.0.0 Sources
  - Photran HEAD Sources
  - Nibbler Sources
- Cygwin
- GFortran for Cygwin 2008-11-18
- FRUIT 2.6
- Ruby Gems

## Installation

### Eclipse 3.4 Distribution for Plug-in Developers

1. Download Eclipse 3.4 For Plug-in Developers from <http://www.eclipse.org/downloads/packages/eclipse-rcplug-developers/ganymeder> [http://www.eclipse.org/downloads/packages/eclipse-rcplug-developers/ganymeder]
2. Eclipse doesn't provide an installer. Once uncompressed the files can be stored anywhere on the system. To run Eclipse, the user needs to launch the Eclipse.exe binary inside the uncompressed Eclipse directory.

### CDT and Photran Sources

1. To install the CDT and Photran Sources follow the detailed instructions provided in Appendix A of the Photran Developers Guide available at: [http://dev.eclipse.org/viewcvs/index.cgi/~checkout~/org.eclipse.photran/org.eclipse.photran-dev-docs/dev-guide/dev-guide.pdf?cvsroot=Technology\\_Project](http://dev.eclipse.org/viewcvs/index.cgi/~checkout~/org.eclipse.photran/org.eclipse.photran-dev-docs/dev-guide/dev-guide.pdf?cvsroot=Technology_Project) [http://dev.eclipse.org/viewcvs/index.cgi/~checkout~/org.eclipse.photran/org.eclipse.photran-dev-docs/dev-guide/dev-guide.pdf?cvsroot=Technology\_Project]
2. One deviation from the directions provided with Photran is that instead of using the CDT\_5\_0 branch, the user should use the CDT\_5\_0\_0 tag as this tag point is the one that is compatible with the version of Photran in the HEAD branch.

### Nibbler Project Sources

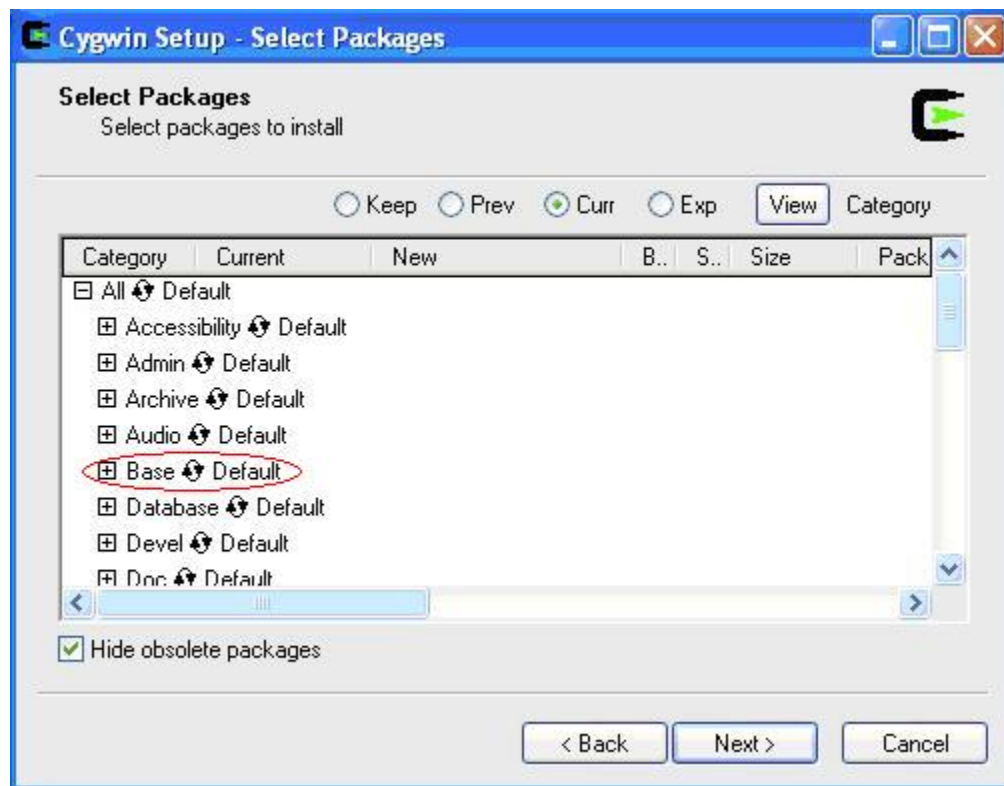
1. Checkout the sources from the CS427 Subversion repository at the FinalSubmission tag: <https://csil-projects.cs.uiuc.edu/svn/fa08/cs427/Nibbler/tags/FinalSubmission> [https://csil-projects.cs.uiuc.edu/svn/fa08/cs427/Nibbler/tags/FinalSubmission] The required packages are:

- edu.uiuc.cs427.nibbler.fruit.core
- edu.uiuc.cs427.nibbler.fruit.tests
- edu.uiuc.cs427.nibbler.fruit.wizards
- edu.uiuc.cs427.nibbler.punit

## Cygwin

1. To install Cygwin, download the latest version of Cygwin from <http://www.cygwin.com/setup.exe> [http://www.cygwin.com/setup.exe] . Once downloaded, run the executable `setup.exe`.
2. Select all of the "Base" utilites by clicking on the word "Default" next to Base until it shows "Install".

**Figure A.1. Cygwin Install Screen**



3. Select the following "Developer" utilities: *autoconf*, *automake*, *binutils*, *gcc*, *gdb*, *make*, and *libtool* by expanding the "Devel" category and selecting the checkboxes next to each of the mentioned packages.
4. Change Category to "Full", by clicking on the button that displays "View" to the left of the category until the button displays "Full".
5. Select *gmp*, *mpfr*, and *ruby* by selecting the checkboxes next to each of the mentioned packages.
6. The installation can now be completed by selecting "Next" repeatedly until the installation completes.

## GFortran

1. Download the gfortran archive from *GFortran* [<http://gcc.gnu.org/wiki/GFortranBinaries>] by clicking the "here" link in the Windows Cygwin section and then save the file in your `{cygwin_install_directory}\home\{user_name}` directory.
2. Open a Cygwin shell and make sure that you are in your home directory by typing the command: `cd ~` at the prompt.
3. Ensure the gfortran archive is in your directory by using the command `ls` at the prompt. You should see a list of the files in the directory printed to the console. One of them should be named similarly to `gfortran-4.4-Cygwin-i686.tar.bz2`
4. Extract and install gfortran by using the command `tar -xjvf gfortran-4.4-Cygwin-i686.tar.bz2 -C /` Do not forget the trailing slash. Also, this command is case sensitive and you will need to replace the filename given in the sample command with the name of the file that you actually downloaded.

## Setting up Environment Variables (Windows)

1. Cygwin does not put itself on the Windows path. You need to add it to the PATH environment variable. Go to Control Panel > System. Click on the Advanced tab, then click on the Environment Variables button. Go to the System Variables section. Look for PATH, select it, and click edit. Prepend `C:\{cygwin_install_dir}\usr\local\gfortran\bin;C:\{cygwin_install_dir}\cygwin\bin;` to the path. Remember that paths are separated by `;`, so do not leave them out. Double check that there are not any trailing or dangling symbols, etc. Also, replace `{cygwin_install_dir}` with the location you installed Cygwin.
2. You will need to restart Windows before Eclipse will recognize the new PATH.

## Setting Up Environment Variables (Cygwin)

1. Open a Cygwin bash shell.
2. Edit the file `.bash_profile` in your home directory with your favorite text editor.
3. Add `PATH=/usr/local/gfortran/bin:${PATH}` to the end of the file. Note the direction of the slashes and remember that this is case sensitive. Save and close the file.
4. If you used a Windows editor to change `.bash_profile` such as Notepad or Wordpad, you will need to eliminate the carriage returns that Windows editors put into files. Use the command `dos2unix .bash_profile`
5. Close your shell and then open a new one to load the new PATH.

## Ruby Gems

1. Download the latest ruby gems archive from *Gems* [[http://rubyforge.org/frs/?group\\_id=126](http://rubyforge.org/frs/?group_id=126)] and save it to your home directory in Cygwin.
2. Expand the archive by issuing the command `tar -xvf rubygems-1.3.1.tgz` using the filename that you downloaded in place of `rubygems-1.3.1.tgz`

3. Change to the directory that contains the file `setup.rb` which should be in the root directory named for the archive, in this case `rubygems-1.3.1`
4. Install ruby gems by issuing the command `ruby setup.rb install`

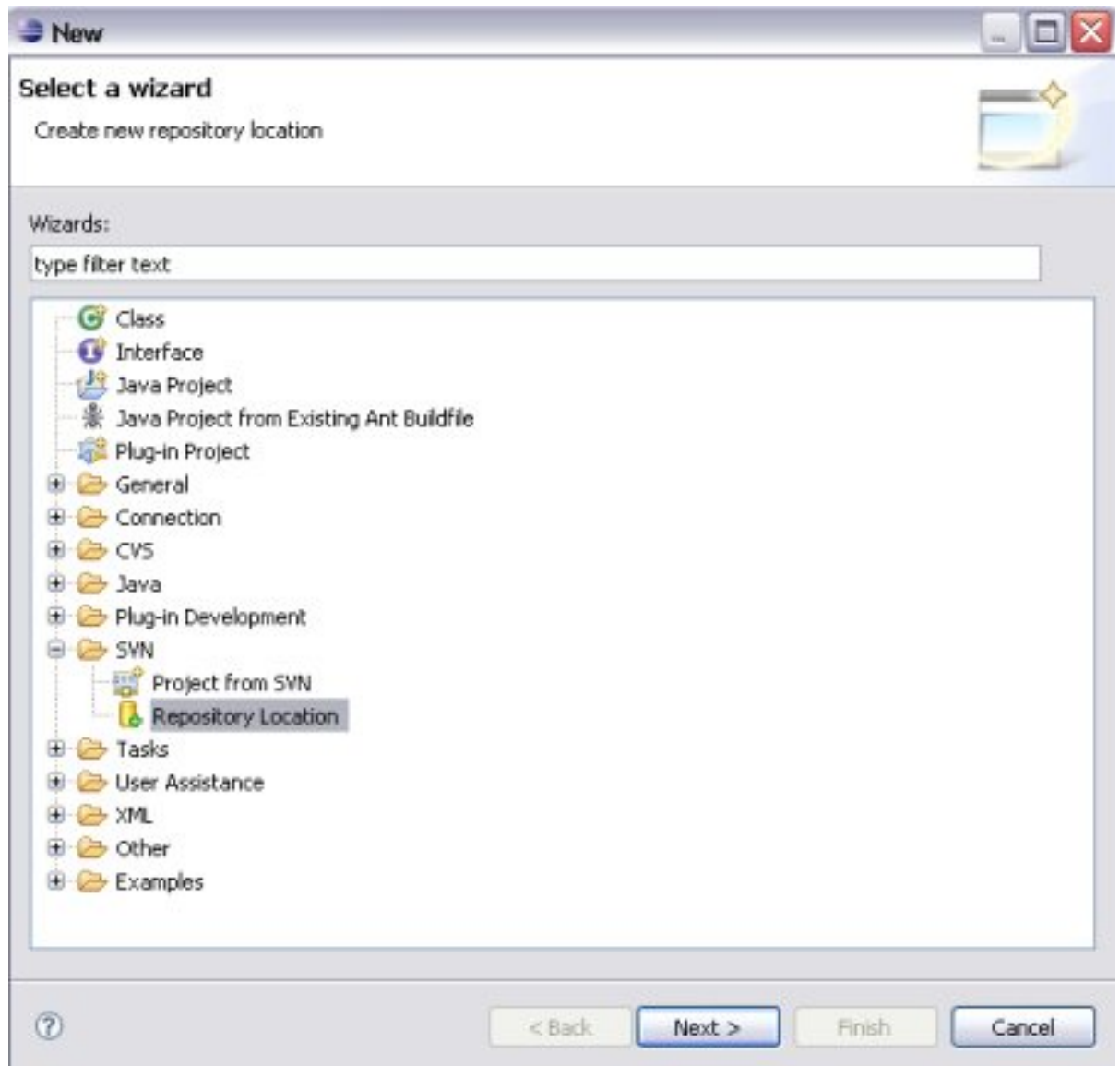
## FRUIT

1. Download the latest version of FRUIT from <http://sourceforge.net/projects/fortranxunit> [http://sourceforge.net/projects/fortranxunit] and save it in your Cygwin home directory.
2. Unzip the archive by issuing the command `unzip fruit_2.6.zip` replacing `fruit_2.6.zip` with the name of the file that you downloaded.
3. Change into the directory created, in this case `cd fruit_2.6`
4. Make the following edits to the `rake_base.rb` in the `fruit_2.6` directory:
  - a. Change `$compiler='ifort'` to `$compiler='gfortran'`
  - b. change `sh "#{$compiler} -g -debug inline_debug_info -c -o #{t.name} #{t.source} -module #{build_dir} #{FruitProcessor.new.inc_flag($inc_dirs)}"` to read `sh "#{$compiler} -g -c -o #{t.name} #{t.source} -J #{build_dir} #{FruitProcessor.new.inc_flag($inc_dirs)}"` Note how the `-debug` option was removed and the `-module` option is replaced by `-J`
5. Next, you have to make some small changes to the `fruit.f90` file in the `src` directory in order for fruit to compile with `gfortran`. There are two function declarations in the file that you must add `()` to. The first is function `get_last_message` should be changed to function `get_last_message()` and function `is_last_passed` should be changed to function `is_last_passed()`
6. Install the Rake Gem using the command `sudo gem install rake`
7. Change to the `fruit_processor_gem` directory and then issue the command `sudo rake install` to install the fruit processor gem.
8. In the fruit directory, run `rake clobber` and then `rake` in build FRUIT.

## Nibbler Code

The following instructions on how to download the Nibbler code into your workspace assume you have the Subversive Eclipse plugin installed. If you do not have it installed, please see the instructions at *the Subversive web page* [http://www.eclipse.org/subversive/]. We also assume that your UIUC netid has access to the Nibbler SVN repository. Please contact TSG if you need to be given access to the repository.

1. Make sure you are in the same workspace where you checked out the CDT and Photran source code.
2. Select File > New > Other... > Expand the SVN Folder > New Repository Location

**Figure A.2. New Repository Location**

3. Fill in the the information as follows:

- *URL* - `https://csil-projects.cs.uiuc.edu/svn/fa08/Nibbler`
- *Label* - Select "Use the repository URL as the label"
- *User* - Your UIUC netid
- *Password* - Your UIUC Active Directory Password



**Figure A.3. New Repository Location Window**

**New Repository Location**

**Enter Repository Location Information**

Define the SVN repository location information. You can specify additional settings for proxy and svn+ssh, https connections.

SVN

General | Advanced | SSH Settings | SSL Settings

URL:  Browse...

**Label**

☒ Use the repository URL as the label

☐ Use a custom label:

**Authentication**

User:

Password:

☐ Save password

⚠ Saved secret data is stored on your computer in a file that's difficult, but not impossible, for an intruder to read.

Show Credentials For:

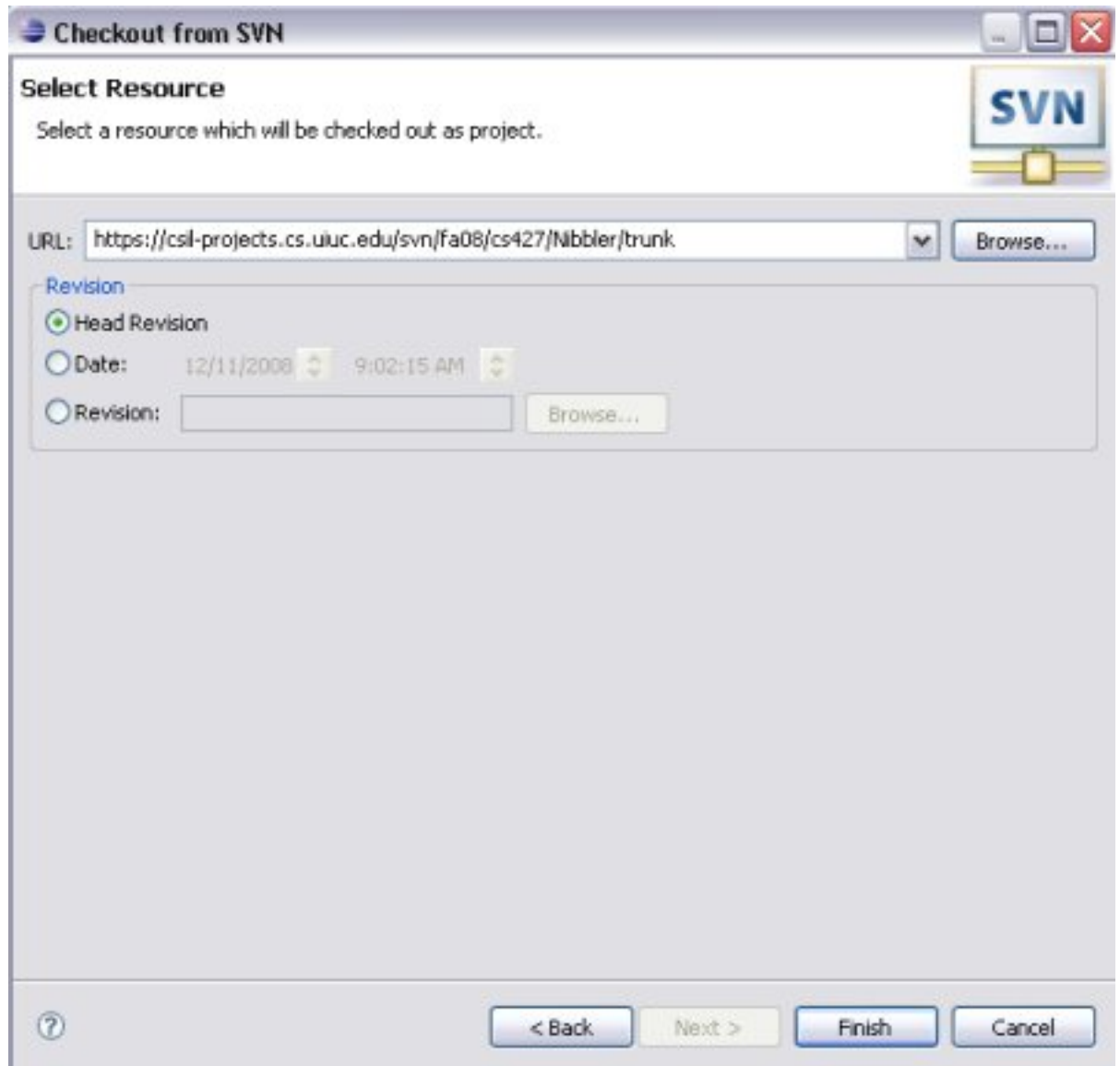
☒ Validate Repository Location on finish

Reset Changes

? < Back Next > Finish Cancel

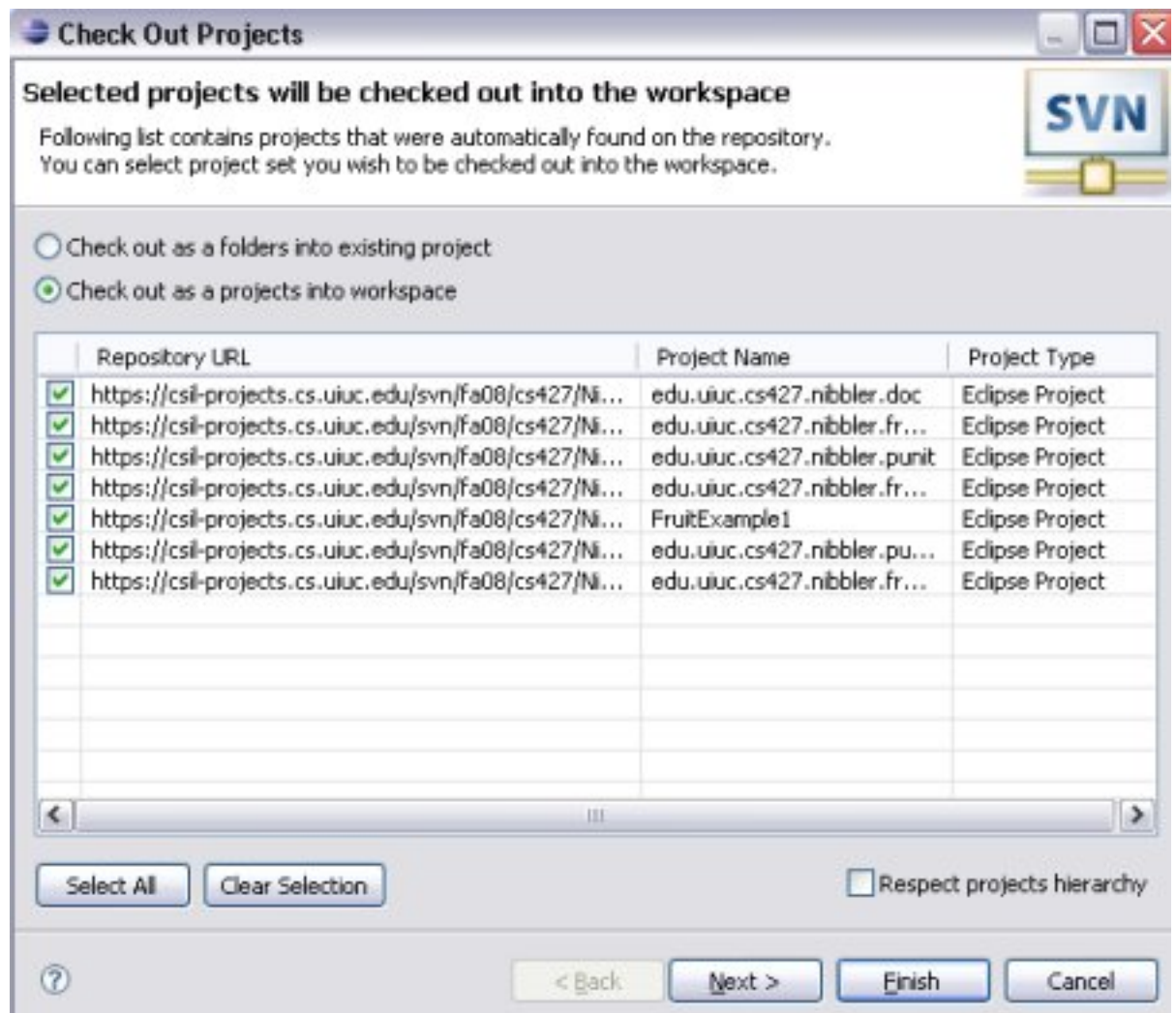
Click Finish

4. Select File > New > Expand the SVN folder > Project from SVN
5. Select the newly created repository location in the next window
6. Add /trunk to the end of the URL on the next window and select Head Revision

**Figure A.4. Checkout from SVN**

Click Finish.

7. On the next window, select the option "Find Projects in the Children of the Selected Resource". Click Finish.
8. Subversive will then recursively scan the repository to find projects. This will take a minute or so. On the window that appears next, select all of the projects then click Finish.

**Figure A.5. Checkout Projects**

9. All the code will now be downloaded from the repository into your workspace.

---

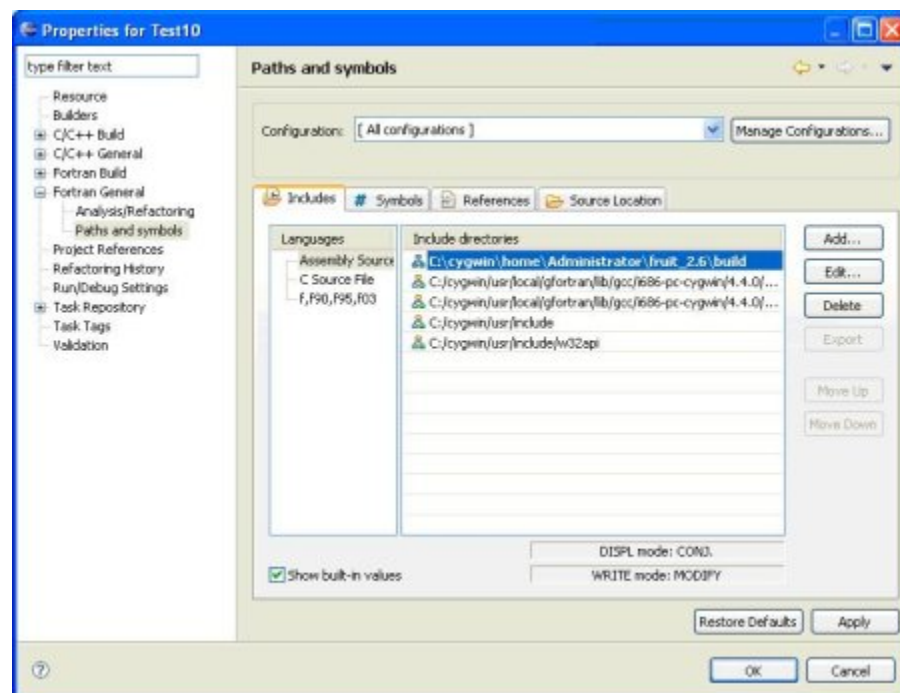
# Appendix B. Software User Guide

## Fortran Project Creation

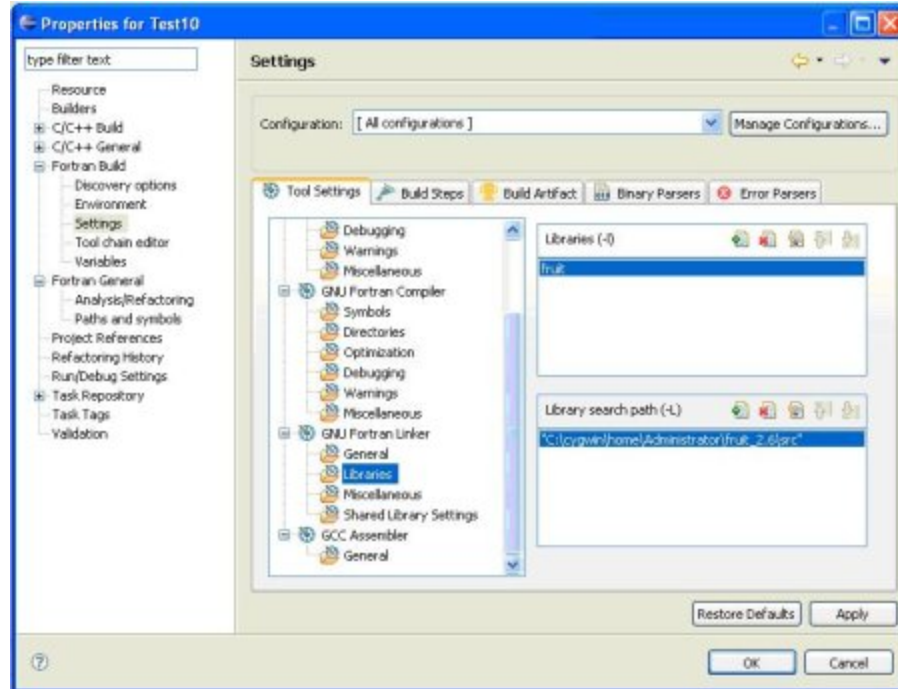
The Photran FRUIT plugin supports using normal Photran Fortran Projects. As a result the first step in using the FRUIT plugin is to create a Fortran Project in Eclipse. After the project is created the following changes need to be made to support the building of FRUIT projects.

1. Under Project Properties > Fortran General > Paths and Symbols
  - a. Select Configuration "All Configurations"
  - b. Select the fortran files under languages (f,f90,f95,f03)
  - c. Then add a new include path. This will be a path to the fruit module files. In this case, they are in the {cygwin\_home\_directory}\fruit\_2.6\build.

**Figure B.1. The Include Paths Required for FRUIT**



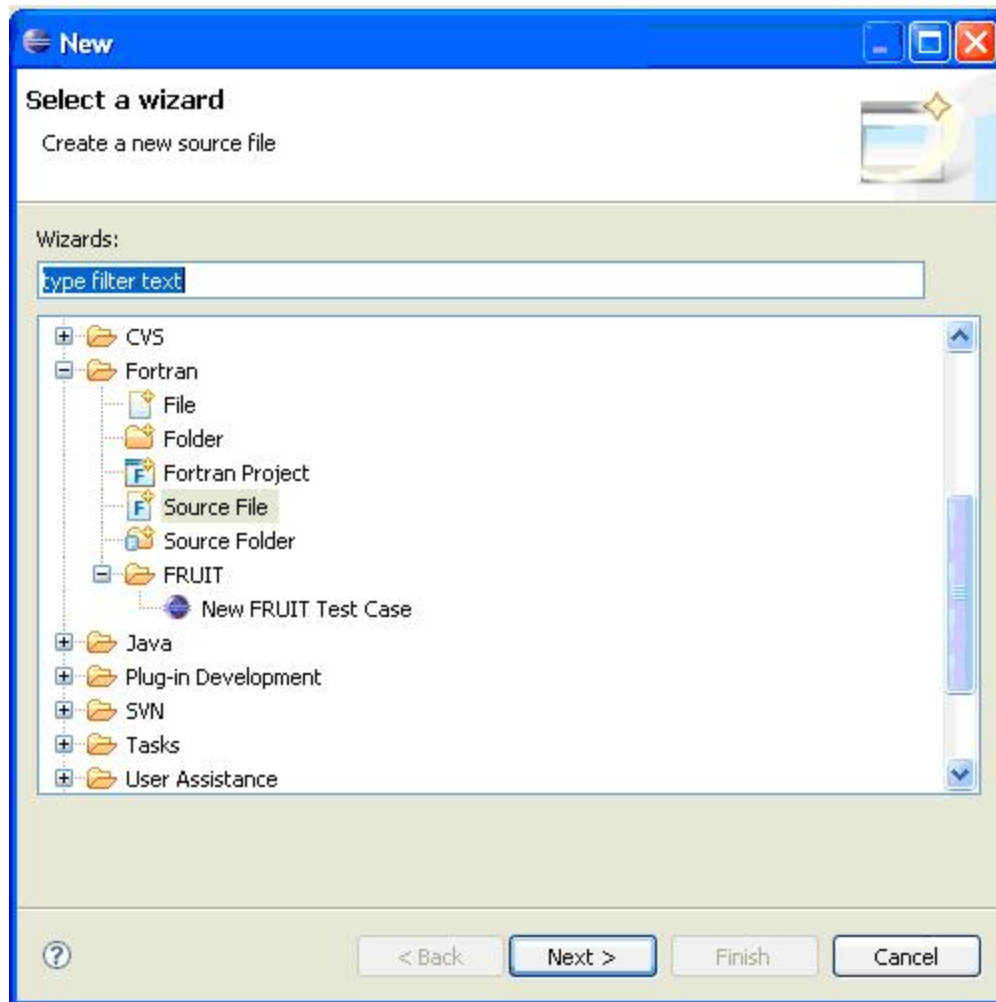
2. Under Project Properties > Fortran Build > Settings
  - a. Select Configuration "All Configurations"
  - b. Select the libraries entry under Fortran Linker
  - c. In the Libraries (-l) window, create a new entry, place `fruit` in the text box
  - d. In the Library Search Path, create a new entry and place the path to `libfruit.a`, in this case it is located in {cygwin\_home\_directory}\fruit\_2.6\src

**Figure B.2. Configuring FRUIT libraries**

## Creating a Test Case

The Photran FRUIT plugin provides a means to create a skeleton test case file. This is similar to the approach JUnit follows which provides a wizard to create a skeleton version of the test case. This skeleton doesn't provide any actual test case code, it is primarily used to provide the repetitive code that needs to be created for each test case.

To create a new test case a new entry under the File->New menu is provided under the Fortran/FRUIT subcategory.

**Figure B.3. Create a FRUIT test case**

Upon selecting “New FRUIT Test Case” the user will be presented with a prompt to supply both the file containing the Module Under Test and the actual module name in that file.



Upon clicking “Finish” the skeleton file `calculator_test.f90` (or another `f90` file named after the module under test) will be created and populated with a basic test case outline.

## Running FRUIT Tests

Once all the test cases are written, to run the test suite the user will right click on the Project and select “Run FRUIT Test”. When “Run FRUIT Test” executes it performs several actions before finally displaying the results. These steps are as follows:

- Update `fruit_basket_gen.f90` which contains the Fortran driver to run each FRUIT Test Case specified in the test files in the project.
- Update `fruit_test_driver.f90` which contains the main Fortran program definition.
- Add/Update a build configuration called “FRUIT”. This build configuration selects just the test case files along with the files of the modules under test to be built.
- Build the “FRUIT” Build Configuration.
- Run the executable created by the “FRUIT” Build Configuration to generate the test case output.
- Populate the PUnit View to display the “FRUIT” results.

When FRUIT completes, the PUnit View will be displayed showing the results of each test case along with how many tests passed and failed.

**Figure B.4. The PUnit View Displaying Results**

Runs: 3/3   Errors: 0   Failures: 1				
Test Name	Elapsed Time	Test Passed?	Expected Value	Actual Value
test_calculator_...	0.0	true		
test_more_with...	0.0	false	4.0000000	4.0999999
test_calculator_...	0.0	true		



---

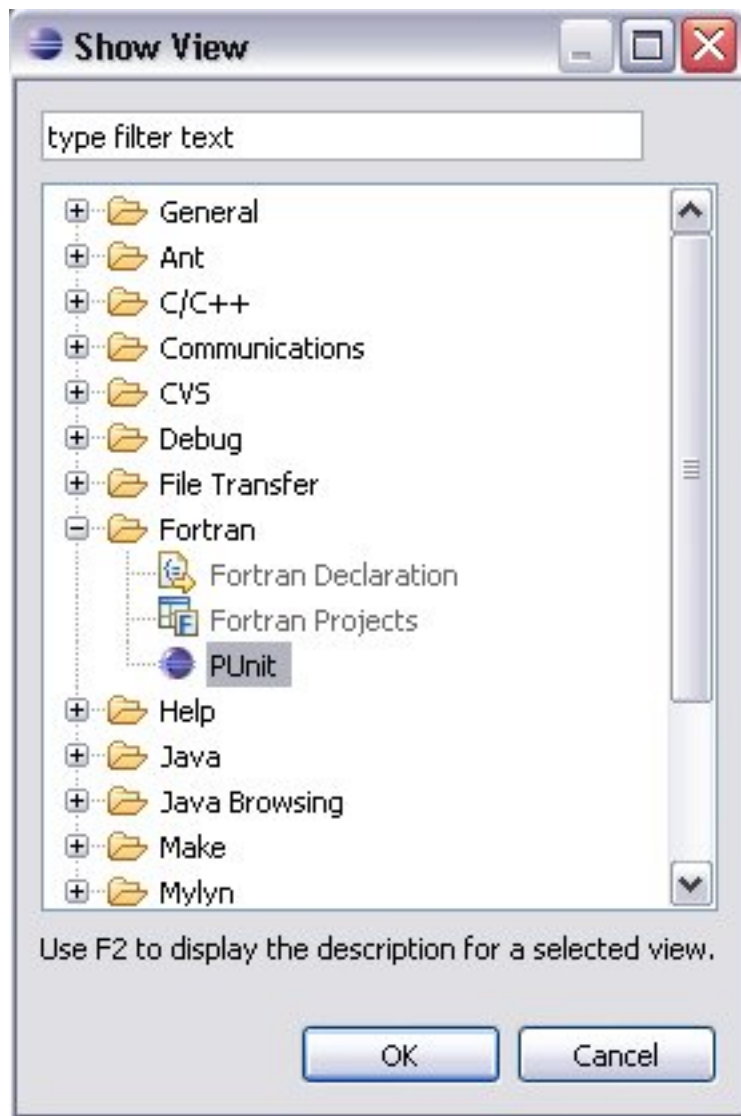
# Appendix C. Test Suite Usage

## Manual Test Procedures

### Verifying Empty View

*This test assumes that your Eclipse workspace has been set up according to the instructions earlier in this document. If you have not done so, please go back and set up your workspace with all of the source code properly before continuing with this test.*

1. Right click on any of the Nibbler projects in your workspace, select Run As > Eclipse Application.
2. Once the new instance of Eclipse loads, select the Window Menu > Show View > Other
3. Expand the Fortran folder. Select the PUnit view, click ok.

**Figure C.1. Select the PUnit View**

4. Verify that the view that launches matches the screenshot below. *Note that your tabs might not match exactly the ones in the figure below, that is ok.*

**Figure C.2. The Empty Punit View**

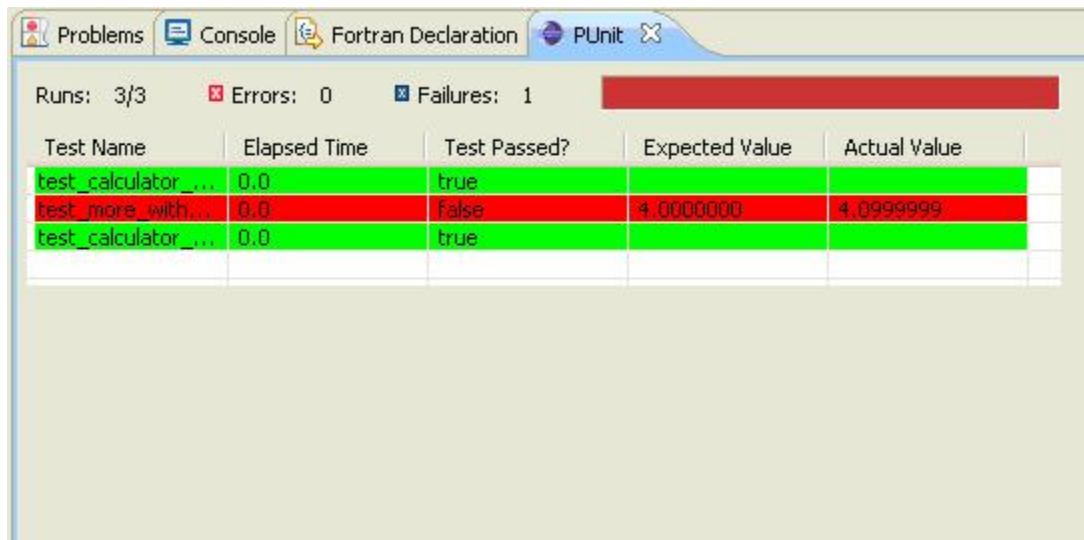
## Running The Sample FRUIT Tests

*This test assumes that your Eclipse workspace has been set up according to the instructions earlier in this document. If you have not done so, please go back and set up your workspace with all of the source code properly before continuing with this test.*

1. Right click on any of the Nibbler projects in your workspace, select Run As > Eclipse Application.
2. Once the new instance of Eclipse loads, install the Subverse Plugin as described earlier in this document [11].
3. Check out the Nibbler code into the new instance of Eclipse as described earlier in this document with one minor difference. Once the screen appears with the check boxes next to each project, uncheck all projects except for FruitExample1, then continue per the instructions.
4. The checked out project, FruitExample1, is a Fortran Project containing the sample code that is shipped with FRUIT. Because it is a Fortran Project, it contains metadata that references specific locations of compilers and other machine dependent components that are specific to the computer where the project was first created. For this reason, create a new Fortran Project by selecting the File menu > New > Other > Expand the Fortran Folder > Fortran Project.
5. Name the project 'Calculator' or something similar. For Project type, select Executable (Gnu Fortran). Uncheck the "Show project types and toolchains only if they are supported on the platform" checkbox, then select GCC Tool Chain under Toolchain. Click finish.
6. Right click on the new project, select Properties. On the resulting window, click on C/C++ Build. Select "All Configurations" on the drop down menu. Then select settings under C/C++ build. Click on the Binary Parsers tab. Deselect any of the checked items, then check Elf Parser. Finally, click once on Elf Parser and click the Move Up button until it is at the top of the list.
7. From the FruitExample1 project, right click on `calculator.f90`, and select Copy. Then right click on the new project, and select paste.
8. Right click on the new project, select New > Other. Expand the Fortran folder, then expand the FRUIT folder. Select New FRUIT Test Case. In the next window, Expand workspace, then single left click on the name of the project you copied `calculator.f90` into. Then select `calculator.f90` in the right pane by checking its check box. Select ok. For module under test, enter `calculator` into the text box. Click ok.

9. Open the newly created `calculator_test.f90` file in the editor by double clicking on it. Delete all its contents. Then open `calculator_test.f90` from the `FruitExample1` project. Copy all of its contents and paste them into the now empty `calculator_test.f90` file.
10. Right click on the new project, select `Run FRUIT Test`. The PUnit view should appear and be populated. Verify that its contents match the figure below. *Note that your tabs might be different than those shown in the figure, that is acceptable.*

**Figure C.3. PUnit View After Manual Test**



Test Name	Elapsed Time	Test Passed?	Expected Value	Actual Value
test_calculator_...	0.0	true		
test_more_with...	0.0	false	4.0000000	4.0999999
test_calculator_...	0.0	true		

## Automated Test Procedures

### FruitLaunchManagerTest

The Fruit Automated tests requires a specially setup workspace to run the test on. This workspace is included in the `edu.uiuc.cs427.nibbler.fruit.tests` plugin.

1. In Eclipse right click on the `edu.uiuc.cs427.nibbler.fruit.tests` and select `Run As > Run Configurations...`
2. In the Run Configurations Dialog Box create a new JUnit Plugin Test Configuration
3. On the test tab select run all tests in the selected project, and specify `edu.uiuc.cs427.nibbler.fruit.tests` as the project.
4. On the main tab, for the workspace data field, select the `Workspace` button and select `edu.uiuc.cs427.nibbler.fruit.tests/src/workspaceForFruitLaunchManagerTest`. Verify that the `clear` checkbox is not selected.
5. Select `Run` and this configuration will run the `FruitLaunchManagerTest` using the workspace provided at `edu.uiuc.cs427.nibbler.fruit.tests/src/workspaceForFruitLaunchManagerTest`