



**University of Illinois at Urbana-Champaign**  
**Department Of Computer Science**

## **Uchuva Managed Build Project**

**CS 427 - Fall 2012**

**Professor Ralph Johnson**

**Teaching Assistant Caius Brindescu**

Prepared By:

**Morrow, Lorentz Arthur**

**O'Sullivan, Michael**

**Kolavenu, Ramesh**

**Lozano Hinojosa, Jose Alberto**

**Biglari, Mehrdad**

# Table of Contents

Title	Page
Managed Build System Overview.....	3
Project Description.....	3
Architecture and Design.....	4
<i>Managed Build &amp; Module Name</i> .....	4
<i>New Project Wizard</i> .....	6
Future Plans .....	8
User manual .....	8
Appendix.....	11
<i>Installation Procedures</i> .....	11
<i>Prerequisites</i> .....	11
<i>Running the System</i> .....	12
<i>Running the Unit Tests</i> .....	13

## Managed Build System Overview

The Managed Build System in Photran is responsible for creating the makefile which is used to compile the executable. Managed Build has a few shortcomings which prevent Photran users from doing things they could normally do with FORTRAN. There are also some usability issues with the new project wizard which we consider to be a part of Managed Build.

Uchuva's Managed Build Project addresses the dependency between a module name and its file name and the ability to have multiple modules in a single source file. Additionally, the confusion around the new project wizard has been addressed by correctly filtering the project types by the OS and toolchains available on the system Photran is running on.

## Project Description

This section summarizes the changes of managed build related to file name and module name. The following items were fixed:

- Managed Build compiles and links pure FORTRAN file that contains module with any valid name. Before, Managed Build requires having file and module have the same name.
- Managed Build compiles and links pure FORTRAN file that contains more than one module. Before, managed build only references the first module in the file if its name matches the file name. In the current version, Managed Build references all modules of a FORTRAN file

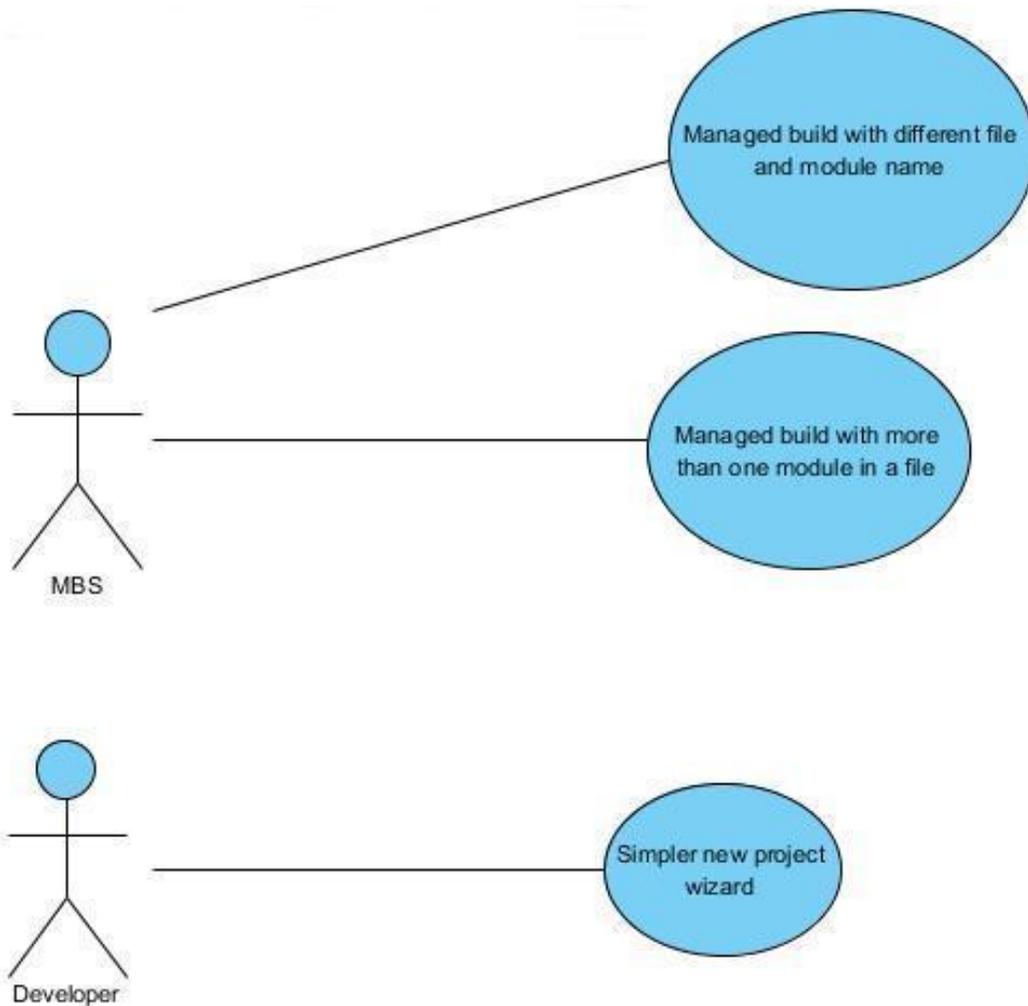


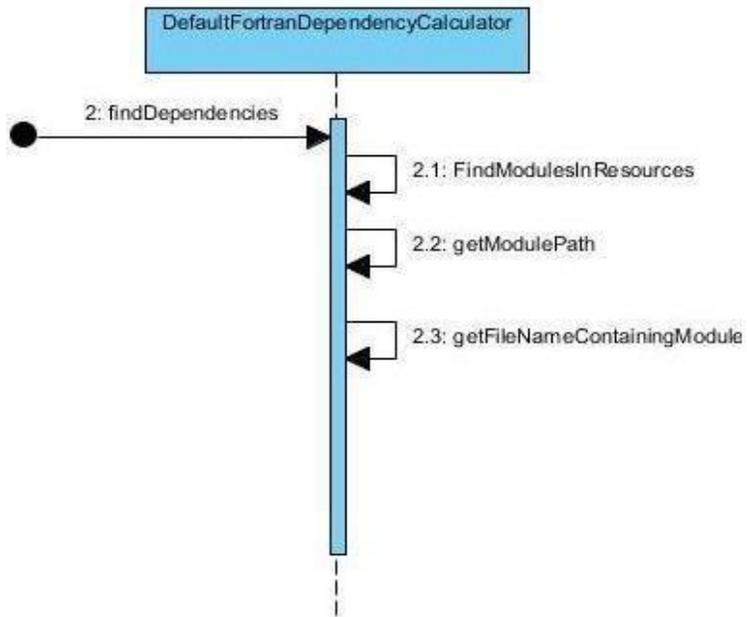
Figure 1

## Architecture and Design

### *Managed Build & Module Name*

The MBS relies on a Dependency Calculator in order to resolve the dependencies for each target that will be listed in the `makefile`. We enhanced the existing FORTRAN Dependency Calculator to use the Photran VPG instead of brute force text parsing. Figure 2 shows the before and after behavior of the part of the system we modified.

## Before



## After

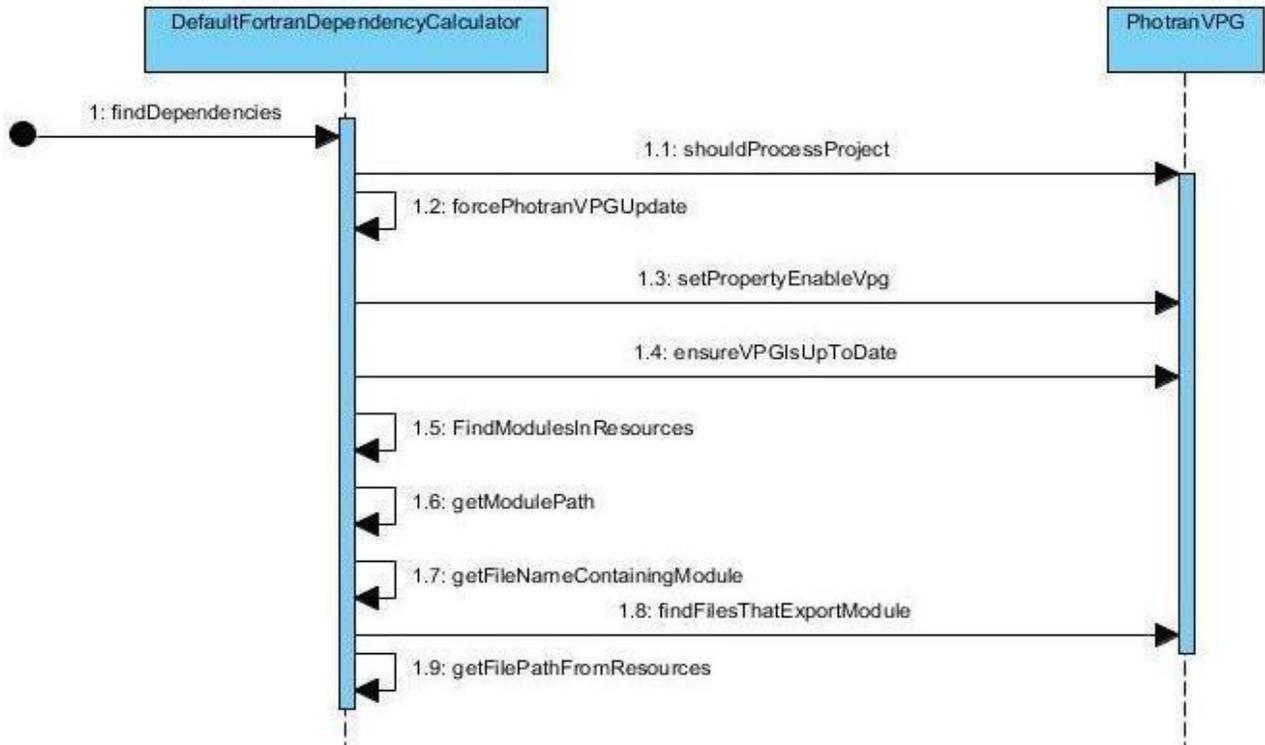


Figure 2

## New Project Wizard

When determining the set of supported project types, CDT delegates to the project's associated configurations, which delegate to their associated tool chains, and so on. If the tool chains in all configurations report that they are unsupported, then that project type is considered unsupported. By default everything reports that it is supported. The `toolChain` schema, part of the Build Definitions Model, includes a property (`isToolChainSupported`) where the tool developer can specify a class that implements the `IManagedIsToolChainSupported` interface to override the response of the tool chain.

We created a set of classes, one for each project type, which determine their state on the current machine. We specified these classes in the plugin.xml files for the relevant Photran packages. These new classes also check the current operating system and filter based on that information as well as the compilers installed on the machine. Figure 3 shows the class diagram of the new classes and the existing classes that are also involved. Figure 4 shows the old behavior of the system and the behavior with the addition of our classes. The After diagram only shows the new behavior starting with the `AbstractCWizard`.

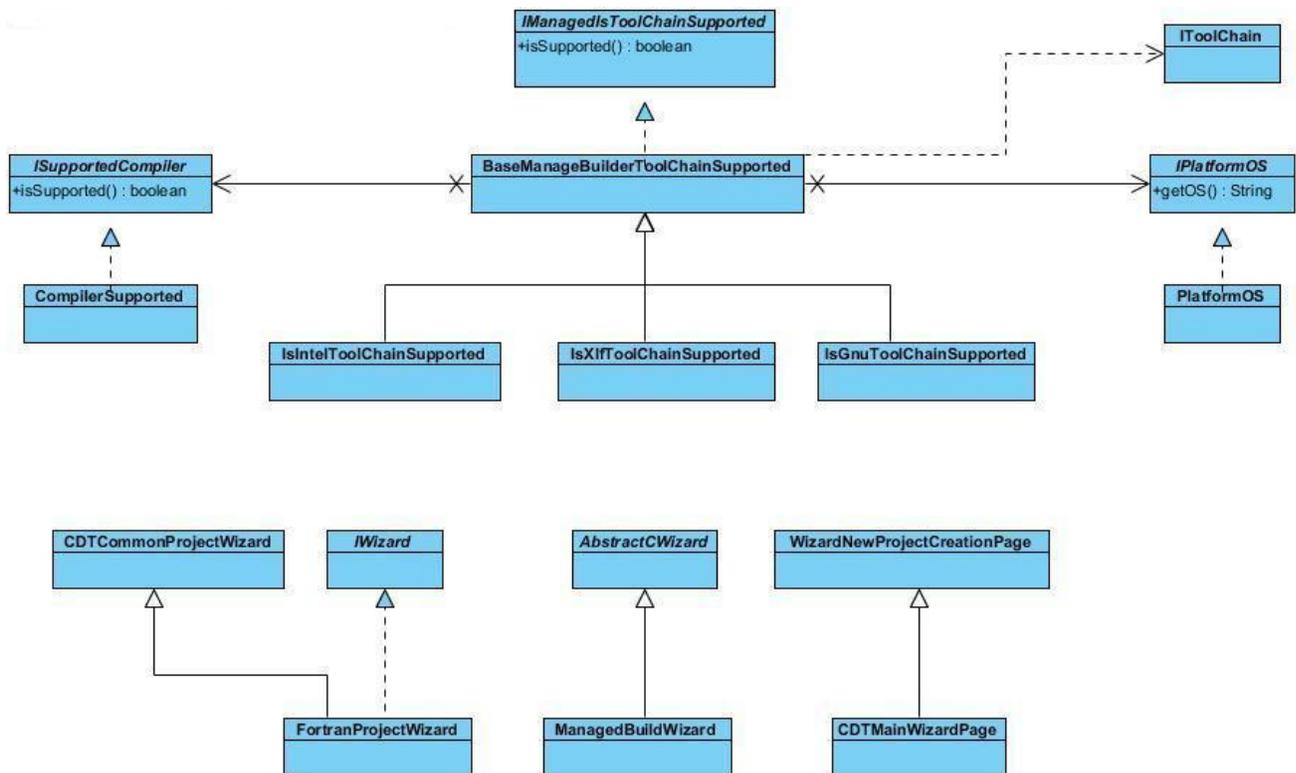
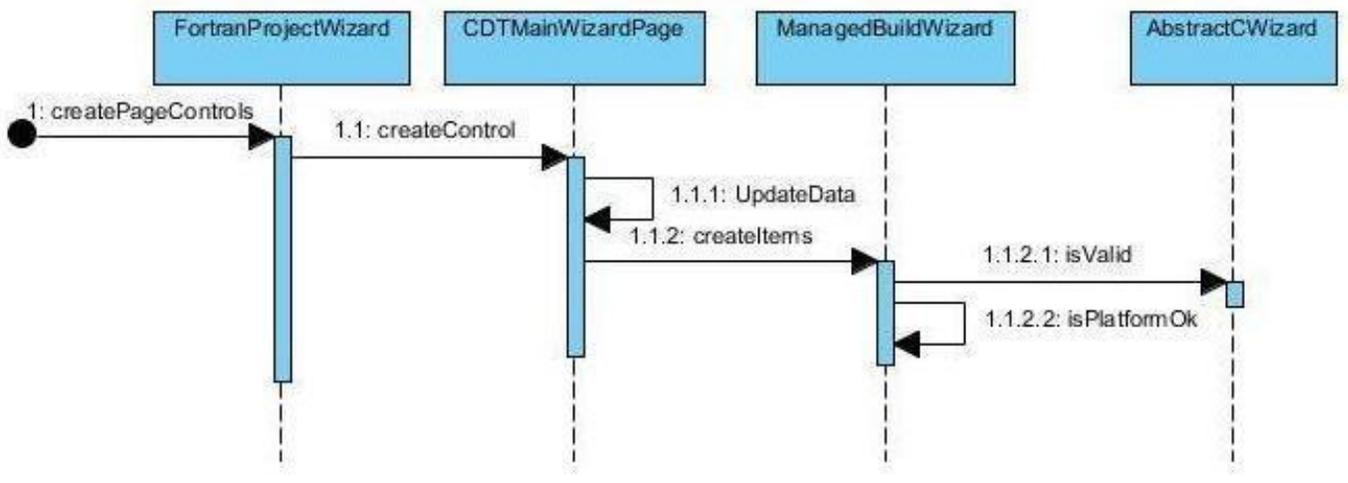


Figure 3

# Before



# After

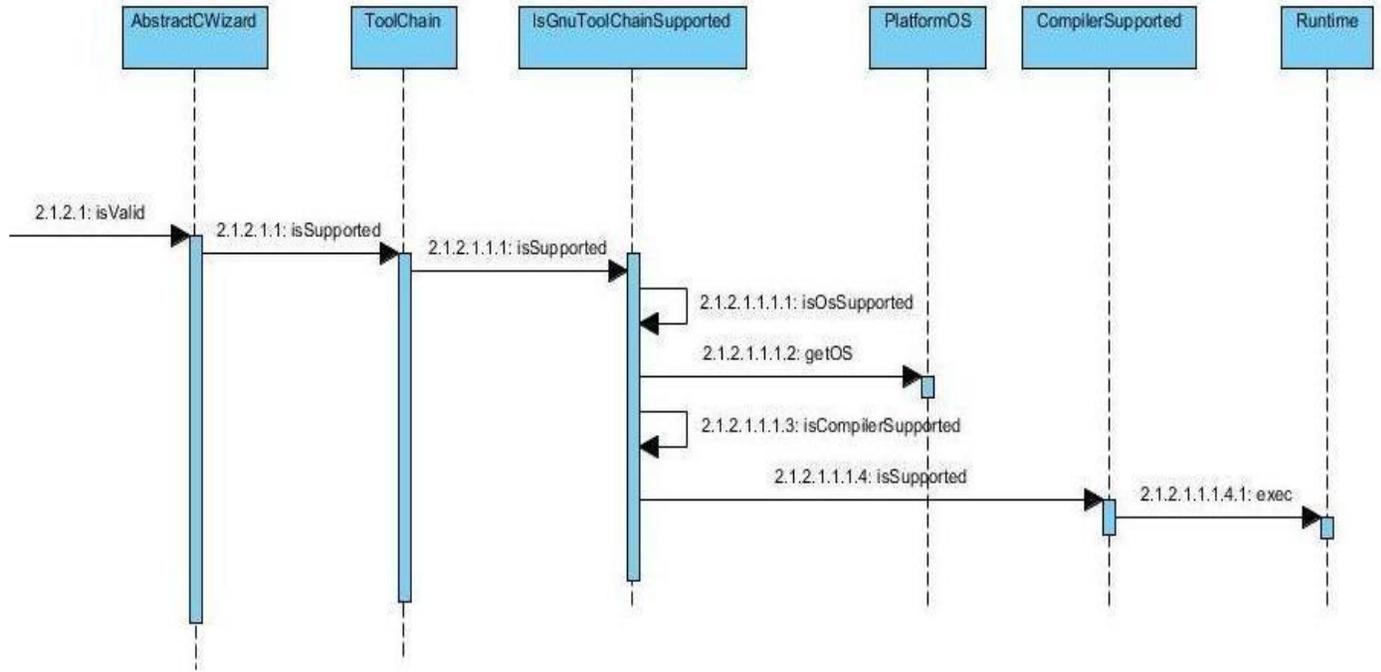


Figure 4

## Future Plans

There is a potential performance issue with how our implementation uses the VPG within the Managed Build System. The VPG is updated for every dependency, which is unnecessary. While no impact to the user experience was noticed, it would be advisable to measure the difference and try to reduce the number of calls to update the VPG.

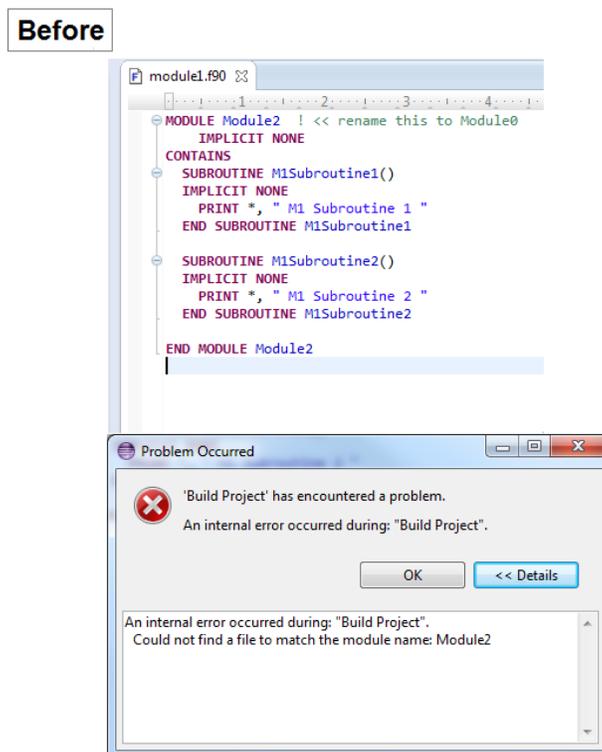
We discovered a bug in which the Managed Build System does not perform a make clean on the Windows platform because it tries to use a UNIX command to do so. This was not directly related to our project and therefore was not addressed at this time, but this bug should be fixed.

We have developed a good solution to the deficiencies presented to us and are interested in submitting them for inclusion in the next Photran release.

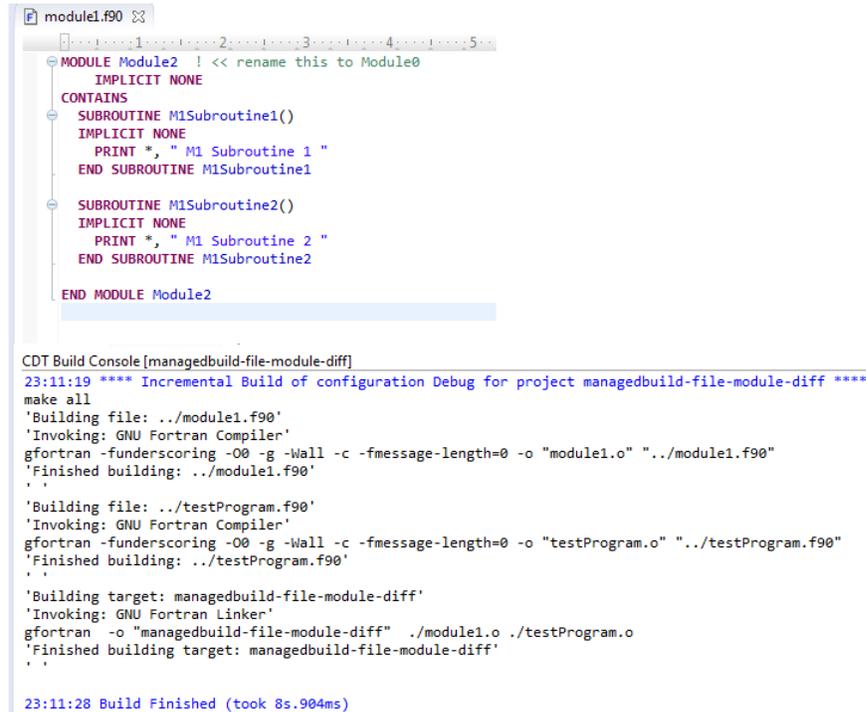
## User manual

The purpose of this manual is to provide technical information for FORTRAN developers how create complex project and use Managed Build System of Photran.

As it is depicted in figure 5, developers could not build FORTRAN files containing more than one module or if its name does not match the module name before completion of this project, however figure 6 shows this problem is resolved. This behavior is added to the Photran by default and there is no specific action required for the developers to use this feature.



After



The image shows a code editor window titled 'module1.f90' with a scroll bar on the right. The code is as follows:

```
MODULE Module2 ! << rename this to Module0
  IMPLICIT NONE
  CONTAINS
  SUBROUTINE M1Subroutine1()
    IMPLICIT NONE
    PRINT *, " M1 Subroutine 1 "
  END SUBROUTINE M1Subroutine1

  SUBROUTINE M1Subroutine2()
    IMPLICIT NONE
    PRINT *, " M1 Subroutine 2 "
  END SUBROUTINE M1Subroutine2

END MODULE Module2
```

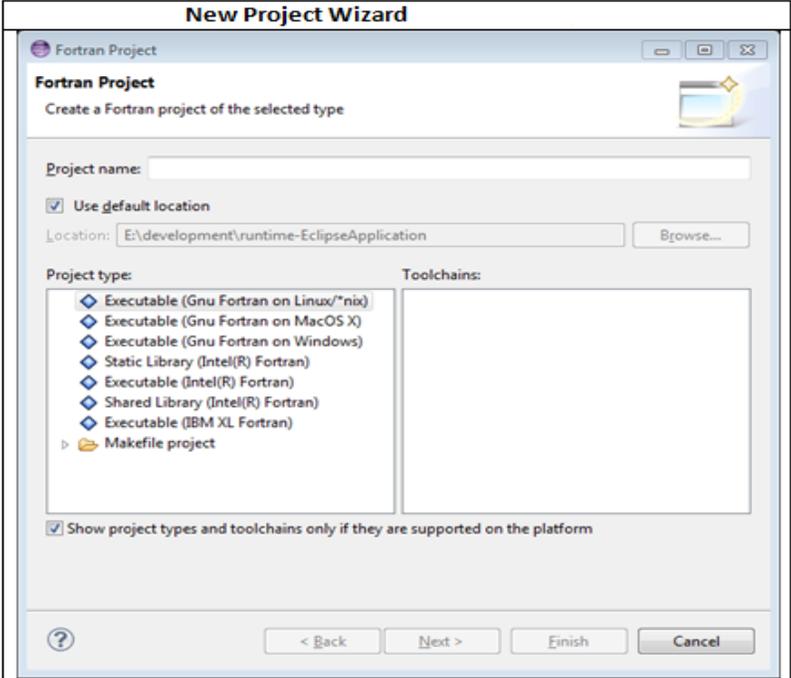
Below the code is a CDT Build Console window titled 'managedbuild-file-module-diff'. The output is as follows:

```
23:11:19 **** Incremental Build of configuration Debug for project managedbuild-file-module-diff ****
make all
'Building file: ../module1.f90'
'Invoking: GNU Fortran Compiler'
gfortran -funderscoring -O0 -g -Wall -c -fmessage-length=0 -o "module1.o" "../module1.f90"
'Finished building: ../module1.f90'
'
'Building file: ../testProgram.f90'
'Invoking: GNU Fortran Compiler'
gfortran -funderscoring -O0 -g -Wall -c -fmessage-length=0 -o "testProgram.o" "../testProgram.f90"
'Finished building: ../testProgram.f90'
'
'Building target: managedbuild-file-module-diff'
'Invoking: GNU Fortran Linker'
gfortran -o "managedbuild-file-module-diff" ../module1.o ../testProgram.o
'Finished building target: managedbuild-file-module-diff'
'
23:11:28 Build Finished (took 8s.904ms)
```

Figure 5

New project wizard that makes a user-friendly wizard for developers and only shows supported compilers in the project type as it is shown in figure 6.

**Before**



**After**

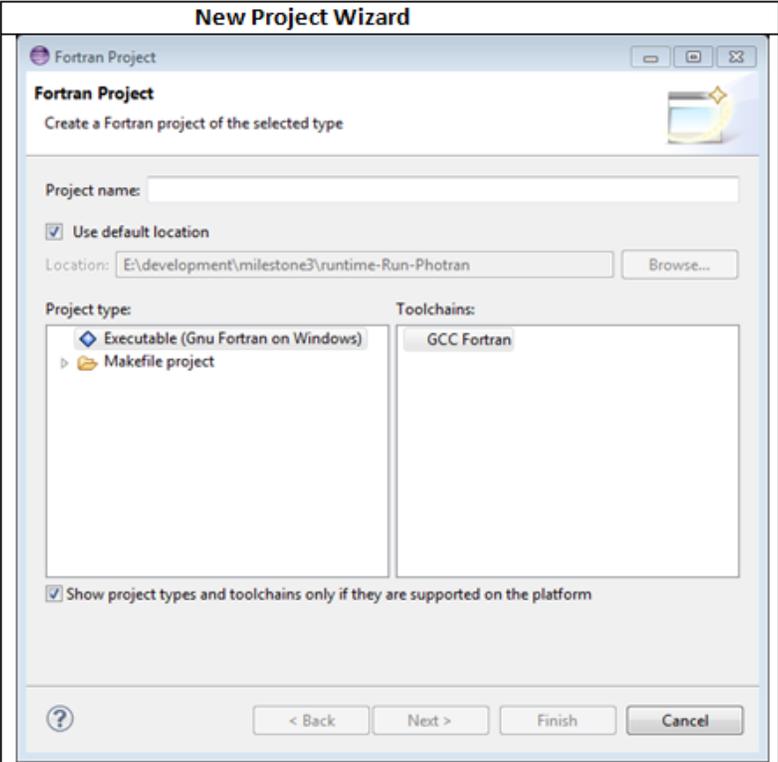


Figure 6

# Appendix

## *Installation Procedures*

Following installation steps assume you have Eclipse 4.2 Juno for RCP/Plug-in development installed already installed, if that's not the case we strongly recommend to install the right version for your platform using the following [link](#). Similarly install the appropriate JRE for your platform.

## *Prerequisites*

### ***Subversion Control***

After you have installed the right flavor of eclipse on your machine, make sure you have a Subversion control plugin installed too. The staff of CS 427 recommends [Subversive](#) along with the appropriate Subversive connector, at the time this document is being written the recommendation is 'Subversive - SVN Team Provider' and the 'SVN Kit 1.7.5' as connector.

### ***CDT SDK***

Photran reuses parts of CDT so you will need to install the C/C++ Development Tools Software Development Kit ([CDT SDK](#)).

### **Photran 8.0**

Uchuva Managed Build Project was designed, developed and tested and using Photran 8.0. We recommend installing the latest stable version of Photran 8.0.

General installation instructions for Photran 8.0 and CDT can be found [here](#). After you successfully installed Photran and CDT and before proceeding to the next section make sure you are able to create Photran project.

### ***Photran baseline API.***

Uchuva Managed build modified Photran 8.0 API baseline and due to the strict requirements of Eclipse when modifying API it's recommended to define an API baseline so changes can be compared. CS 427 staff was kindly enough to provide detailed instructions of how to do that, please follow the instructions for Step 1 part 2 described [here](#).

### ***Check out Uchuva Managed build source code***

To check out Uchuva Managed build source code please follow the instructions described [here](#) for Step 1 part 3, except that on item 5 replace 'https://subversion.ews.illinois.edu/svn/fa12-cs427/\_shared/photran/trunk' with '<https://subversion.ews.illinois.edu/svn/fa12-cs427/projects/G13/trunk>'.

Once you reach this point proceed with instructions for Step 1 part 4 and Step 2 described [here](#) to ensure Uchuva Managed build compiles successfully.

If you reach this point please refer to Running the System section to run Managed build.

## Running the System

To run the system, follow below steps:

- With Eclipse running in the Java perspective, go to the package explorer and right-click one of the Photran plugin packages, e.g., `org.eclipse.photran.core`.
- Select Run As > Eclipse Application. Selection should look like below figure.

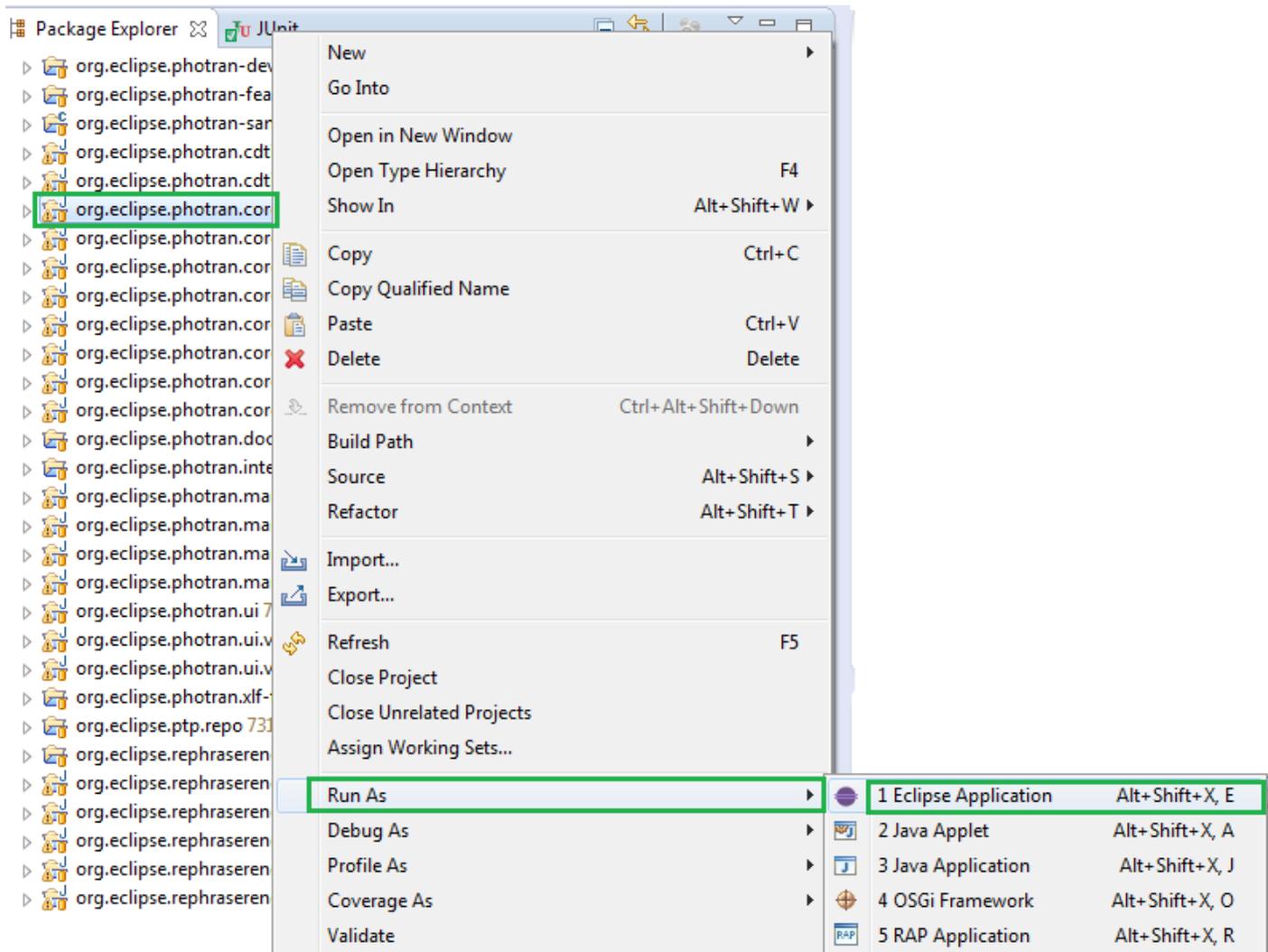


Figure 7

- In the Photran instance, select the Fortran Perspective from Window > Open Perspective > Other. See below figure for reference.

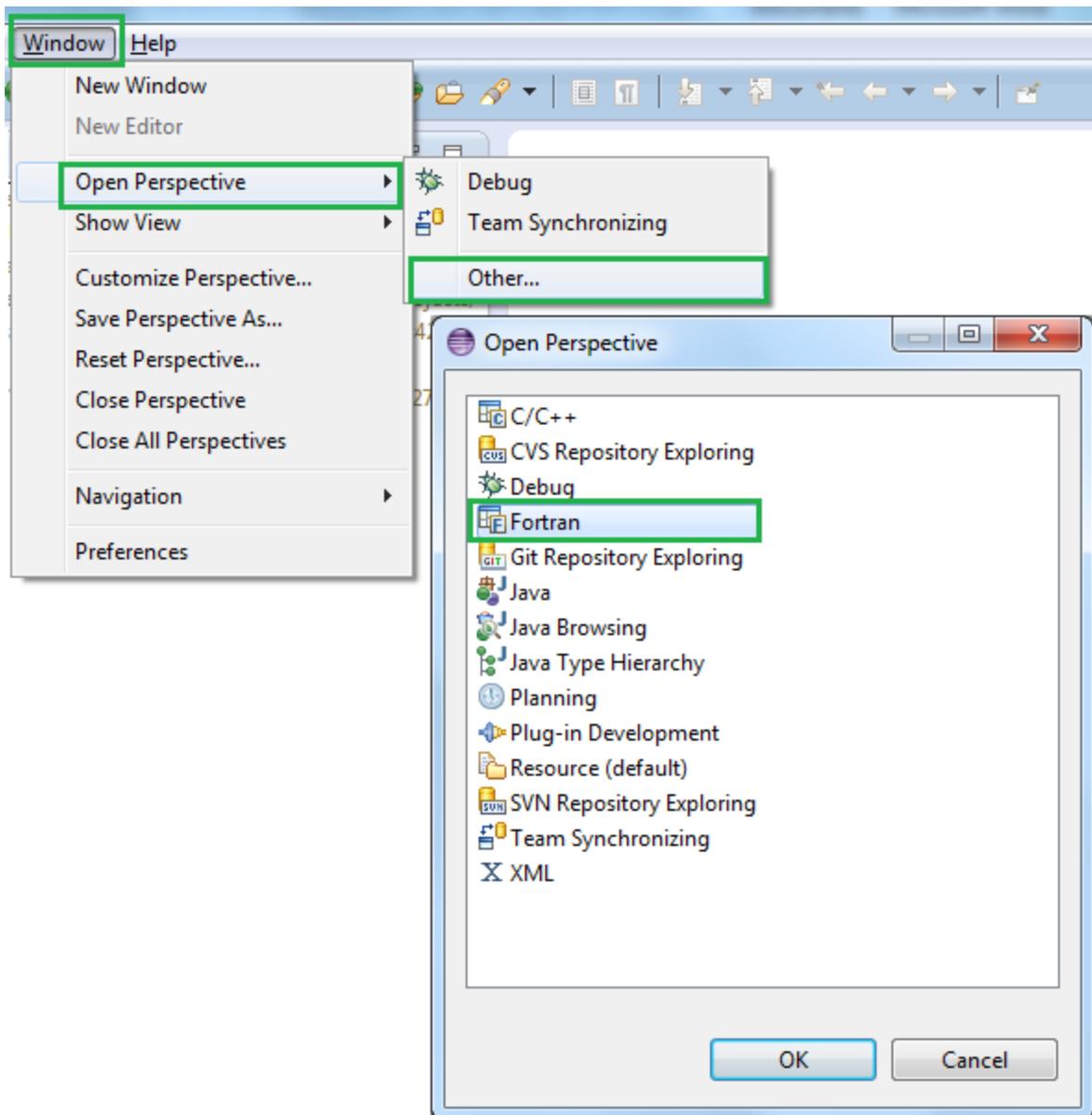


Figure 8

## Running the Unit Tests

All tests for managed build & new project wizard are automated. No prior tests were found either on managed build or project wizard, hence all tests under package `org.eclipse.photran.internal.tests.managed.build` are new. To run tests for this managed build project follow below steps:

1. Get latest tests with source code located in repository at [https://subversion.ews.illinois.edu/svn/fa12-cs427/\\_projects/G13/trunk/](https://subversion.ews.illinois.edu/svn/fa12-cs427/_projects/G13/trunk/)
2. Go to project `org.eclipse.photran.core.vpg.tests`

- Right click package `org.eclipse.photran.internal.tests.managed.build` > Select “Run As” > Select “JUnit Plug-In Test”. Below snapshot shows required selection (highlighted in green) to run all tests.

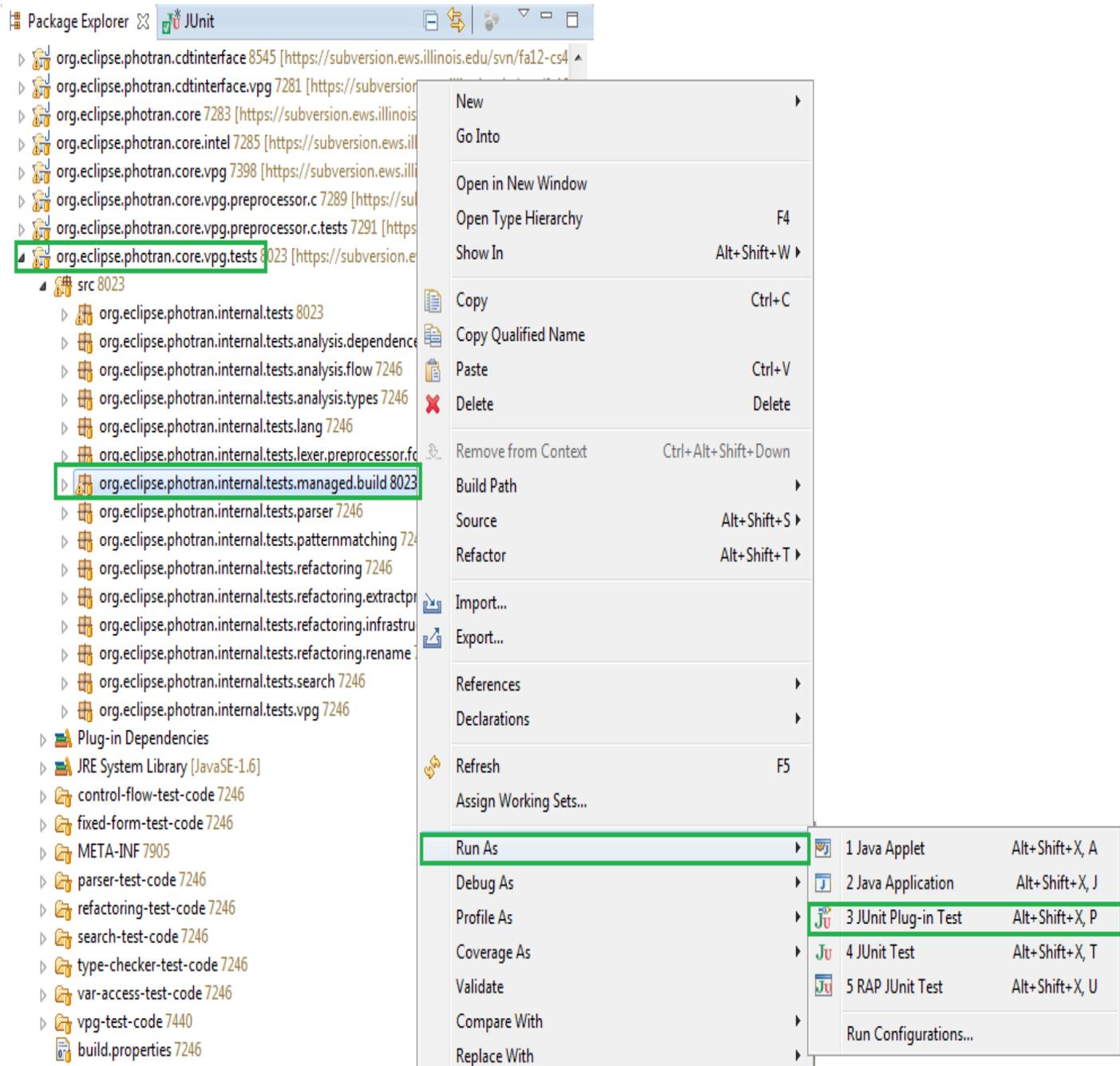


Figure 9

- After running tests, all tests should pass as shown below in JUnit.

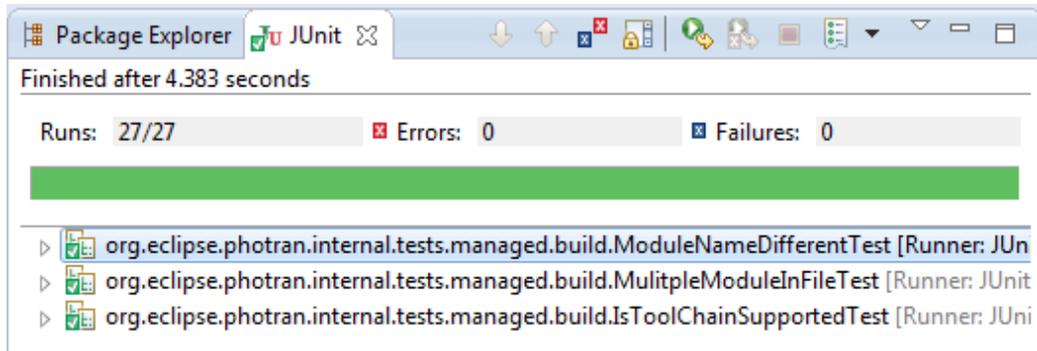


Figure 10

Test suite was designed based on test cases located [here](#).