

# Removal of Obsolete Language Features

Photran Refactoring

AvocadoChestnut Development Team

## Table of Contents

<b>Abstract</b> .....	<b>3</b>
<b>Overview</b> .....	<b>3</b>
<b>Supported Refactorings</b> .....	<b>4</b>
Remove-Branch-To-End-If .....	4
<i>Example 1-1 (Existing CONTINUE)</i> .....	4
<i>Example 1-2 (Inserted CONTINUE)</i> .....	4
<i>Example 1-3 (Mixed Inner/Outer GOTO Statements)</i> .....	5
Remove-Real-And-Double-Precision-Do-Loop-Counters .....	7
<i>Example 2-1 (Explicit Step Size – DO Loop)</i> .....	7
<i>Example 2-2 (Implicit Step Size – DO WHILE Loop)</i> .....	8
Remove-Pause-Statement.....	10
<i>Example 3-1 (PAUSE Replacement)</i> .....	10
<b>Future Plans</b> .....	<b>12</b>
Personal Project Reflections .....	12
<b>Appendix</b> .....	<b>13</b>
Installation Guide.....	13
Refactoring Test Suites.....	14
Technical References .....	15
How To Run Tests.....	15
Development Team .....	16
Open Issues.....	16
Release History .....	16

## List of Figures

Figure 1 – Example: Remove-Branch-To-End-If (Existing CONTINUE).....	4
Figure 2 – Example: Remove-Branch-To-End-If (Inserted CONTINUE) .....	5
Figure 3 – Example: Remove-Branch-To-End-If (Mixed Inner/Outer GOTO Statement) .....	5
Figure 4 – Remove-Branch-To-End-If (Photran GUI) .....	6
Figure 6 – Example: Remove-Do-Loop (Explicit Step Size – DO Loop) .....	8
Figure 7 – Example: Remove-Do-Loop (Implicit Step Size – DO WHILE Loop).....	8
Figure 8 – Remove-Do-Loop Refactoring (Photran GUI).....	8
Figure 8 – Remove-Do-Loop (DO / DO WHILE Selection) .....	9
Figure 9 – Example: Remove-Pause-Statement (PAUSE Replacement) .....	10
Figure 10 - Remove-Pause-Statement Refactoring (Photran GUI) .....	11
Figure 11 – Refactoring Test Suite Run Configuration.....	15
Figure 12 – Remove-Branch Configuration.....	16

## List of Tables

Table 1 - Remove-Branch-To-End-If Method Description .....	7
Table 2 - Remove-DO-Loop Method Description.....	10
Table 3 - Remove-Pause-Statement Method Description .....	12

## Abstract

Photran is a Fortran Integrated Development Environment (IDE) that provides Fortran developer's tools to write, build, run, and debug their programs. An important feature of the Photran tool is the ability to refactor code. Refactoring is the process of changing a programs structure without modifying its behavior. Photran already supports a number of refactorings that allow the user to quickly modify Fortran programs in an automated way.

The Removal of Obsolete Language Features (ROLF) project is an extension to the Photran refactoring library providing a GUI component from which to execute the refactoring and an automated test suite for each refactoring implemented with JUnit.

## Overview

The purpose of this document is to communicate to Photran users about the ROLF refactoring options added to Photran as part of this project. This set of options allows the user to replace Fortran obsolete language features in the code with newer ones that are more accepted as proper coding constructs in later versions of the Fortran language. The refactoring features supported are:

- Removing Branches to END IF Statements
- Removing Real and Double Precision DO Loop Counters
- Removing PAUSE Statements

Each refactoring feature was implemented as an Editor-Based Refactoring, which means the user must specifically select specific parts of the Fortran code in the GUI in order to indicate which section should be refactored. The following is a general sequence of events for the user to perform a refactoring:

1. Select any part of the statement to be refactored (rules specified for each refactoring).
2. Right click the selection.
3. Select Refactoring-><ROLF of choice>.
4. The code selection is used to verify it contains part of the code needed to perform the refactoring.
5. If the selection is valid, the user can preview the new code, cancel the refactoring, or perform the refactoring, else a warning/error is given.

Photran represents elements of a Fortran program via an Abstract Syntax Tree (AST). The ROLF refactorings make use of this program representation to search, add, and remove items of that AST, thus changing the source code of that program. Through Photran's use of the Visitor Pattern it is easy for refactorings such as ROLF to search the program tree in a structured way for a specific AST node. The reader is referred to the Photran User Guides linked to in the Technical References section of this report, which provides greater detail about the architecture and implementation of Photran.

For each of the refactoring options the document will go over a brief description of what the refactoring does, how to perform the refactoring, and how it was implemented. For more details on the implementation, see the documentation provided within the code. Then the future plans for the refactorings will be discussed. Finally, there will be an Appendix, which list the of files changed for each refactoring option, gives details on how to get started with installing the ROLF options, gives details on how to run the automated tests using JUnit and what is tested, list references used, and lists the development team.

**Note:** It is assumed that the reader/user has some familiarity with Photran for Fortran program development. Therefore details such as how to launch Photran are not described and the reader is referred to the Photran User's Guides in the Technical reference section below which already provide a good explanation of how to run Photran. Our descriptions refer strictly to how to run our install and run our refactorings in Photran.

## Supported Refactorings

The following is a detailed discussion of each of the three refactorings that are implemented as part of the Removal of Obsolete Language Features project.

### Remove-Branch-To-End-If

The **Remove-Branch-To-End-If** refactoring option removes the branch to END IF statements. The GOTO statements carry out branching. Branching to END IF is replaced with branching to CONTINUE statement that immediately follows the END IF statement. Example 1-1 below shows the result of the refactoring for this scenario.

#### Example 1-1 (Existing CONTINUE)

```
PROGRAM RemoveBranchEX1_1
  INTEGER :: sum, i
  sum = 0
  DO 20, i = 1, 10
    IF (MOD(i,2).eq.0) THEN
      GOTO 10
    END IF
    sum = sum + i
    IF (sum.ge.100) THEN
      sum = sum + sum
10  END IF
20 CONTINUE
  PRINT *, 'sum:', sum
END PROGRAM RemoveBranchEX1_1
```

```
PROGRAM RemoveBranchEX1_1
  INTEGER :: sum, i
  sum = 0
  DO 20, i = 1, 10
    IF (MOD(i,2).eq.0) THEN
      GOTO 20
    END IF
    sum = sum + i
    IF (sum.ge.100) THEN
      sum = sum + sum
    END IF
20 CONTINUE
  PRINT *, 'sum:', sum
END PROGRAM RemoveBranchEX1_1
```

Before

After

Figure 1 – Example: Remove-Branch-To-End-If (Existing CONTINUE)

If there is no CONTINUE statement following an END IF statement that is a target of a branch, the refactoring inserts a CONTINUE statement immediately after this END IF statement. The label for the continue statement can be one of two scenarios:

1. If there is a GOTO statement within the selected IF block, the END IF label is removed, used for the new CONTINUE statement.
2. Else the END IF label remains and a unique label is generated for the new CONTINUE statement. Then for all the labels of the GOTO statements outside of the selected if block are renamed to the new label.

Example 1-2 below shows the result of a reused label for the newly inserted CONTINUE statement.

#### Example 1-2 (Inserted CONTINUE)

```
PROGRAM RemoveBranchEX1_2
  INTEGER :: k, i
  READ(*,*) k
```

```
PROGRAM RemoveBranchEX1_2
  INTEGER :: k, i
  READ(*,*) k
```

<pre> IF (k.lt.10) THEN   GOTO 20 END IF i = k - 10 IF (i.gt.100) THEN   i = i - 100 20 END IF PRINT *, i END PROGRAM RemoveBranchEX1_2 </pre>	<pre> IF (k.lt.10) THEN   GOTO 20 END IF i = k - 10 IF (i.gt.100) THEN   i = i - 100 END IF 20 CONTINUE PRINT *, i END PROGRAM RemoveBranchEX1_2 </pre>
--	---

**Before**
**After**  
 Figure 2 – Example: Remove-Branch-To-End-If (Inserted CONTINUE)

Example 1-3 below shows the result of refactoring when there are GOTO statements from inside and outside of the selected IF block.

#### Example 1-3 (Mixed Inner/Outer GOTO Statements)

<pre> PROGRAM RemoveBranchEX1_3   INTEGER :: k, i   READ(*,*) k   IF (k.lt.10) THEN     GOTO 20   END IF   i = k - 10   IF (i.gt.100) THEN     i = i - 100     IF (i.lt.10) THEN       GOTO 20     END IF     i = i - 10 20 END IF PRINT *, i END PROGRAM RemoveBranchEX1_3 </pre>	<pre> PROGRAM RemoveBranchEX1_3   INTEGER :: k, i   READ(*,*) k   IF (k.lt.10) THEN     GOTO 30   END IF   i = k - 10   IF (i.gt.100) THEN     i = i - 100     IF (i.lt.10) THEN       GOTO 20     END IF     i = i - 10 20 END IF 30 CONTINUE PRINT *, i END PROGRAM RemoveBranchEX1_3 </pre>
--	--

**Before**
**After**  
 Figure 3 – Example: Remove-Branch-To-End-If (Mixed Inner/Outer GOTO Statement)

#### User Action

To initiate the refactoring, the user must select a labeled END IF statement in the Photran editor. Then right click the selection and select *Refactor->RemoveBranch To End If* option as shown in Figure 4.

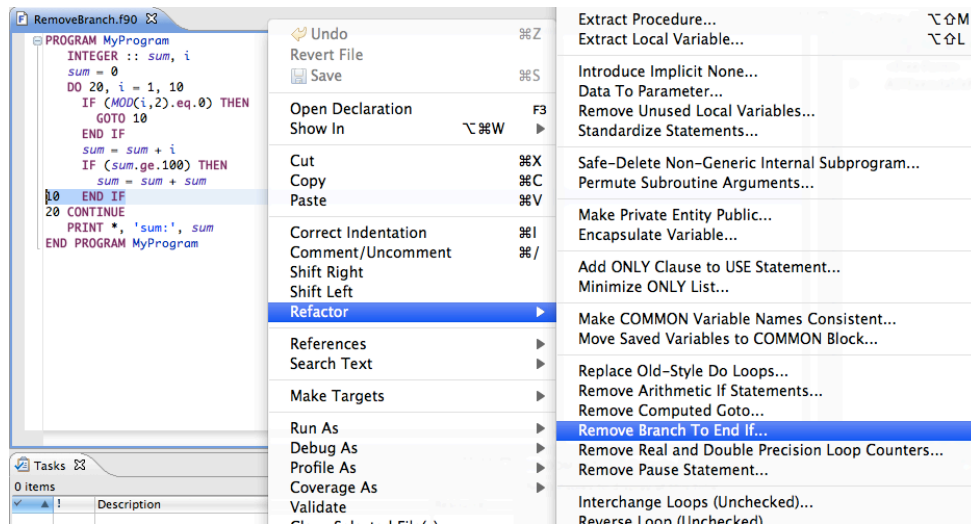


Figure 4 – Remove-Branch-To-End-If (Photran GUI)

### Checked Error Conditions

There are conditions that are checked to verify that the refactoring can be applied to the user selection. Failure to meet any of these conditions will halt the refactoring and the user will be notified via appropriate GUI messages. (This is equivalent to a *fail-initial* in Photran refactoring test environment.)

For the **Remove-Branch-To-End-If** refactoring those conditions are:

- User must select a portion of an **END IF** statement with a labeled.
- Selected **END IF** statement must be part of an **IF** block.
- There must be at least one **GOTO** statement outside of the selected **IF** block.

### Class/Method Description

The `RemoveBranchToEndIfRefactoring` class, which extends the `FortranEditorRefactoring` class, is the main code that implements the Remove-Branch-To-End-If refactorings. It can be found in the `org.eclipse.photran.core.vpg` package in the `src/org.eclipse.photran.internal.core.refactoring` directory. The following is a brief description of the methods and functions in the class.

Method Name	Description
General Photran Refactoring Methods (Overridden)	
<code>getName()</code>	Provides the refactoring name in the Photran refactoring GUI menu.
<code>doCheckInitialConditions()</code>	Verifies that refactoring is enabled ( <code>ensureProjectHasRefactoringEnabled()</code> ) and that the selection to be refactored is valid as described in the <i>Checked Error Conditions</i> section above.
<code>doCheckFinalConditions()</code>	No final conditions (no user input).
<code>doCreateChange()</code>	Called after the conditions pass. Refactoring is delegated to <code>changeGotoLabelToContinueLabel()</code> or <code>changeNoContinueAfterEndIf()</code> depending if a <b>CONTINUE</b> statement exists after the selected <b>END IF</b> block.
Main Refactoring Methods	
<code>changeGotoLabelToC</code>	For the <b>GOTO</b> statements targeting the selected <b>END IF</b> that are found

continueLabel ( )	outside of the selected IF block, change the labels to the label of the CONTINUE immediately after the selected END IF. If there are no GOTO statements inside the selected IF block, the label of the selected END IF would be removed.
changeNoContinueAfterEndIf ( )	Adds a CONTINUE statement after the selected END IF statement. If there are no GOTO statements within the selected IF block, move the label of the selected END IF to the new CONTINUE statement, else create a new unique label.
Support Methods	
getGotoNodes ( )	Build a list of GOTO nodes within a desired node.
getGotoStmtsInAllProperLoopConstructs ( )	Build a list of GOTO nodes within all proper-loop-construct nodes within a desired node.
findGotoForLabel ( )	Build a list of GOTO nodes with a specified label within a desired node.
continueAfterIfStmt ( )	Find the CONTINUE statement immediately after an if construct node.
getUniqueLabel ( )	Generate a unique label based on the labels of action statements passed in by finding the largest integer label and then adding 10.
getActionStmts ( )	Build a list of Action statement nodes within a desired node.
getActionStmtsInAllProperLoopConstruct ( )	Build a list of Action statement nodes within all proper-loop-construct nodes within a desired node.
getProperLoopConstructs ( )	Build a list of proper loop construct nodes within a desired node.
Support Methods Shared By Other Refactorings	
findEnclosingNodeOfType ( )	Find a node of specified type that encloses a given node.

Table 1 - Remove-Branch-To-End-If Method Description

## Remove-Real-And-Double-Precision-Do-Loop-Counters

The **Remove-Real-And-Double-Precision-Loop-Counters** targets to transform a DO loop with control to a DO loop without control or DO WHILE loop base on user selection. A qualified control DO loop for this refactoring is when the counter is a real or double precision type

The refactoring supports both cases where the step-size is explicit or implicit. When the step size is implicit, a value of +1 or -1 is used (based on comparison of lower and upper loop bounds).

Example 2-1 shows the result of a refactoring performed on an explicit DO loop with real precision counter and example 2-1 shows the result of a refactoring performed on implicit DO loop with real precision counter.

### Example 2-1 (Explicit Step Size – DO Loop)

```
PROGRAM RemoveDoLoopEX2_1
  REAL :: counter, sum
  sum = 0.0
  DO counter = 1.2, 1.8, 0.1
    sum = sum + counter
  END DO
  PRINT *, sum
END PROGRAM RemoveDoLoopEX2_1
```

```
PROGRAM RemoveDoLoopEX2_1
  REAL :: counter, sum
  sum = 0.0
  counter = 1.2
  DO
    sum = sum + counter
    counter = counter + 0.1
    IF(counter > 1.8) THEN
      EXIT
    END IF
```

```

END DO
PRINT *, sum
END PROGRAM RemoveDoLoopEX2_1

```

Before

After

Figure 5 – Example: Remove-Do-Loop (Explicit Step Size – DO Loop)

### Example 2-2 (Implicit Step Size – DO WHILE Loop)

```

PROGRAM RemoveDoLoopEX2_2
REAL :: counter, sum
sum = 0.0
DO counter = 1.8, 1.2
    sum = sum + counter
END DO
PRINT *, sum
END PROGRAM RemoveDoLoopEX2

```

```

PROGRAM RemoveDoLoopEX2_2
REAL :: counter, sum
sum = 0.0
counter = 1.8
DO WHILE (counter >= 1.2)
    sum = sum + counter
    counter = counter - 1
END DO
PRINT *, sum
END PROGRAM RemoveDoLoopEX2_2

```

Before

After

Figure 6 – Example: Remove-Do-Loop (Implicit Step Size – DO WHILE Loop)

### User Action

To initiate the refactoring the user must select a qualified DO loop statement in the Photran editor. Then right click the selection and select *Refactor->Remove Real and Double Precision Loop Counters* option as shown in Figure 7.

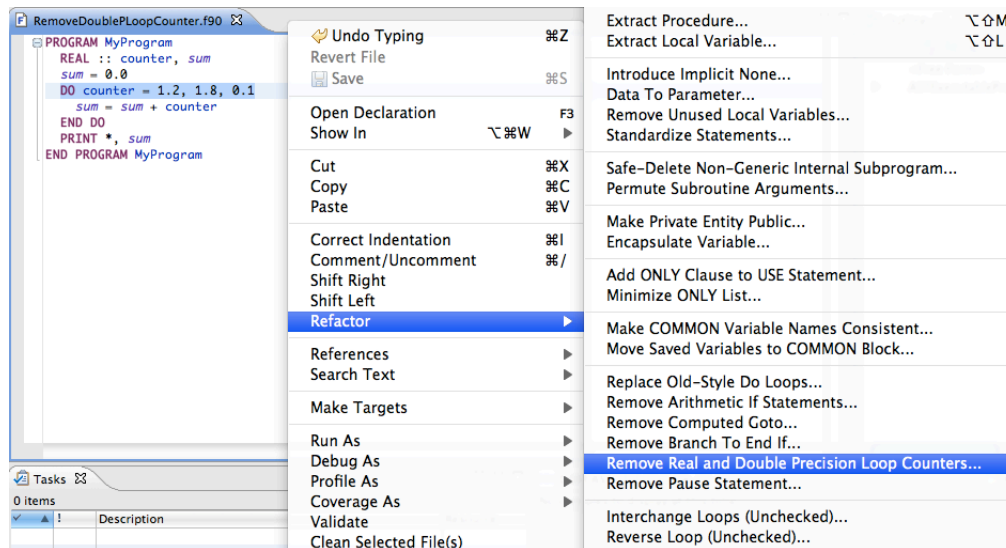


Figure 7 – Remove-Do-Loop Refactoring (Photran GUI)

After selecting the refactoring, the user will be prompted to select if the DO loop should be converted to a DO or DO WHILE loop as shown in Figure 8 below.

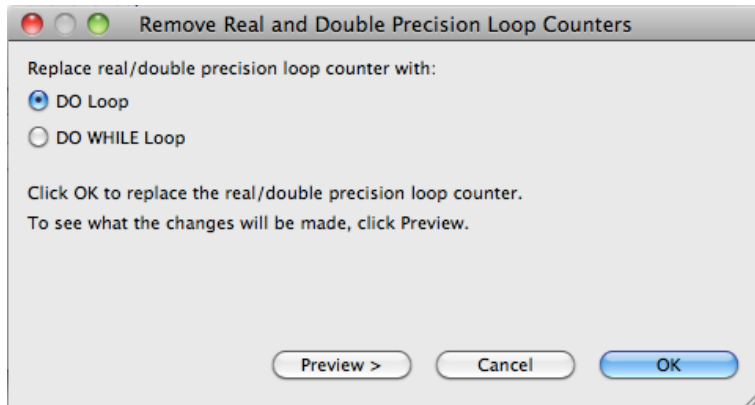


Figure 8 – Remove-Do-Loop (DO / DO WHILE Selection)

### Checked Error Conditions

There are conditions that are checked to verify that the refactoring can be applied to the user selection. Failure to meet any of these conditions will halt the refactoring and the user will be notified via appropriate GUI messages. (This is equivalent to a *fail-final* in Photran refactoring test environment.)

For the **Remove-Real-And-Double-Precision-Loop-Counters** refactoring those conditions are:

- User must select a portion of a controlled DO loop statement.
- Both loop control variables (loop sum and loop index counter) must only be declared as REAL or DOUBLE PRECISION types.

### Class/Method Description

The `RemoveRealAndDoublePrecisionLoopCountersRefactoring` class that extends the `FortranEditorRefactoring` class is the main code that implements the refactorings. It can be found in the `org.eclipse.photran.core.vpg` package in the `src/org.eclipse.photran.internal.core.refactoring` directory. The following is a list and brief description of the different methods in the class.

Method Name	Description
General Photran Refactoring Methods (Overridden)	
<code>getName()</code>	Provides the refactoring name in the Photran refactoring GUI menu.
<code>doCheckInitialConditions()</code>	Verifies that refactoring is enabled ( <code>ensureProjectHasRefactoringEnabled()</code> )
<code>doCheckFinalConditions()</code>	Verifies that the selection to be refactored is valid as described in the <i>Checked Error Conditions</i> section above – after user input.
<code>doCreateChange()</code>	Performs the refactoring by building a number of AST node elements and inserting them in to the program to do the work that the controlled DO loop would have done. This includes: 1) Initial loop control variable assignment (starting value). 2) Increment/Decrement control variable by specified amount (implicit or explicit step size) 3) Check inside DO loop to see if loop variable limit specified is exceeded. Most of the work is delegated to the support methods.
Support Methods	
<code>setShouldReplaceWithDoWhileLoop()</code>	Set method for DO or DO WHILE refactoring selection variable.
<code>insertNewDoLoop()</code>	With provided strings this method builds the new AST nodes for the

	initial variable assignment, increment/decrement statement, and IF DO loop break statement. They are then inserted in to the proper hierarchy of the program AST structure.
insertNewDoWhileLoop()	With provided strings this method builds the new AST nodes for the initial variable assignment, increment/decrement statement, and DO WHILE loop check statement. They are then inserted in to the proper hierarchy of the program AST structure.
insertInitialCounterAssignment()	Creates an assignment statement node for the initial loop counter value assignment and inserts it before the selected DO loop of the refactoring.
insertCounterAssignment()	Creates an assignment statement node for the loop counter variable assignment and inserts it as the last statement in the selected DO loop body.
getTypeDeclarations()	Get all variable type declarations in the program.
Support Methods Shared By Other Refactorings	
findEnclosingNodeOfType()	Find a node of specified type that encloses a given node.

Table 2 - Remove-DO-Loop Method Description

## Remove-Pause-Statement

The **Remove-Pause-Statement** refactoring addresses the replacement of the PAUSE statement with a PRINT and READ statement. Execution of a PAUSE statement may be different on different platforms. The refactoring assumes the most basic functionality: it replaces the PAUSE statement with a PRINT statement that displays the message of the PAUSE statement, immediately followed by a READ statement that waits for any input from the user. Example 3-1 below shows an example of this refactoring.

### Example 3-1 (PAUSE Replacement)

```

PROGRAM RemovePauseEX3_1
  INTEGER :: i
  DO i = 1, 100
    IF (i == 50) THEN
      PAUSE 'mid job'
    END IF
  END DO
  PRINT *, 'i=', i
END PROGRAM RemovePauseEX3_1

```

```

PROGRAM RemovePauseEX3_1
  INTEGER :: i
  DO i = 1, 100
    IF (i == 50) THEN
      PRINT *, 'mid job'
      READ (*, *)
    END IF
  END DO
  PRINT *, 'i=', i
END PROGRAM RemovePauseEX3_1

```

Before

After

Figure 9 – Example: Remove-Pause-Statement (PAUSE Replacement)

## User Action

To initiate the refactoring the user must select a PAUSE statement the Photran editor. Figure 10 below shows in the Photran editor an example of what should be selected in a Fortran program after right clicking on the selection in Photran (**Refactor->Remove Pause Statement**).

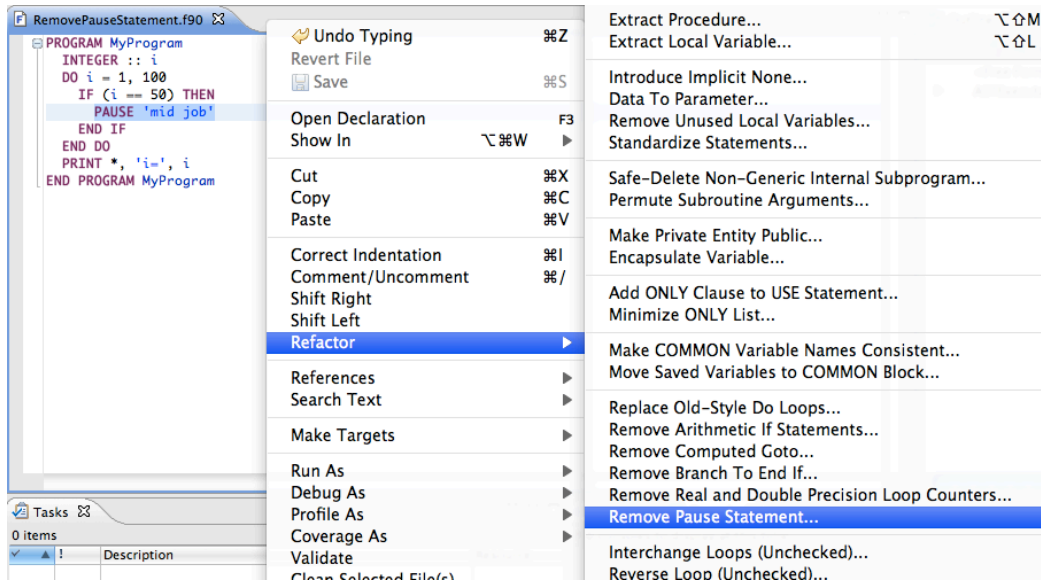


Figure 10 - Remove-Pause-Statement Refactoring (Photran GUI)

### Checked Error Conditions

There is one condition that is checked to verify that the refactoring can be applied to the user selection. Failure to meet any of this condition will halt the refactoring and the user will be notified via appropriate GUI messages. (This is equivalent to a *fail-initial* in Photran refactoring test environment.)

For the **Remove-Pause-Statement** refactoring the condition is:

- User must select a PAUSE statement in the program.

### Class/Method Description

Our `RemovePauseStmtRefactoring` class that extends the `FortranEditorRefactoring` class is the main code that implements the refactoring. It can be found in the `org.eclipse.photran.core.vpg` package in the `src/org.eclipse.photran.internal.core.refactoring` directory. The following is a list and brief description of the different methods in the `RemovePauseStmtRefactoring` class.

Method Name	Description
General Photran Refactoring Methods (Overridden)	
<code>getName()</code>	Provides the refactoring name in the Photran refactoring GUI menu.
<code>doCheckInitialConditions()</code>	Verifies that refactoring is enabled ( <code>ensureProjectHasRefactoringEnabled()</code> ) and that the selection to be refactored is valid as described in the <i>Checked Error Conditions</i> section above.
<code>doCheckFinalConditions()</code>	No final conditions (no user input).
<code>doCreateChange()</code>	Called after initial/final conditions are checked and passed. Refactoring is delegated to <code>changePauseStmt()</code> .
Main Refactoring Methods	
<code>changePauseStmt()</code>	Performs the refactoring making modifications to the transient AST.
Support Methods Shared By Other Refactorings	
<code>findEnclosingNode</code>	Find a node of specified type that encloses a given node.

OfType ( )	
------------	--

Table 3 - Remove-Pause-Statement Method Description

## Future Plans

Our refactorings are implemented as an Editor-Based refactorings. A potentially more useful version of this refactoring would be a Resource-Based refactoring that could operate on whole files or sets of files. (See Specialized Photran Developers guide for further details about each refactoring type.) Both refactoring types are useful so rather than converting the refactoring we could leverage these refactorings to a Resource-Based refactoring so it could be used either way. We will need to consult with our customer and Photran representatives if such a refactoring would be more useful in a real Photran release. We are in the process of contacting Jeff Overbey ([jeffrey.l@over.bz](mailto:jeffrey.l@over.bz)) and we submitting our code as part of a Bugzilla enhancement request for Photran. Hopefully this project will be accepted as new refactoring features in the next release of Photran.

## Personal Project Reflections

**Chamil:** Working on the Fortran framework which is build on top of Eclipse CDT framework made life a lot easier and helped us move very fast. Most of the time, we studied couple of examples that are already there to accomplish what we needed to do. Even though there was not much room for creative Object Oriented design, I was able to see some applications of design patterns in the Fortran framework. It was good to experience and follow the XP development process. I got to experience the benefits of writing the tests before the code for the first time. Getting use to the XP development process and how to prioritize tasks are the most important lessons I learnt from the project. It was great to work with a team where everyone brought an important set of skills to the table.

**Jerry:** Working in pairs on code implementation proved helpful in the mid to late stages of coding. During the initial ramp up, I found it beneficial to first look at the code as an individual then gather notes, and then perform a second review of the existing code with my partners. Initially, writing test also proved useful in understanding the goals of the customer. Having constant short deadlines, although pressure filled even though the task were broken up correctly into manageable chunks, made the completion of all the tasks feel surmountable much like walking up stairs versus climbing a vertical high wall. Through the project, I can clearly see the pros and cons to pair programming and XP practices.

**Mark:** The ROLF Photran project for our group was a good application of our studies in the CS427 course. Working with such a large program was a bit overwhelming at first, but applying the reverse engineering principles we learned made it a manageable project. After understanding what was needed to implement a new refactoring in to the system (and associated test suite) it is clear that the Photran source code is well architected to allow for the easy addition of new refactoring. For our development the Photran user's guide were very useful but ultimately it was the ability to study other refactorings in the system and their tests that made our implementation easier. Working with a large group remotely and following the XP processes was also an interesting experience. In my opinion releasing our code in structured iterations rather than one final release also led to higher quality code useful functionality to demonstrate to our customer sooner. With additional time it may be useful to convert our refactoring to a Resource-Based refactoring and that seems to be the next logical extension of this project.

**Nicola:** The Photran project provided a good learning experience. I had many challenges with the project including issues with setting up the environment and as a result I had a late start at trying to get everything sorted out. For me, the class was overly challenging since I do not have a solid background in Computer Science, and we were dealing with ASTs and concepts that are taught in advanced computer science courses. The project did however provide me with a solid general understanding of how Java is used to insert nodes in the AST tree for the Fortran code, and it is a great foundation on my path to increase and improve my abilities. I was backed my team members who are very skilled and proficient and that enhanced my overall experience with the project and the course.

**Rita:** The ROLF refactorings project was a good learning experience and rewarding one. It was intimidating at first given a system that we weren't familiar with, but the following was helpful in finishing our project with quality on time: 1) the learning spikes by reading the documentation and code, and stepping through the code of an existing refactoring, 2) many standup (Skype) meetings with the team by sharing our new found knowledge, problems, and ideas, 3) pair programming to work on the code with someone and exchange ideas, 4) having test cases ready to test new/refactored code for confidence before checking in, and 5) lots of trial and error. Working through challenges together and successfully getting the code to work properly were probably the most rewarding for me. Since we were given this system we thought should work almost perfectly, we were surprised several times when we discovered features of the system or trying to follow a refactoring's solution actually didn't work as expected. Some of these areas were using the Reindenter method and searching through DO loops using the Visitor pattern. We were able to find ways to work around these problems by developing understanding of what the AST consists of and how we can manipulate them to perform the refactorings we need. I also find that breaking up the tasks to be completed and reviewed in small iterations is beneficial to building a more reliable and well-tested system. The more features that get into the code at one time, the more likely something will not be tested. Other than Photran installation not being too reliable and the extra time needed to plan for extra documentation and demos, the project was a good learning experience about the XP process and patterns.

## Appendix

### Installation Guide

First, the user should download the latest version of the Photran source code on to their Eclipse system and insure that there are no errors and that the included automated tests pass. The General Photran User's Guide linked below in the Technical References section contains detailed instructions of how download the Photran source code from CVS and run the automated tests in its Appendix A.

Afterwards the user can replace the following 3 core Photran packages with our version of these packages as found on our Subversion site for the final tagged version of our code. The direct link to this tagged version on the AvocadoChestnut SVN site is:

[https://subversion.ews.illinois.edu/svn/fa10-cs427/AvocadoChestnut/RemovalOfObsoleteLanguageFeatures/tags/Final\\_Release\\_1.0](https://subversion.ews.illinois.edu/svn/fa10-cs427/AvocadoChestnut/RemovalOfObsoleteLanguageFeatures/tags/Final_Release_1.0)

The following is a list of the three packages that our SVN release contains. Replacing all three packages, however, could override changes to the Photran source code that occurred since the

ROLF project was started with Photran and CDT 7.0. Therefore, the list below also contains a specific list of the files and directories we added/modified. The user may elect to selectively import these files to receive just the ROLF functionality provided by this project.

- *Org.eclipse.photran.core.vpg* Package – Core refactoring code.
  - *src/org/eclipse/phoran/internal/core/refactoring* Directory
    - *RemoveBranchToEndIfRefactoring.java*
    - *RemovePauseStmtRefactoring.java*
    - *RemoveRealAndDoublePrecisionLoopCountersRefactoring.java*
  - *src/org/eclipse/phoran/internal/core/refactoring/infrastructure* Directory
    - *FortranResourceRefactoring.java*
- *org.eclipse.photran.core.vpg.tests* Package – Refactoring test suite and test cases. (test cases in directories now shown.)
  - *src/org/eclipse/phoran/internal/tests/refactoring* Directory
    - *RemoveBranchToEndIfTestSuite.java*
    - *RemovePauseStmtTestSuite.java*
    - *RemoveRealandDoublePrecisionLoopCountersTestSuite.java*
  - *refactoring-test-code* Directory
    - *remove-branch-to-end-if* Directory
    - *remove-pause-stmt* Directory
    - *remove-real-and-double-precision-loop-counters* Directory
- *org.eclipse.photran.ui.vpg* Package – Contains GUI messages and menu extensions for refactorings.
  - *src/org.eclipse.photran.internal.ui.refactoring* Directory
    - *plugin.xml*
    - *RemoveRealAndDoublePrecisionLoopCounterInputPage.java*
    - *Messages.java*
    - *Message.properties*
    - *RemoveRealAndDoublePrecisionLoopCountersAction.java*

## Refactoring Test Suites

A fully automated JUnit test suite for each refactoring was developed using the Photran refactoring test framework. All refactorings follow the convention of a test directory linked to the refactoring's JUnit test suite. The JUnit test suite for each refactoring can be found in the *org.eclipse.photran.core.vpg.tests* package under the *src/org.eclipse.photran.internal.tests.refactoring* directory. The file name and class name for each JUnit test suite for each of our three ROLF refactorings in that directory are:

- *RemoveBranchToEndIfTestSuite* - *RemoveBranchToEndIfTestSuite.java*
- *RemoveRealAndDoublePrecisionLoopCountersTestSuite* - *RemoveRealAndDoublePrecisionLoopCountersTestSuite.java*
- *RemovePauseStmtTestSuite* - *RemovePauseStmtTestSuite.java*

Under that directory a source file with a *.f90* extension is placed and a expected results file with a *.f90.result* extension is also provided. Through the `!<<<` pragma in the source code the Photran test framework is instructed as to what the user selection would be for that refactoring. (See Photran Specialized User's Guide in the Technical Reference section for more details).

All refactoring test suites for each refactoring can be found in the *org.eclipse.photran.core.vpg.tests* package *refactoring-test-code* directory. For each of the three ROLF refactorings there is a corresponding directory from which test folder and files are placed:

- *remove-branch-to-end-if/*

- *remove-real-and-double-precision-loop-counters/*
- *remove-branch-to-end-if/*

Note: Due limitations in the length of this report we are unable to include a test list/description table for each test we wrote. However, each test has a brief description in the header in Fortran comments of what the test is checking and the expected behavior of the refactoring.

## Technical References

The following is a list of technical references related to Photran and refactoring that the reader may find useful to further understand our refactorings as they relate to the Photran program

- General Photran Developer's Guide (<http://bit.ly/eGfGwP>)
  - o Introduction to Photran architecture, instructions for downloading Photran and building/launching Photran in Eclipse, and how to run Photran's included automated refactoring test suites.
- Specialized Photran Developer's Guide (<http://bit.ly/fyZiqi>)
  - o More detailed description of Photran architecture and implementation (ASTs, VPG) and how Photran implements Fortran refactorings, how to implement a new Fortran refactorings and associated refactoring test suite, and details about the Fortran editor that Photran provides access to.

## How To Run Tests

With the appropriate Photran and ROLF files installed as instructed in the Installation Guide in the previous section the user must setup a run configuration in Eclipse for each refactoring test suite. The description below describes how to setup a run configuration for the **Remove-Branch-To-End-If** refactoring but following the same steps and referencing the test suites listed in the Refactoring Test Suites section above will provide a configuration for each refactoring.

In the Project Explorer navigate to the JUnit test suite directory for all the refactorings, i.e. the *src/org.eclipse.photran.internal.tests.refactoring* directory the *org.eclipse.photran.core.vpg.tests* package. Select the *RemoveBranchToEndIfTestSuite.java* file, right click, select **Run As->Run Configurations** as show in Figure 11below.

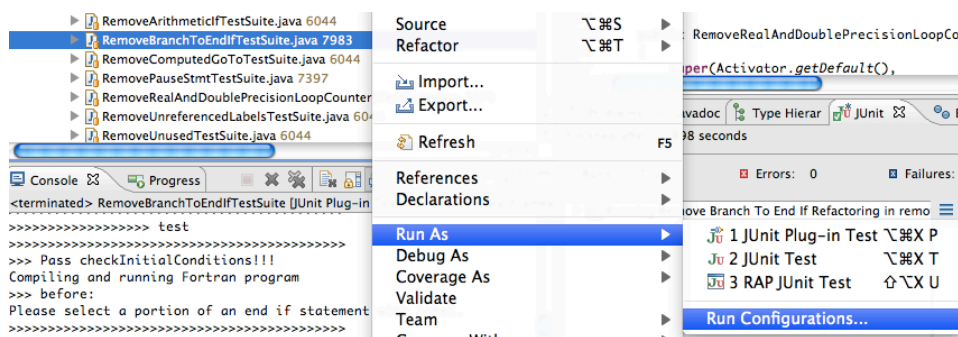



Figure 11 – Refactoring Test Suite Run Configuration

Under the JUnit Plug-in Test listing click “New Launch Configuration” as signified by the + icon in the upper right. A new run configuration will be provided based on the refactoring name selected. Figure 12 below shows the **Remove-Branch-To-End-If** run configuration created by following this process. Under the arguments tab change the “VM Arguments” field to “-ea -Xms40m -Xmx512m” to enable assertions, etc. Afterwards you must save your configuration. The user can

select the Run button to execute the test. The created test configuration will also be listed by name under the normal Eclipse test configuration drop box . The user should create run configurations for each of the 3 ROLF refactorings.

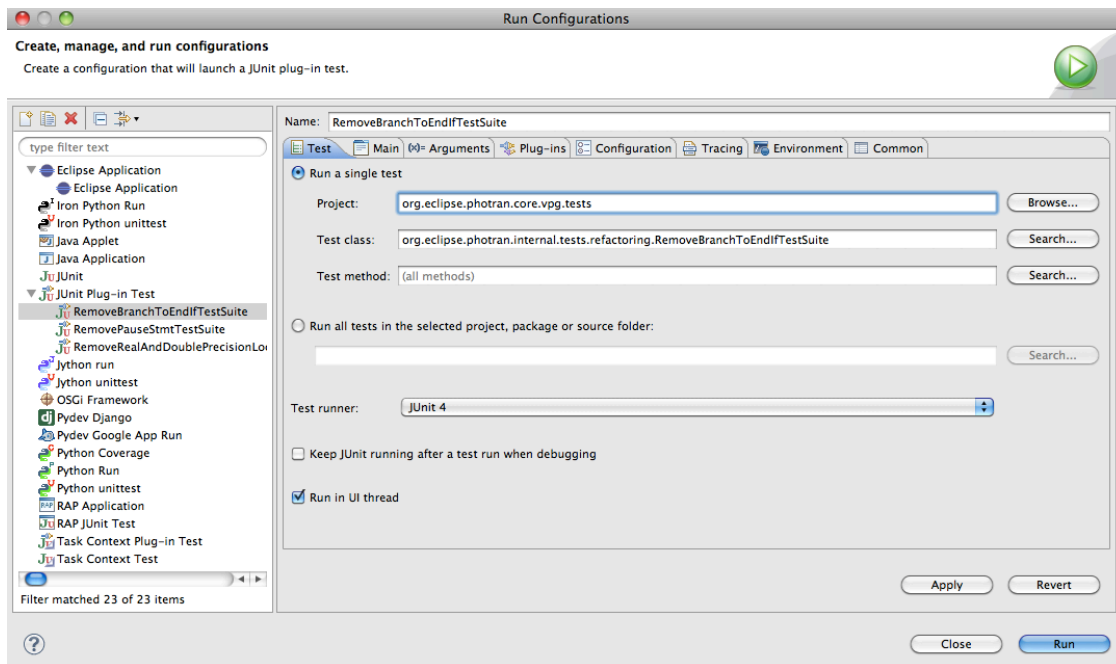


Figure 12 – Remove-Branch Configuration

## Development Team

The AvocadoChestnut programming team is made up of the following developers:

- Rita Chow ([chow15@illinois.edu](mailto:chow15@illinois.edu))
- Nicola Hall ([nfhall2@illinois.edu](mailto:nfhall2@illinois.edu))
- Jerry Hsiao ([jhsiao2@illinois.edu](mailto:jhsiao2@illinois.edu))
- Mark Mozolewski ([mozolews@illinois.edu](mailto:mozolews@illinois.edu))
- Chamil Wijenayaka ([wijenay2@illinois.edu](mailto:wijenay2@illinois.edu))

## Open Issues

There are currently no known open issues with our code.

## Release History

Release Date	Version	Description
12-7-10	1.0	Initial release. (Editor-Based Refactoring.)