Photran 7.0 User's Guide pdf version

Mariano Méndez

September 28, 2011

Contents

Contents **1**

1 Installing Photran **3**
1.1 Prerelease installation instructions 3
1.2 System Requirements 5
1.3 Installation Procedure 5
1.4 Getting Started 9
1.5 Starting a New Project 9
1.6 Converting C/C++ Projects to Fortran Projects 13
1.7 Upgrading Projects Created with Earlier Versions of Photran . 13
1.8 Writing Code 14
1.9 Using the Fortran editor and Fortran perspective 17
1.10 Compiling Fortran Programs 24
1.11 Building Projects 25
1.12 Running and Debugging Fortran Programs 27
1.13 Compiler-specific problems 29
1.14 Advanced Features 29
1.15 Troubleshooting 29

2 Introduction **31**
2.1 Enabling Advanced Features 31
2.2 Advanced Editing Features 33
2.3 Search and Navigation 35

3 Refactoring **39**
3.1 Introduction 39
3.2 Subprogram Refactorings 44
3.3 Module Refactorings 47
3.4 Use Statement Refactorings 50
3.5 Common Block Refactorings 52
3.6 Loop Refactoring 55
3.7 Refactorings to Remove Obsolete Language Features 60
3.8 Refactorings to Improve Coding Style 69
3.9 Debugging 79
3.10 Not Yet Available 94

Chapter 1

Installing Photran

1.1 Prerelease installation instructions

Prerelease versions of Photran 7.0 must be installed as follows. (The installation instructions in subsequent sections are currently incorrect and will be updated prior to the 7.0 release.)

1. Download the latest release candidate build of the Eclipse 3.6 SDK or Platform Runtime Binary (3.6RC1) from : `http://download.eclipse.org/eclipse/downloads/drops/S-3.6RC1-201005131500/`
2. Download the latest release candidate build of CDT 7.0 from: `http://download.eclipse.org/tools/cdt/builds/7.0.0/index.html`
3. Download the latest release candidate build of PTP 4.0 from `http://wiki.eclipse.org/PTP/builds/4.0.0` (this includes Photran 7.0)

To start Eclipse...

1. On Windows, unzip the file eclipse-xxxxx.zip that you downloaded above into a directory of your choice, and launch eclipse.exe. You may need to run it with administrator privileges.
2. On Linux or Mac OS X, untar the file eclipse-xxxxx.tar.gz that you downloaded above, and launch the eclipse binary.

Before you can install Photran into Eclipse, you will need to install CDT. To install CDT...

1. Click on Help ¿ Install New Software...
2. Click on the "Add..." button
3. Click on the "Archive..." button
4. Point it to the file cdt-master-7.0.0-I2010xxxxxxxx.zip that you downloaded above.

5. Click OK to close the Add Site dialog. This will return you to the Install dialog.

6. Expand "CDT Main Features" and check the box next to "Eclipse C/C++ Development Tools"

7. Click on the "Next" button

8. Click the Finish button and agree to the license to complete the installation.

9. After CDT is installed, you will be asked to restart Eclipse. You may then proceed to install Photran.

To install Photran, start Eclipse, then...

1. Click on Help ¿ Install New Software...

2. Click on the "Add..." button

3. Click on the "Archive..." button

4. Point it to the file ptp-master-4.0.0-I2010xxxxxxxx.zip that you downloaded above.

5. Click OK to close the Add Site dialog. This will return you to the Install dialog.

6. Expand "Fortran Development Tools (Photran)" and check the boxes next to "Photran End-User Runtime" and "Rephraser Engine End-User Runtime" (the latter is a supporting component)

7. If you are running Linux and have the Intel Fortran Compiler installed, or if you are on a Macintosh and have the IBM XL Fortran compiler installed, expand "Fortran Compiler Support" and select the appropriate compiler. Note that you cannot install Intel Fortran compiler support unless you are running Linux!

8. Click on the "Next" button

9. If you get an error message, see below for troubleshooting information.

10. Click the Finish button and agree to the license to complete the installation.

1.2 System Requirements

To install Photran 6.0...
You must have Eclipse 3.6 (Helios) installed.

1. You should have the C/C++ Development Tools (CDT) 7.0 installed. If you do not...

   a) If you downloaded Eclipse from eclipse.org and you will be following the instructions below for "Installing on a Machine With Internet Access," then CDT should be installed automatically when you install Photran.

   b) Otherwise, CDT probably will not be installed automatically. You will need to install CDT manually. Follow the instructions below for "Installing on a Machine Without Internet Access." You should also follow these instructions if you did not download Eclipse from eclipse.org (e.g., if you installed it from a Linux distribution like Ubuntu).

2. Eclipse must be running on a Java 1.5 or later Java Virtual Machine (JVM). To get reasonable performance, we recommend Sun's JVM [1] or IBM's J9 [2]. OpenJDK (the default JVM on newer versions Fedora Linux) also works well, although GNU Classpath (the default JVM on older versions of Fedora) is generally too slow to be useful.

3. If you want to compile and build Fortran applications, you must have a make program (such as GNU Make) and a Fortran compiler (such as gfortran, the GNU Fortran compiler) in your system path. Many Linux/Unix systems include these; details on installing them in Windows and Mac are below.

1.3 Installation Procedure

Installing on a Machine With Internet Access

To install Photran, start Eclipse, then...

1. Click on Help ►Install New Software...

2. Click on the "Add..." button

3. In the Location field, type http://download.eclipse.org/tools/ptp/releases/helios

4. Click OK to close the Add Site dialog. This will return you to the Install dialog.

5. Expand "Fortran Development Tools (Photran)" and check the box next to "Photran End-User Runtime"

6. If you are running Linux and have the Intel Fortran Compiler installed, or if you are on a Macintosh and have the IBM XL Fortran compiler installed, expand "Fortran Compiler Support" and select the appropriate compiler. Note that you cannot install Intel Fortran compiler support unless you are running Linux!

7. Click on the "Next" button

8. If you get an error message, see below for troubleshooting information.

9. Click the Finish button and agree to the license to complete the installation.

Installing on a Machine Without Internet Access

You will the following files:

If you do not have CDT installed, you will need to download the latest 7.0.x "CDT master update archive" from http://download.eclipse.org/tools/cdt/releases/helios/). This should be a file named cdt-master-7.0.x.zip (for some value of x) To install Photran, you will need the latest PTP update site archive from http://wiki.eclipse.org/PTP/builds/4.0.0) This should be a file named ptp-master-4.0.0-I2010xxxxxxxx.zip

If you do not have CDT installed, start Eclipse, then...

1. Click on Help ►Install New Software...

2. Click on the "Add..." button

3. Click on the "Archive..." button

4. Point it to the file cdt-master-7.0.x.zip

5. Click OK to close the Add Site dialog. This will return you to the Install dialog.

6. Expand "CDT Main Features" and check the box next to "Eclipse C/C++ Development Tools"

7. Click on the "Next" button

8. Click the Finish button and agree to the license to complete the installation.

9. After CDT is installed, you will be asked to restart Eclipse. You may then proceed to install Photran.

To install Photran, start Eclipse, then...

1. Click on Help ►Install New Software...

2. Click on the "Add..." button

3. Click on the "Archive..." button

4. Point it to the file ptp-master-4.0.0-I2010xxxxxxxx.zip

5. Click OK to close the Add Site dialog. This will return you to the Install dialog.

6. Expand "Fortran Development Tools (Photran)" and check the boxes next to "Photran End-User Runtime" and "Rephraser Engine End-User Runtime" (the latter is a supporting component)

7. If you are running Linux and have the Intel Fortran Compiler installed, or if you are on a Macintosh and have the IBM XL Fortran compiler installed, expand "Fortran Compiler Support" and select the appropriate compiler. Note that you cannot install Intel Fortran compiler support unless you are running Linux!

8. Click on the "Next" button

9. If you get an error message, see below for troubleshooting information.

10. Click the Finish button and agree to the license to complete the installation.

Troubleshooting

Eclipse's installer gives notoriously cryptic error messages, which, unfortunately, are out of Photran's control. Some of the more common ones are below. If you run into a different error message and cannot resolve it, please ask for help on the Photran mailing list.

Problem: You receive the following error message during installation.

Cannot complete the install because one or more required items could not be found. Missing requirement: 125xxxxxxxxxx 0.0.0.125xxxxxxxxxx requires 'org.eclipse.photran.intel.feature.group [5.0.0.xxxx]' but it could not be found

Solution: You are attempting to install support for the Intel Fortran compiler, but you are not running Linux. Go back in the installation wizard, and uncheck "Linux Intel(R) Fortran Compiler Support."

Problem: You receive the following error message during installation.

Cannot complete the install because one or more required items could not be found. Software being installed: Photran End-User Runtime 7.0.x.xxxxxxxxxxxx (org.eclipse.photran.feature.group 7.0.x.xxxxxxxxxxxx) Missing requirement: Photran VPG CDT Interface Plug-in 7.0.x.xxxxxxxxxxxx (org.eclipse.photran.cdtinterface.vpg 7.0.x.xxxxxxxxxxxx) requires 'bundle org.eclipse.cdt.core 0.0.0' but it could not be found Cannot satisfy dependency: From: Photran End-User Runtime 7.0.x.xxxxxxxxxxxx (org.eclipse.photran.feature.group 7.0.x.xxxxxxxxxxxx) To: org.eclipse.photran.cdtinterface.vpg [7.0.x.xxxxxxxxxxxx]

Solution: You do not have CDT 7.0 installed, and it couldn't be downloaded and installed automatically. Try installing CDT 7.0 first, then retry installing Photran.

Additional Instructions for Windows Users

To compile and run Fortran programs in Photran, you will need to have a Fortran compiler and make utility installed. gfortran and GNU Make are commonly used (and free). Most Linux/Unix distributions include these. Under Windows, you will need to install Cygwin [3] (which optionally includes gfortran and GNU Make) or MinGW [4] and put them on your Windows PATH.

Instructions for Cygwin

1. Install Cygwin; the defaults are mostly OK, but you will need to explicitly tell it to include the "Devel" packages (at least gcc4-fortran, gdb, and make) when the installer asks you to select what packages to install. (If gcc4-fortran is not listed under the "Devel" packages, you may have chosen a bad mirror; restart the Cygwin installation, and choose a different mirror instead. Georgia Tech's mirror at gtlib.gatech.edu is generally quite fast and reliable, for example.)
2. Add the Cygwin directories to your Windows PATH. Under Windows XP, the process is as follows:
   a) Make sure you are logged in under an administrator account.
   b) Open the Control Panel.
   c) Double-click the System icon.
   d) Switch to the Advanced tab.
   e) Click the Environment Variables button.
   f) Under System Variables, find the variable "Path" in the list, and click on it.
   g) Click Edit.
   h) At the end of the "Variable Value" text, add
   i) Click OK, click OK, click OK, and close the Control Panel.
   j) Close and re-open Photran. Windows should now search

Additional Instructions for Mac OS X Users

If you install gfortran on Mac OS X, it may be installed in /usr/local/bin, which is not (by default) on the PATH. If you are launching Eclipse from a Terminal, the PATH can be set by modifying /etc/paths. However, if you are launching Eclipse from the Finder (by double clicking on it) or the Dock, then the PATH is not obtained from the shell or /etc/paths. Instead, it's obtained

from /.MacOSX/environment.plist. The format of the environment.plist file is as follows (change the path appropriately). If you create or edit this file, you will need to log out (or reboot) before the changes will take effect.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist
PUBLIC "-//Apple Computer//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
    <key>PATH</key>
    <string>/usr/local/mpi/bin:/usr/local/bin:/bin:/sbin:/usr/bin:
    /usr/sbin:/sbin:/usr/sbin:/Developer/Tools:/usr/texbin:/Users/greg/bin

    </string>
</dict>
</plist>
```

1.4 Getting Started

Photran is an integrated development environment and refactoring tool for Fortran 77, 90, 95, and 2003. It includes an editor with "smart" syntax highlighting, a GUI debugger, CVS support, Makefile-based compilation, error message recognition for many common Fortran compilers, Fortran language-based search and code navigation, content assist, a Fortran declaration view, and many refactorings, including popular refactorings such as Rename and Extract Procedure as well as Fortran-specific refactorings like Introduce Implicit None.

Photran is based on Eclipse, and some of its components are reused from the Eclipse C/C++ Development Tools (CDT). Throughout this documentation, you will see references to relevant parts of the Eclipse and CDT documentation, which can be accessed by clicking on the Help menu (inside Photran/Eclipse) and clicking Help Contents.

If you have never used Eclipse before, please work through the Eclipse Workbench tutorial before attempting to use Photran. Click on Help ►Help Contents in Eclipse, and navigate to Workbench User Guide ¿ Getting started ¿ Basic tutorial. This will introduce you to Eclipse concepts such as perspectives and views; such terminology will be used throughout this manual.

1.5 Starting a New Project

One of the advantages (or disadvantages) of using Photran is that it does not include a Fortran compiler. Instead, it uses make to compile Fortran programs. This allows you to use any Fortran compiler you want, but it also means that a Makefile must be written which tells the make program how to compile your Fortran program.

- For small, simple projects, Photran can auto-generate a Makefile which will use the GNU Fortran compiler (gfortran).
- For more complex projects, you can write a custom Makefile.

For more information on project types, click on Help ►Help Contents in Eclipse, and navigate to C/C++ Development User Guide ►Concepts ►CDT Projects.

Starting a Project with an Auto-Generated Makefile

The following is a typical procedure for starting a new project using an auto-generated Makefile. Your system must have the GNU Fortran (gfortran) compiler installed for this procedure to work. To make sure that gfortran is installed, you can open a command prompt (Windows) or terminal window (Linux/Mac) and type "gfortran" (without the quotes); it should respond with "gfortran: no input files".

1. Click File ►New ►Fortran Project
2. Call it HelloFortran
3. Choose "Executable (Gnu Fortran)" from the project type list
4. Choose "GCC Toolchain" from the toolchain list (you may need to first uncheck the "Show project types..." check box at the bottom of the window before you can see this)
5. Click Next
6. Click on Advanced Settings
7. Expand C/C++ Build in the list on the left, and click on Settings
8. Click on the Binary Parsers tab. Check the appropriate parsers for your platform. If you are using Windows, check PE Windows Parser and/or Cygwin PE Parser; if you are using Linux, check Elf Parser; if you are using Mac, check Mach-O parser.
9. Click on the Error Parsers tab. Check the error parser(s) for the Fortran compiler(s) you will use.
10. Click OK
11. Click Finish
12. Click File ►New ►Source File
13. Call it hello.f90; click Finish
14. Type the standard "Hello, World" program shown below, and click File ►Save.
15. Open the Console view, and make sure "make" ran OK and compiled your program
16. In the Fortran Projects view, expand the Binaries entry, and click on your executable (e.g., "HelloFortran.exe - [x86le]")

17. Run — Run As — Run Local Fortran Application
18. Choose GDB Debugger (Cygwin GDB Debugger if you're under Windows)
19. Check the Console view, and make sure "Hello World" appeared.

hello.f90:

```
program hello
    print *, "Hello World"
end program
```

Starting a Project with a Hand-Written Makefile

The following is a typical procedure for starting a new project using a hand-written Makefile. The following example uses the GNU Fortran (gfortran) compiler, but any Fortran compiler can be used instead. We will assume that you are familiar with how to write a Makefile. If not, there are plenty of resources on the Web, including a tutorial from an introductory Computer Science course at UIUC [5] and another from the University of Hawaii [6]... and, of course, you can always read the entire manual for GNU Make [7].

1. Click File ►New ►Fortran Project
2. Call it HelloFortran
3. Expand "Makefile project" in the project type list (it has a folder icon), and choose "Empty Project"
4. Select "– Other Toolchain –" in the toolchain list in the right-hand column, and click Next
5. Click on Advanced Settings
6. Expand C/C++ Build in the list on the left, and click on Settings
7. Click on the Binary Parsers tab. Check the appropriate parsers for your platform. If you are using Windows, check PE Windows Parser and/or Cygwin PE Parser; if you are using Linux, check Elf Parser; if you are using Mac, check Mach-O parser.
8. Click on the Error Parsers tab. Check the error parser(s) for the Fortran compiler(s) you will use.
9. Click OK

10. Click Finish
11. Click File ►New ►Source File
12. Call it hello.f90
13. Click Finish
14. Type the standard "Hello, World" program shown below.
15. Click File ►New ►File
16. Call it Makefile
17. Click Finish
18. Create a Makefile similar to the one shown below. Again, we assume you are familiar with the structure of a Makefile. You cannot simply copy-and-paste this example because the gfortran and rm lines must start with a tab, not spaces. The -g switch instructs gfortran to include debugging symbols in the generated executable so that it can be debugged later. The -o switch tells it what to name the generated executable.
19. Click Project ►Clean, then click OK
20. Open the Console view, and make sure "make" ran OK and compiled your program
21. In the Fortran Projects view, expand the Binaries entry, and click on your executable (e.g., "hello.exe - [x86le]")
22. Click Run ►Run As ►Local Fortran Application
23. Choose GDB Debugger (Cygwin GDB Debugger if you're under Windows)
24. Check the Console view, and make sure "Hello World" appeared.

hello.f90

```
program hello
    print *, "Hello World"
end program
```

Makefile: **(You MUST replace the spaces beginning the gfortran and rm lines with a tab character!)**

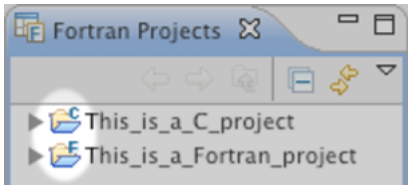
```
all: gfortran -o hello.exe -g hello.f90
clean: rm -f hello.exe
```

1.6 Converting C/C++ Projects to Fortran Projects

Every Fortran project is also a C project, so CDT's functionality will work as expected on Fortran projects. However, a C/C++ is not a Fortran project until it has been explicitly converted. (In the Fortran Projects view, a Fortran project will have an "F" icon, while a C/C++ project will have a "C" icon.)

To convert an existing C/C++ project to a Fortran project:

1. Switch to the Fortran perspective.
2. In the Fortran Projects view, right-click on the project you want to convert, and select Convert to Fortran Project from the pop-up menu.
3. The project should now have an "F" icon in the Fortran Projects view.



1.7 Upgrading Projects Created with Earlier Versions of Photran

There are three major changes in Photran 6 that affect users of previous versions of Photran:

1. Projects must be converted to Fortran projects. In the past, Photran would treat C projects and Fortran projects similarly. In fact, in Photran 4 and earlier versions, Photran's new project wizard actually created C projects! But in Photran 6, many of Photran's features will only be available after a project is converted to a Fortran project. (Every Fortran project is a C project too, so CDT's features will still work on the project.) Learn how to convert projects.
2. Project properties must be reset. Photran 6 uses a new mechanism for storing project properties. Unfortunately, this means that the project properties for old projects must be re-created. This includes enabling analysis/refactoring and setting include paths and module paths.
3. Source form settings must be set for the project. Previous versions of Photran used Eclipse's workspace-wide settings to distinguish free and fixed form files. In Photran 6, each project has its own settings that determine which filename extensions correspond to free and fixed source form (and which files contain C preprocessor directives). Learn about source form settings.

1.8 Writing Code

Configuring Source Forms and Filename Extensions

Photran supports both fixed-form and free-form Fortran code. Photran also supports C preprocessor directives, such as #define, #ifdef, #include, __FILE__, and __LINE__ in Fortran code.

Fixed form is an older form designed for punch cards. It is more common in Fortran 77 and earlier programs, although it is still part of Fortran 2003. In fixed form, a "c", "C", "*", or "!" in column 1 indicates a comment; columns 1 through 5 are used for a statement label; a character in column 6 indicates a continuation line; and program statements are contained in columns 7 through 72. Anything after column 72 is ignored. (Although column 72 is specified in the Fortran standard, most compilers allow you to change this to permit longer lines; Photran allows you to change it as well, as described in the next section.)

Photran uses filename extensions to determine whether a file is in fixed- or free-form and whether or not it is C-preprocessed. The default filename extensions are as follows.

Filename Extensions	Source Form/Preprocessing
.f, .fix, .for, .fpp, .ftn .F, .F77, .FIX, .FOR, .FTN, .FPP, .fpp	Fixed source form
.f08, .f03, .f95, .f90	Free source form with INCLUDE lines
.F08, .F03, .F95, .F90	Free source form with C preprocessor directives

Configuring Source Form/Filename Extension Associations

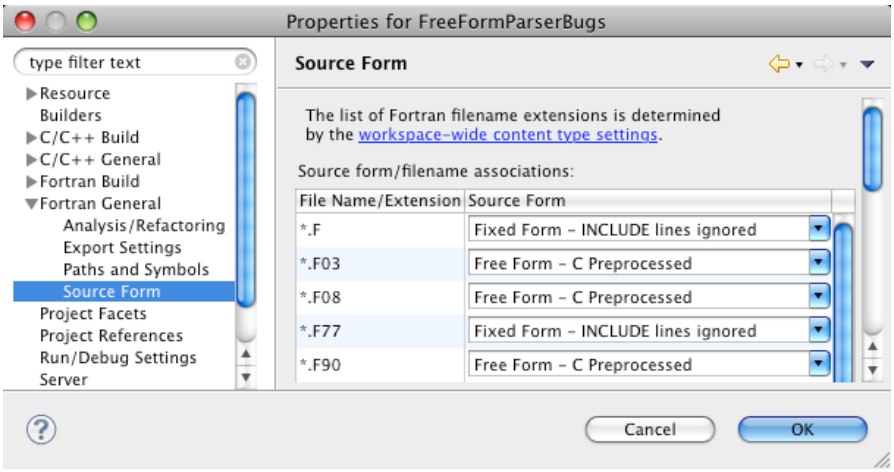
The filename extensions listed above are the defaults for new Fortran projects. To change them:

1. In the Fortran Projects view, right-click on a project, and choose Properties.
2. In the tree on the left, navigate to Fortran General ►Source Form.
3. For each filename extension(s) you want to change, select the appropriate source form from the dropdown box.
4. Click OK to close the dialog box.

Adding Additional Filename Extensions

Additional filename extensions can be added by changing the Eclipse Content Type preferences.

1. Click on Window ►Preferences... (Eclipse ►Preferences... on Mac OS X).



a) In the "Content types:" tree in the right pane, navigate to Text ►Fortran Source File.

2. Click the Add... button.

3. To make all files with a ".fre" filename extension be recognized as Fortran source code, enter *.fre in the dialog. You can also specify individual filenames by omitting the *. prefix; for example, entering old-fortran-code.txt will make any file named old-fortran-code.txt be recognized as Fortran source code, but not other .txt files.

4. Click OK to close the dialogs.

For more information on configuring content types, click on Help ►Help Contents in Eclipse and navigate to Workbench User Guide ►Reference ►Preferences ►Content Types.

Configuring Fixed Form Line Width

While free source form allows lines to be arbitrarily long, in fixed form, there is a predetermined maximum line width. Often, this is 72 characters (i.e., anything after column 72 is ignored and treated as a comment), although most compilers allow you to change this to permit longer lines. Photran allows you to change it as well.

To change the maximum line width in fixed form sources...

1. Open the workbench preferences. On Windows or Linux, click Window ►Preferences; on Mac OS X, click Eclipse ►Preferences.

2. Expand the Fortran category, and choose Editor.

3. Change the value of the field labeled "fixed form line length." The value must be at least 72.

If you have any fixed form editors open, close and reopen them; notice that the rightmost, gray vertical bar has been moved to the column indicated, and the parser (which constructs the Outline view) has adjusted accordingly.

NOTE: This is a workspace-wide preference: It affects all fixed form files in the workspace. It is not currently possible to adjust this setting per project or per file.

The Fortran Perspective

When you write Fortran code, ordinarily you will work in the Fortran perspective. When you create a new Fortran project (as described above), the wizard will ask if you want to switch to the Fortran perspective, or you can do it manually by clicking Window ►Open Perspective ►Other... and selecting Fortran from the list.

For more information on perspectives, click on Help ►Help Contents in Eclipse, and navigate to Workbench User Guide ►Concepts ►Perspectives. Perspectives are also covered in the Eclipse tutorial, which can be found in Workbench User Guide ►Getting started ►Basic tutorial.

In particular, note that you can add views to a perspective by clicking Window ►Show View ►Other... and selecting a new view from the Show View dialog. You can also reset the current perspective to its original layout: From the menu bar, choose Window ►Reset Perspective.

By default, the Fortran perspective contains a central area for editing code, as well as the following views:

- The Fortran Projects view displays your project(s) and any files and folders in it.
- The Problems view will show errors from your Fortran compiler, if Photran can recognize them. (This requires configuring an error parser, described elsewhere in this manual.)
- The Console view shows the output from your Fortran compiler. When you run your Fortran program, its output is also displayed in the Console view.
- The Fortran Declaration view is described in the Photran Advanced Features manual.
- The Outline view shows the high-level structure of your program, i.e., the subprograms, modules, derived types, etc. comprising it.
- The Make Targets view allows you to quickly run make on a particular target defined in a custom Makefile.

For more information on the Make Targets view, click on Help ►Help Contents in Eclipse, and navigate to C/C++ Development User Guide ►Reference ►C/C++ Views and Editors ►Make Targets view.

The Fortran Projects View

The Fortran Projects view displays project files in a tree structure. In this view you can do the following:

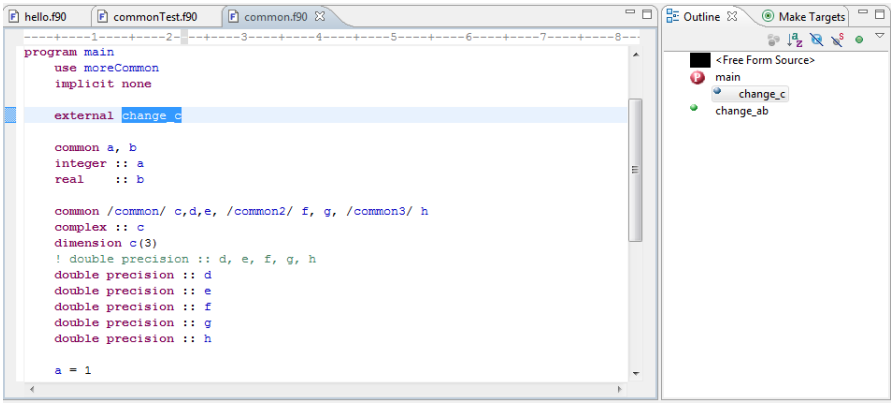
- Open files in the Fortran editor
- Manage existing files (cut, paste, delete, move or rename)
- Create new projects, files, or folders
- Import or Export files and projects
- Browse the high-level structures in Fortran files
- Open projects in a new window
- Perform some multi-file refactorings (such as Introduce Implicit None; see the Photran Advanced Features Manual)

Files can be opened by double-clicking on the filename or by right-clicking on the file and choosing "Open With" from the context menu. Most other actions are performed by right-clicking on the file and choosing an action from the context menu.

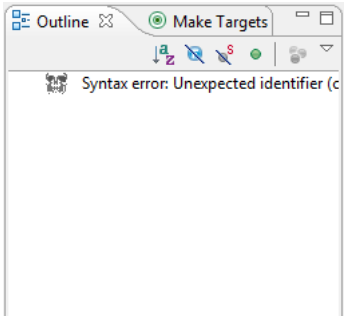
The Fortran projects view is similar to the C/C++ Projects View. For more information on the Fortran Projects view, refer to the documentation on the C/C++ Projects view by clicking on Help ►Help Contents in Eclipse and navigating to C/C++ Development User Guide ►Reference ►C/C++ Views and Editors ►C/C++ Projects view.

1.9 Using the Fortran editor and Fortran perspective

Outline View



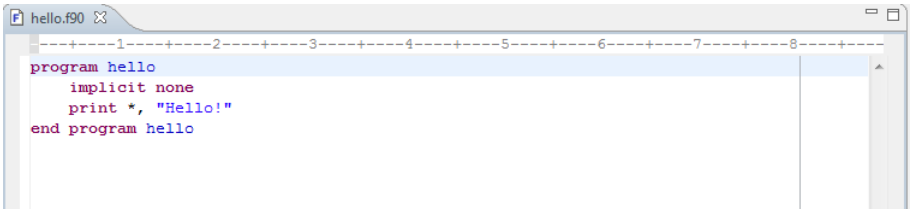
When a Fortran file is open in the Fortran editor, the Outline view shows an outline of the structural elements of that file: subprograms, main programs, modules, derived types, etc. The Outline view notes the content type of the file (free form or fixed form); it will also note when there is an error in the code. To add the Outline view to the current perspective go to Window ►Show View ►Other... ►General ►Outline.



Overview of the Fortran editor

The Fortran editor includes a number of features. The main features to note are the horizontal ruler at the top of the editor, code folding, and syntax highlighting.

The horizontal ruler measures the width of the editor in characters: There is either a symbol (-/+) or number for each column in the editor. A plus sign (+) is shown every fifth character, and every tenth character is numbered. It is also possible to turn on the print margin (gray vertical line) in the 80th column; this is discussed further below in the section titled "Configuring the Editor."

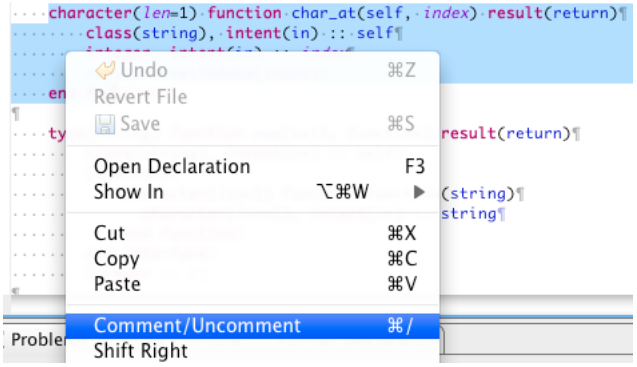


Code folding is also supported in the free form editor, but turning on code folding will disable the horizontal ruler; how to do this is described below as well. When folding is turned on, a minus sign (-) will be displayed in the left margin of the editor next to each program, function, subroutine, etc.; clicking on the minus sign will temporarily collapse the display of that entity to a single line.

The editors also support syntax highlighting. As you write programs in Fortran, you will notice that different colors are applied to comments, identifiers, intrinsics, keywords, and strings to make code more readable. It is possible to change the colors for each of the previously mentioned code components in the Fortran editor preferences (this is discussed in the section "Configuring the Editor," below). Often, the editor can even distinguish between, for example, the keyword "if" and a variable named "if." However, it is not perfect; it uses a faster, but less reliable, heuristic than the actual Fortran parser that drives the Outline view and all of the Advanced Features.

Comment/uncomment

Photran can comment or uncomment a block of Fortran statements, placing an exclamation point (!) before the statements or removing it, respectively. Select the statements in the Fortran editor. Then right-click the selection, and choose Comment/Uncomment from the context menu.



Setting bookmarks

Bookmarks are an Eclipse feature which allows you to mark important lines in a file and then quickly jump to those lines using the Bookmarks view. To set

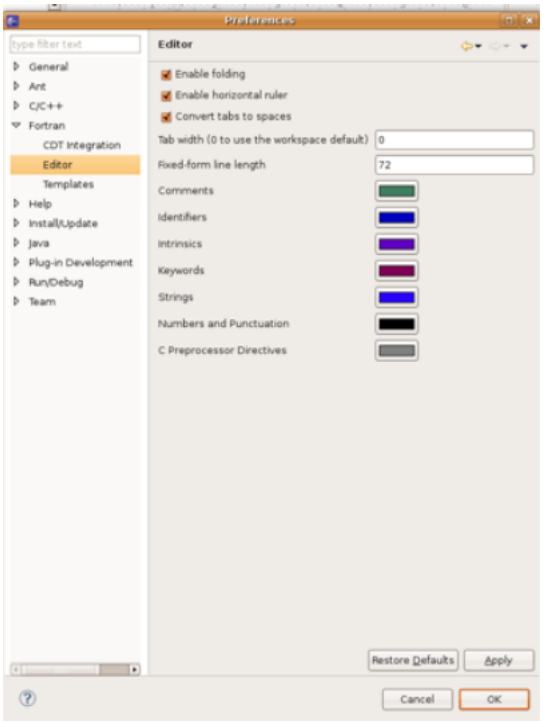
a bookmark, right-click the grey bar on the left side of the editor at the line where the bookmark should be located and select "Add Bookmark..." from the popup menu. A bookmark can also be placed on a line by locating the cursor on the line, going to Edit ►Add Bookmark... in the menu bar. Type in the desired name for the bookmark in the dialog box and click OK. A blue bookmark will show up in the grey bar on the left side of the editor.

Bookmarks can then be viewed in the Bookmarks view (Window ►Show View ►Other..., select Bookmarks in the General folder).

Go to line (Ctrl+L)

The "Go To Line..." feature allows you to jump to a specific line number in a file when it is open in a Fortran editor. In the menu bar, go to Navigate ►Go To Line... (or press the hotkey, Ctrl+L on Windows/Linux or Command+L on Mac) to open the Go To Line dialog box. Typing in the line you wish to navigate to, and clickg OK. The cursor will be moved to the beginning of the specified line.

Configuring The Editor



Selecting horizontal ruler vs. folding

Fortran editors can either display the horizontal ruler, or they can have code folding enabled, but not both. To determine which will be enabled, go to Window ►Preferences in the menu bar. Expand Fortran and select Editor. Check the appropriate box(es) to enable folding in the free-form and/or fixed-form Fortran editor. Setting colors for syntax highlighting

The colors for Fortran syntax highlighting can be modified in the Fortran editor preferences. To modify the color settings, go to Window ►Preferences in the menu bar. Expand Fortran and select Editor. The color choosers for Comments, Identifiers, Intrinsics, Keywords, and Strings are available on this page.

Showing the print margin

Editors can optionally display a gray vertical bar at a specified column; generally, this is set in column 80 to act as a visual "print margin." To enable display of the print margin, go to Window ►Preferences in the menu bar. Under General, expand Editors, and select Text Editors. Check the box for "Show print margin". Optionally, you can also change the column in which the print margin will be displayed. Click Apply and the print margin should now display in the editor.

Setting fixed-form line length

In fixed form, Fortran specifies that anything past column 72 (default) is a comment. However, that can be changed to a value other than 72. To change the length of the line, go to Window ►Preferences in the menu bar. Under Fortran, select Editor. Change the value of field labeled "Fixed-form line length" to the desired length of the line. Note that the accepted range for the length of the line is 72-999 inclusive.

Converting tabs to spaces

To enable conversion of tabs to spaces, go to Window ►Preferences in the menu bar. Under General, expand Editors, and select Text Editors. Check the box for "Insert spaces for tabs" to have Eclipse automatically convert Tabs to spaces.

Note that this will only apply to new tabs that you type; it will not change your existing files.

It is easy to replace tabs with spaces in an existing file, as long as the tabs all occur at the beginning of a line. (For tabs in the middle of a line, this won't work quite right.) (1) Move the cursor to the beginning of the file; (2) click on Edit ►Find/Replace...; (3) in the Find box, type ""^ (without the quotes); (4) in the Replace With box, type 4 or 8 spaces (each tab character will be replaced with exactly what you type, so if you type four spaces, each tab character will

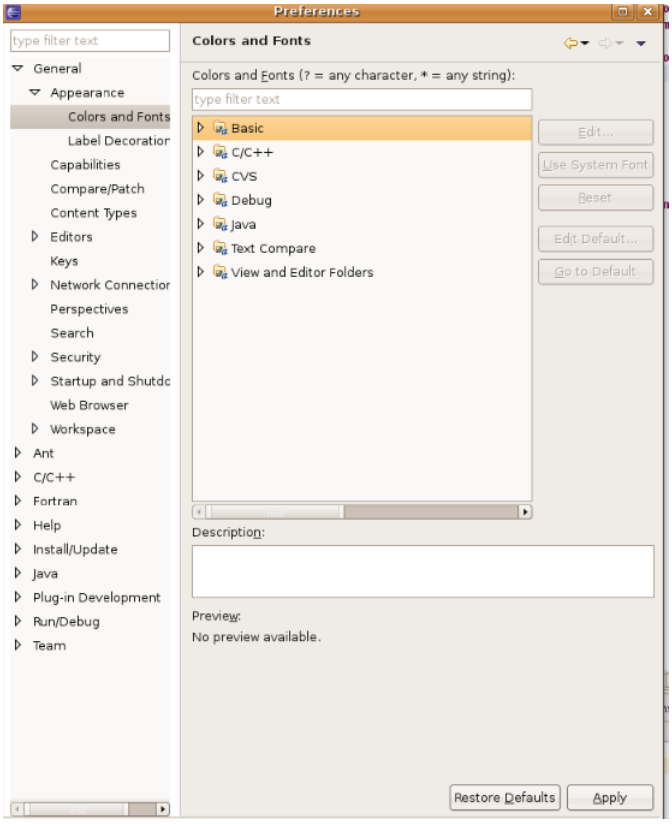
be replaced with four spaces); (5) select Search: Forward and Scope: All; (6) check the "regular expression" box; and (7) click Replace All. This will replace every tab character with spaces.

Show whitespace

Whitespace characters can be displayed by going to Window ►Preferences. Under General ►Editors ►Text Editors, check the box next to "Show whitespace characters" and click Apply. Spaces, tabs, and newlines will now be shown as light gray double-angle quotes (), dots (), and paragraph symbols (), respectively.

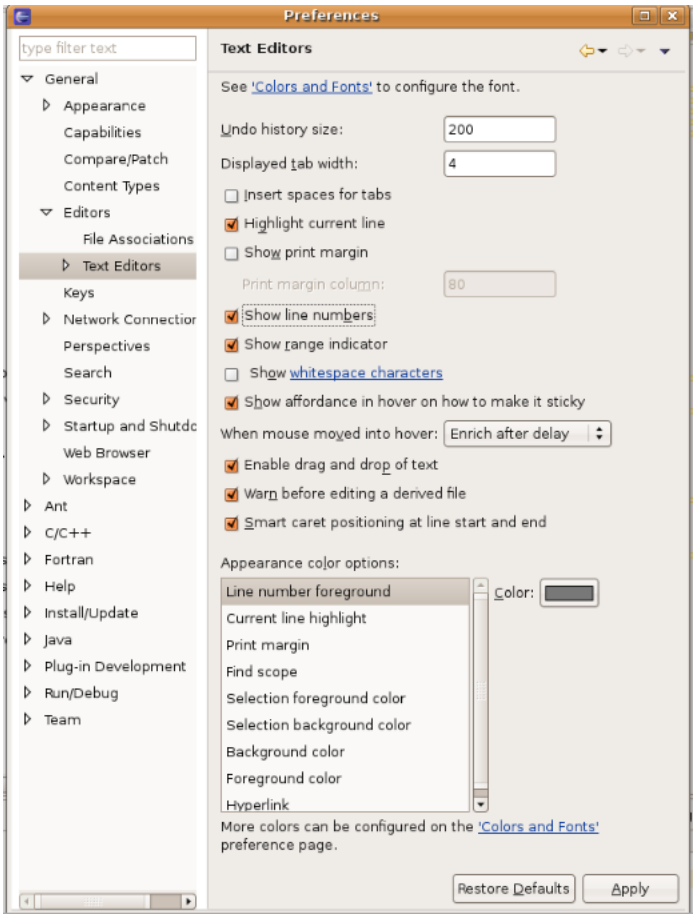
Set Font

Photran uses the same font as the basic Eclipse text editor. To change it, click on Window ►Preferences in the menu bar. Under General, expand Appearance, and select Colors and Fonts. Search for the font you want.



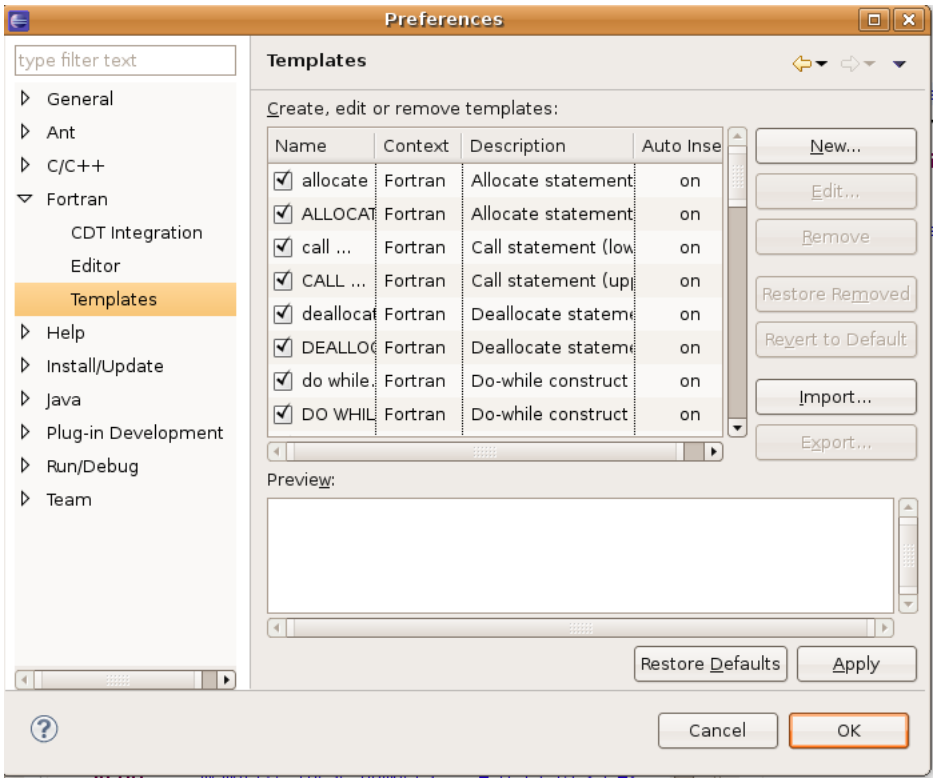
Show line numbers

Line numbers can be displayed in the editor in the Text Editor preferences page. Go to Window ►Preferences. Under General ►Editors ►Text Editors, check the box "Show line numbers" and click Apply.



Code Templates

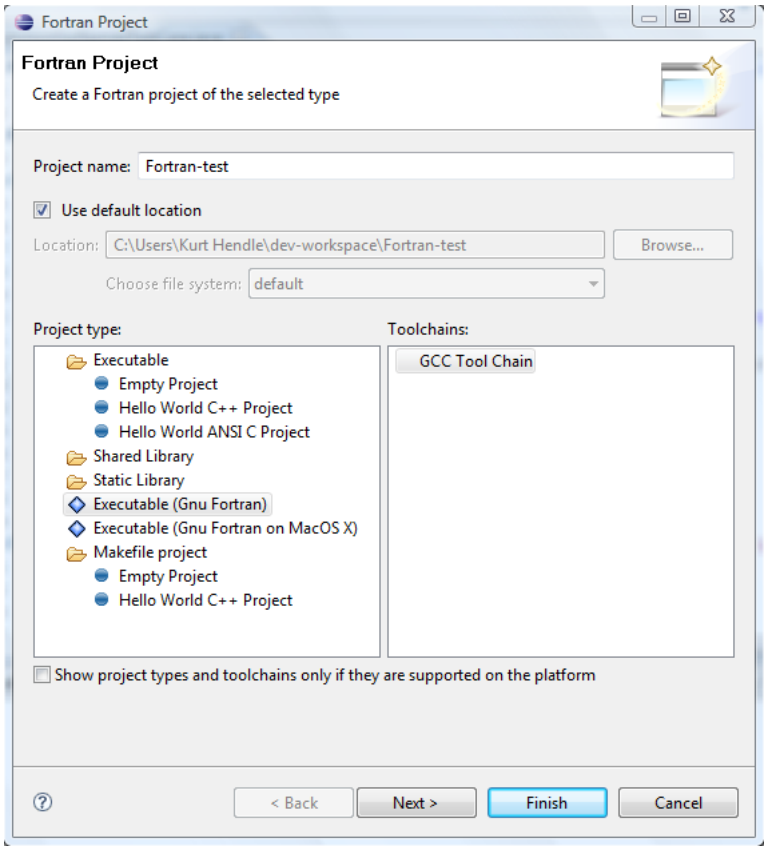
Code templates are used to make coding faster and easier. After typing the first letter of whatever you are typing, press Ctrl+Space and a list of choices will appear. Go to Windows ►Preferences ►Fortran ►Templates to manage which templates you want available.



1.10 Compiling Fortran Programs

To compile a project, Photran generally executes whatever make command is on your system path. (On most systems, this will invoke GNU Make.) As noted above, when you create a project, you can decide if Photran will auto-generate a Makefile that uses GNU Fortran, Intel Fortran, or IBM XL Fortran, or you can elect to hand-write your own Makefile.

To use an automatically generated Makefile, choose "Executable (Gnu Fortran on XYZ)" from the Project type menu, where XYZ is your platform (Windows, Linux, or Mac OS X). There are similar options available for Intel Fortran and IBM XL Fortran. To create a project using a custom Makefile, in the project type selection box, expand the Makefile Project folder, and select Empty Project.



1.11 Building Projects

Compiling Fortran projects is identical to compiling C/C++ Projects.

For more information on building projects, click on Help ►Help Contents in Eclipse, and navigate to C/C++ Development User Guide ►Concepts ►Build ►Building C/C++ Projects. [8]

Setting Make Targets

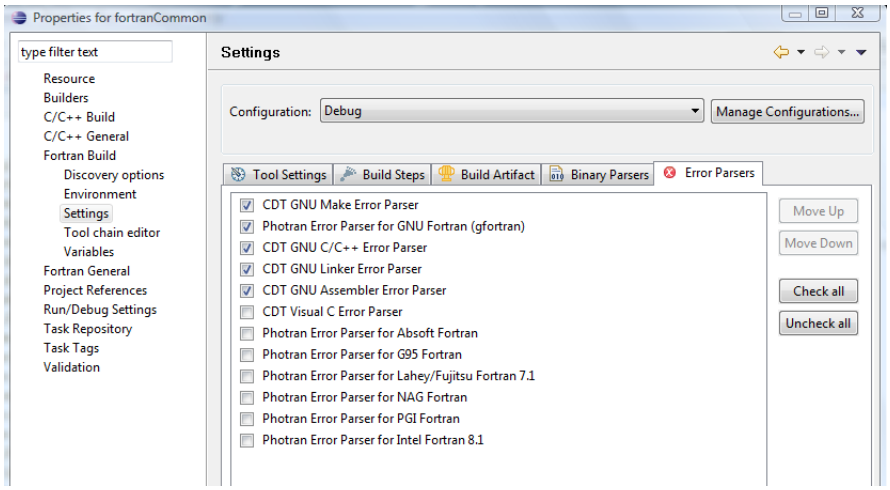
Setting make targets for Fortran programs is also identical to setting make targets for C/C++ programs.

For more information on make targets, click on Help ►Help Contents in Eclipse, and navigate to C/C++ Development User Guide ►Tasks ►Building projects ►Creating a make target. [9]

Marking Error Messages from Your Fortran Compiler (Error Parsers)

When you build (compile) your Fortran program, the output from your Fortran compiler (including any error messages) will be displayed in the Console view. However, for many compilers, Photran can "recognize" error messages, placing the problem description in the Problems view and marking the corresponding line in the source file with a red X icon.

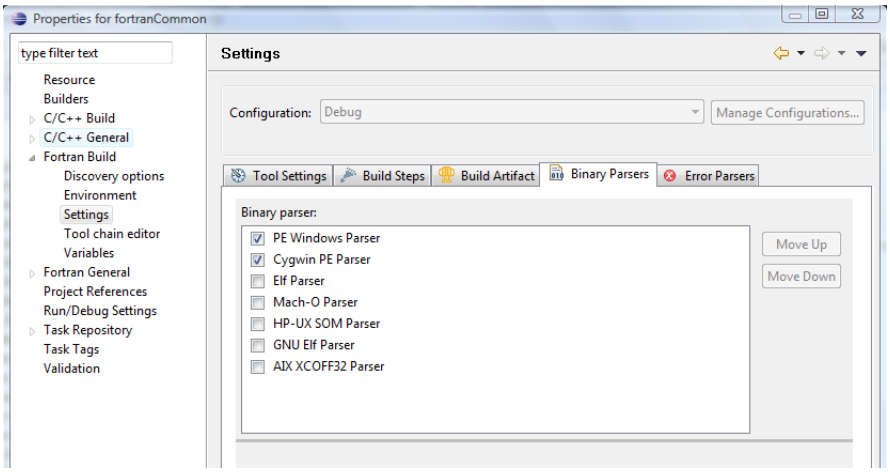
To do this, you must have the correct error parsers selected in your Fortran project's build settings. To modify these settings, right-click on your Fortran project, go to Properties ►Fortran Build ►Settings and select the Error Parsers tab. A list of error parsers is displayed; select the one(s) corresponding to the Fortran compiler(s) you are using and click OK.

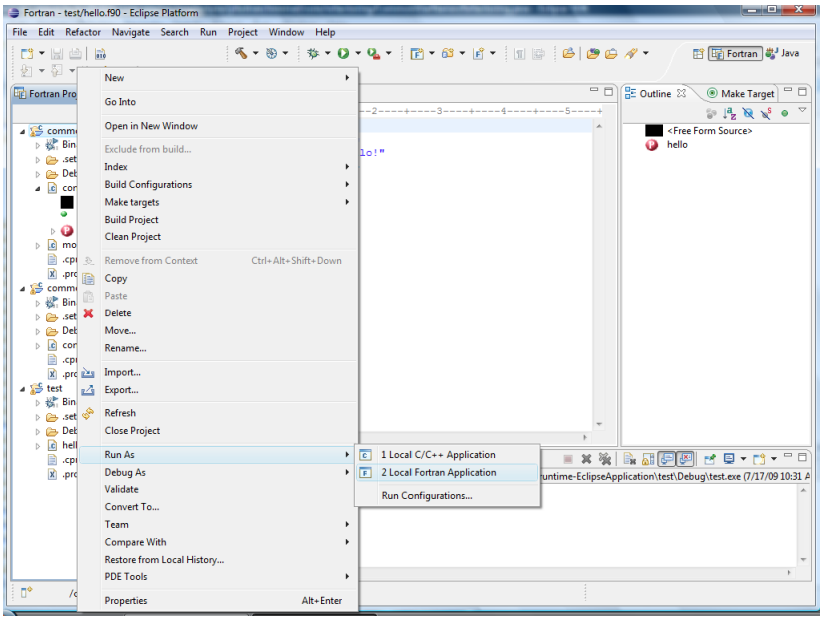


Recognizing Executables (Binary Parsers)

In order to run (execute) your Fortran application, you will need to tell Eclipse which Binary Parser(s) to use for your project. This can be done in the project properties (Right-click project ►Properties ►Fortran Build ►Settings ►Binary Parsers Tab).

Which binary parser you want to use will depend on the operating system environment you are working in. For example, the default choice is the Elf Parser which is for Linux environments. Windows users would need to deselect the Elf Parser and select the PE Windows Parser and/or the Cygwin PE Parser. Mac users should choose the Mach-O parser.





1.12 Running and Debugging Fortran Programs

To run a Fortran application, right-click the project in the Fortran Projects view, and click Run As ▶Local Fortran Application. This will build and run the Fortran application, and output will be printed to the console. (Remember, if you are using a custom makefile, the makefile will need to be written correctly for the project to be built.)

If you want to manage the run configuration (for example, if you want to pass command line arguments to your Fortran application or set certain

environment variables), right-click on the project in the Fortran Projects view, click Run As ►Run Configurations... Create a new launch configuration under Fortran Local Application (or edit an existing one) and modify the settings as needed.

Debugging programs works similarly, except you can choose Debug As ►Local Fortran Application or Debug Configurations from the context menu.

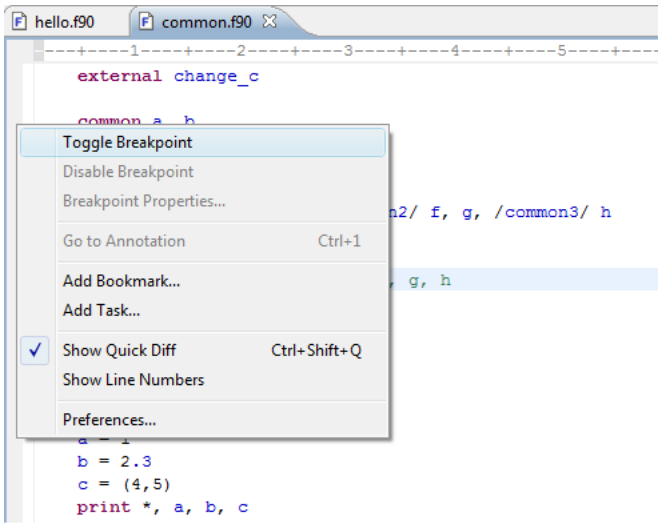
Running and debugging Fortran applications is similar to running and debugging C/C++ applications. For more information on running and debugging binary applications, click on Help ►Help Contents in Eclipse, and navigate to C/C++ Development User Guide ►Tasks ►Running and debugging projects. [10]

Setting Breakpoints

There are two ways to set breakpoints in the editor.

The easiest way to set a breakpoint in the editor is to double click in the left margin (gray vertical bar) on the line at which you wish to set the breakpoint. A blue dot will appear in the margin on the line where the breakpoint is set. Double clicking again at the same location will remove the breakpoint.

You can also set a breakpoint by right-clicking in the left margin on the line where you wish to set the breakpoint and clicking "Toggle Breakpoint." The blue dot indicating the breakpoint will appear or disappear (if already set) in the left margin of the editor.



When you switch to the Debug perspective, the breakpoints you set will be listed in the Breakpoints view.

For more information on debugging, click on Help ►Help Contents in Eclipse, and navigate to C/C++ Development User Guide ►Tasks ►Running and debugging projects ►Debugging. Known Run/Debug Issues

When using Cygwin in Windows, debugging a program can result in a "cannot find gcc source" error as described in Bug 280492. Note that this error does not have an effect on the running of the program. The program will still run correctly in Debug mode.

1.13 Compiler-specific problems

G95

In debug mode, using the step button to step through the code line-by-line sometimes jumps ("leapfrog" effect) around in the source code editor view. For example, pressing F6 (step) after a breakpoint several times, might sometimes result in a jump to line 1 and stepping again will return to the correct line in the code.

gFortran

No known issues.

Intel Fortran Compiler

No known issues.

IBM XLF

No known issues.

1.14 Advanced Features

Photran 7.0 includes a number of sophisticated features that are designed to make it easier to write, modify, search, and maintain Fortran code. These include content assist, which can "auto-complete" variable and function names as you type; a declaration view, which can show the leading comments for the selected variable or procedure; Fortran Search, which allows you to find declarations and references to modules, variables, procedures, etc.; and refactorings, which change your source code to improve its design while preserving its behavior.

1.15 Troubleshooting

If you are experiencing problems installing or working with Photran, please review the FAQ [11]. If you don't find an answer there, please join the Photran mailing list [12]; a large number of users (as well as Photran's developers) monitor that list, and they are generally eager to help. Retrieved from "http://wiki.eclipse.org/PTP/photran/documentation/photran7"

Chapter 2

Introduction

Photran 7.0 includes a number of sophisticated features that are designed to make it easier to write, modify, search, and maintain Fortran code. These include content assist, which can "auto-complete"variable and function names as you type; a declaration view, which can show the leading comments for the selected variable or procedure; Fortran Search, which allows you to find declarations and references to modules, variables, procedures, etc.; and refactorings, which change your source code to improve its design while preserving its behavior.

2.1 Enabling Advanced Features

In order to use any of the advanced features described in this document, you must specifically enable them as described below. After you do this, Photran will index your project; that is, it will build a database of what modules, subprograms, etc. are declared in every file in your project. This information will be updated incrementally every time you save a file. Although this process is usually reasonably fast, it may become disruptive when working on very large projects, so it has been disabled by default. Note that the first time your project is indexed, it may take a while, because Photran must analyze every file in your project; after that, it will only index files that have changed (and files that depend on a file that has changed), so it will generally be much faster.

How to Enable Advanced Features

1. Right-click on your project in the Fortran Projects view
2. Click on Properties
3. Expand Fortran General in the list on the left, and click on Analysis/Refactoring (see screenshot fig. 2.3)
4. Check the "Enable Fortran analysis/refactoring" check box,

5. If you want to enable content assist, the Fortran Declaration view, etc., check those boxes as well
6. You may also want to set module and include paths at this point (see fig.2.3)
7. Click OK

Setting Module and Include Paths

If your source code contains INCLUDE lines or USE lines referencing modules in other files, Photran needs to know where to look in order to find these. It will not figure this out automatically. For each project in which you plan to use refactoring support,

1. Right-click on your project's folder in the Fortran Projects view
2. Click on Properties
3. Expand Fortran General in the list on the left, and click on Analysis/Refactoring
4. List the folders in which Photran should search for INCLUDE files and modules when refactoring. They will be searched in order from the first folder listed to the last. Subfolders are not searched automatically; you must include them explicitly.
5. Click OK

Advanced Features and C Preprocessed Code

Starting with Photran 6, C preprocessor directives are supported in Fortran code, as described in the Photran User's Guide. While most of the advanced features described in this manual can handle C preprocessed code, refactorings currently cannot. If you attempt to refactor C preprocessed code, you will receive an error message.

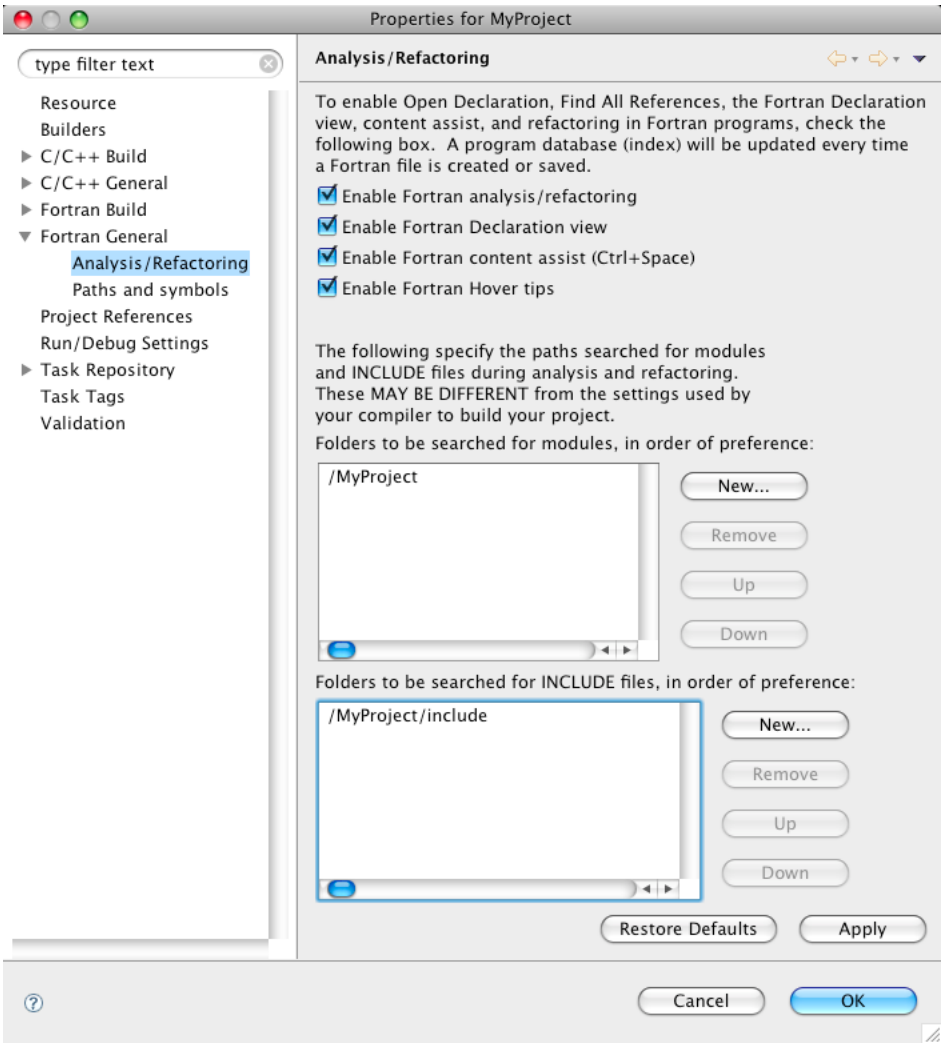


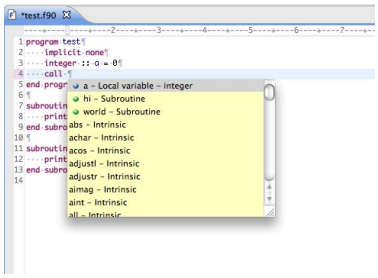
Figure 2.1: Enable Fortran analysis/refactoring

2.2 Advanced Editing Features

Content Assist

As you are typing a Fortran program, content assist pops up a list of the subprograms, variables, intrinsics, etc. that are in scope. It is invoked by pressing Ctrl+Space. As you start typing a name, the list is refined based on what you have typed; you can then use the arrow keys to choose from the list and press Enter to complete the definition. Content assist is useful when you don't remember the exact name of a function, or when a function has a long

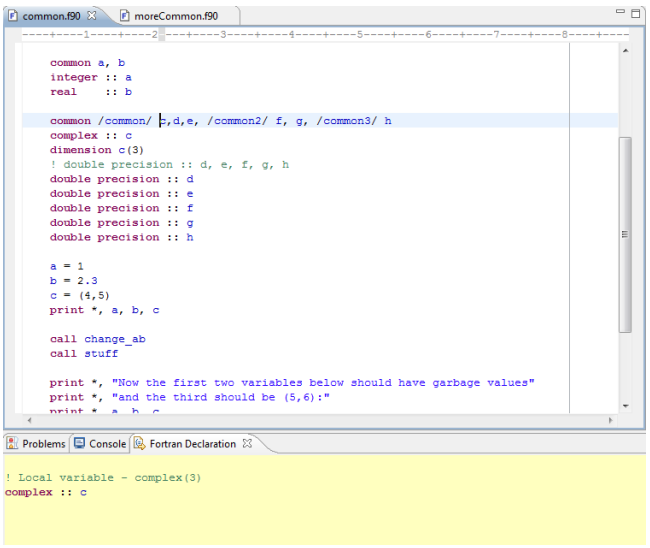
name and you don't want to type all of it. An example is shown below.



To use content assist, you will need to open the project properties, switch to the Fortran Analysis/Refactoring category, and check the "Enable Fortran content assist (Ctrl+space)" box. You will need to close and re-open any Fortran editors before you can start using content assist, however.

Fortran Declaration View

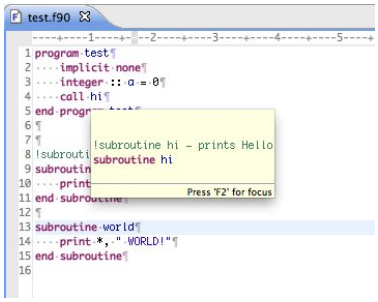
The Fortran Declaration view displays the statement where a variable, program, or subprogram is declared. If you place the text editor cursor within (or highlight) a variable name or the name of a subroutine in a CALL statement, the Fortran Declaration view will then display the actual declaration. For variables, the declaration view displays the variable declaration as well as a comment regarding the type of variable and other information, such as dimension information for an array as shown in the example below. For subprograms, it displays the FUNCTION or SUBROUTINE statement, including any comments that precede it.



Hover Tips

Hover tips display the same text as the Fortran Declaration view (described above); however, they display it in a popup window when the mouse cursor is hovered over an identifier.

Fortran hover tips are enabled in a project's analysis/refactoring properties. To turn them on or off, check the box labeled "Enable Fortran Hover Tips" in the project properties under Fortran General ►Analysis/Refactoring. (Note that Analysis/Refactoring must also be enabled.)



2.3 Search and Navigation

Open Declaration

The Open Declaration command allows you to quickly find the declaration of a particular variable or subprogram. In the Fortran editor, click on (or select) an identifier. Then either

- click on the Navigate menu, and select Open Declaration, or
- right-click on the identifier, and, and select Open Declaration, or
- press the F3 key on your keyboard.

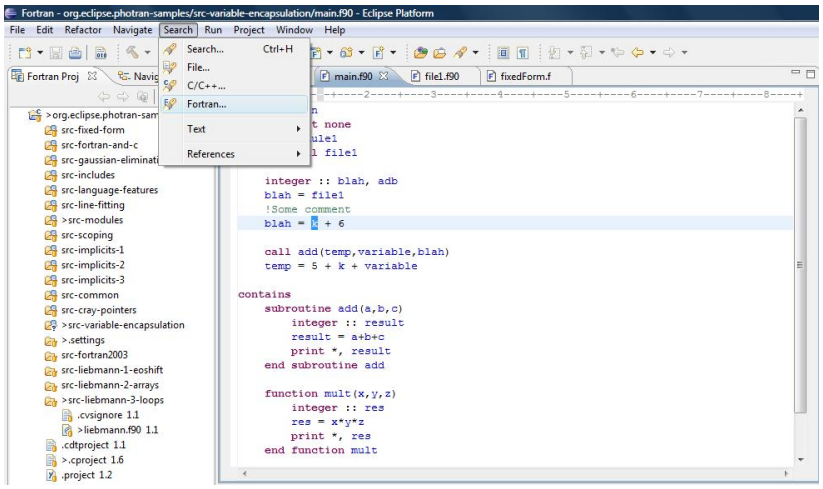
This feature is particularly useful when the declaration is in another file. For example, if your program contains a CALL statement which invokes a subroutine imported from a module in another file, invoking Open Declaration on the subroutine name in the CALL statement will open the module in a new editor, and the cursor will be located on the declaration of that subroutine.

Fortran Search

Fortran Search is a syntax-aware, multi-file search tool. Unlike textual searches, Fortran Search "understands" Fortran programs. It is used to search for a particular type of declaration (e.g., only modules, or only subroutines), or it can find all of the uses of a particular variable or subroutine. It also allows

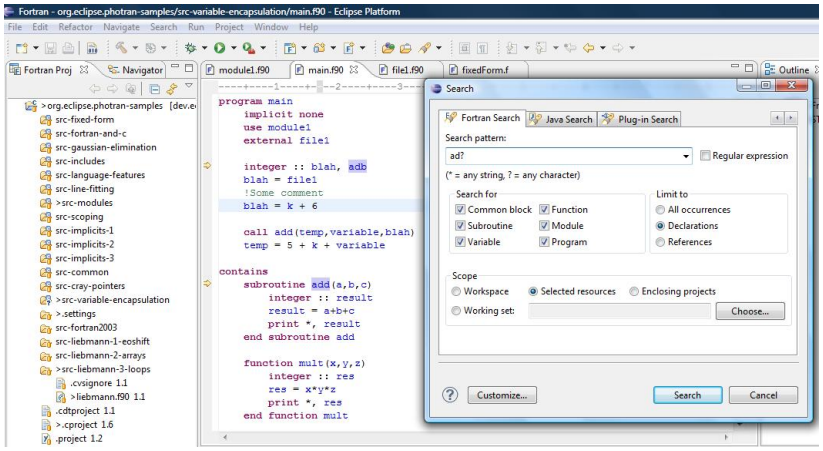
to you to limit the scope of the search (e.g., it can search every file in the workspace, or just a particular project).

In order to use Fortran Search, click on the Search menu and select Fortran...



This will open the Eclipse Search dialog, and the Fortran tab will be active. There, you can specify several things:

- The Search for frame allows you to specify whether it should search for subroutines, functions, modules, variables, etc.
- The Scope frame allows you to specify what file(s) will be searched.
- The Limit to frame allows you to specify whether you want to find declarations, references, or both.



Notice that only the declarations of variables/functions were found. Also note the use of the wild-card character(?) in the search box

Wildcards

Wildcards are special symbols used for pattern matching. For more information, click on Help ►Help Contents in Eclipse, and navigate to Workbench User Guide ►Reference User interface information ►Search.

- * - represents any number of any symbols. For example, a search for *d would match bad, rescued, d, assembled, etc.
- ? - represents any one single character. For example, a search for ?ad would match bad, mad, sad, dad, zad, etc.

Regular expressions are an even more advanced pattern matching languages. For more information, click on Help ►Help Contents in Eclipse and navigating to Workbench User Guide ►Getting started ►Basic tutorial ►Searching

Chapter 3

Refactoring

3.1 Introduction

What is Refactoring?

Refactorings are changes to a program that improve its internal design but do not change its behavior. These include minor, coding style changes (like using IMPLICIT NONE statements), code readability improvements (like replacing a variable named N with one called NUM_POINTS), performance improvements (like interchanging loops under certain conditions), and even larger-scale design changes (like moving a procedure from one module to another).

Although these types of changes can be done by hand, making them is often tedious and error-prone. Photran automates many such refactorings. For example, the Refactor ►Rename command can automatically locate the declaration(s) and uses of a particular subroutine, and change its name in all of those locations. It is "smart," too; if there is a subroutine named d and a variable named d in a different context, it won't confuse the two. Moreover, before making such a change, Photran will attempt to verify that the change is safe to make. For example, a subroutine A cannot be renamed to B if there is already a variable named B in a context where that subroutine is called.

For more information on refactoring, see M. Fowler, Refactoring: Improving the Design of Existing Code, Addison-Wesley, 1999.

Refactoring in Photran

Most refactorings can be accessed via the Refactor menu in the menu bar, as described below. However, the most common refactorings also have hotkeys (e.g., Alt+Shift+R for Rename; hotkeys are listed in the Refactoring menu next to each command). Also, most refactorings can be accessed by right-clicking in an editor and choosing Refactor from the popup menu.

Some refactorings (such as Introduce Implicit None and Replace Obsolete Operators) can be applied to several files at once. As described below, this

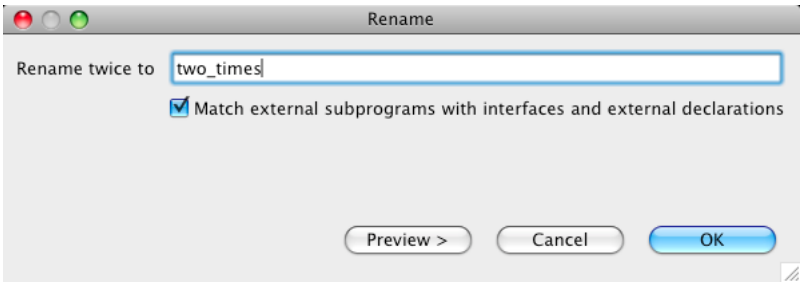
involves selecting one or more files in the Fortran Projects view, then right-clicking on any of the selected filenames and choosing Refactor from the popup menu.

- Clicking on a filename in the Fortran Projects view selects that file (and only that file).
- Ctrl+click (Command+click on Mac OS X) can be used to select or deselect additional files.
- To select a range of files, click on one filename, and Shift+click on a later filename; those files and all of the files in between will be selected as well.

Caution: Photran can only refactor free-format Fortran source code. It is not possible to refactor fixed-form code. Make sure that only free-form Fortran files are selected. The Refactor menu may not be available if any of the files are fixed-form or non-Fortran files.

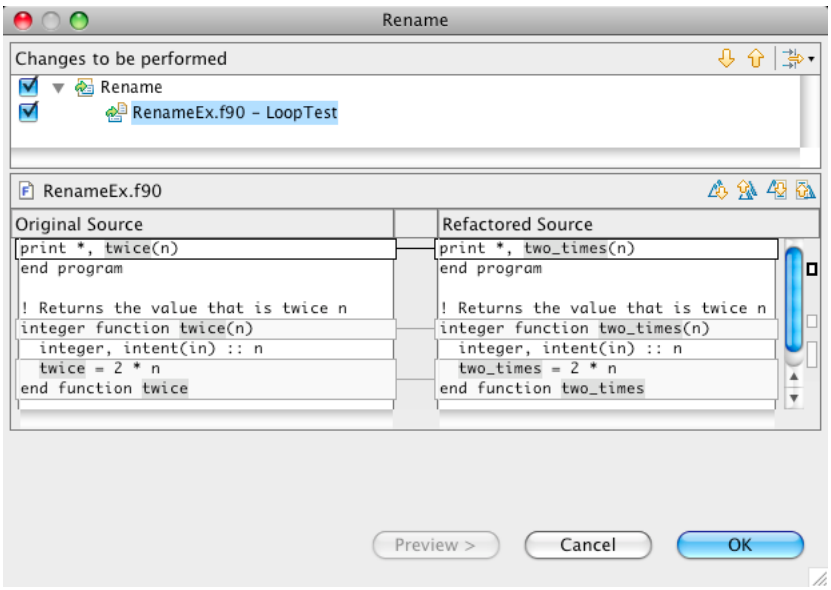
Rename

- Description: Rename is essentially a "smart" search and replace: It allows you to change the name of a variable, subprogram, etc. It correctly observes scoping and shadowing rules can also rename subprograms and module entities across files.
- Applies To:
  1. Local variables1,2
  2. Subprograms3 (including external and interface declarations)
  3. Derived types
  4. Module entities (variables and subprograms)
  5. Main programs
  6. Namelists
  7. Common blocks
  8. Block data subprograms
- Operation:
  1. Click on the name of a local variable, subprogram, etc.
  2. Click Refactor ►Rename... The Rename dialog will appear. Rename dialog
  3. Enter a new name for the variable/subprogram/etc.



4. If you are renaming an external subprogram or a subprogram declared in an interface block, you may want to (un)check the box labeled Match external subprograms with interfaces and external declarations. If this is checked, the refactoring will attempt to find all external subprograms, EXTERNAL statements, and subprogram declarations in INTERFACE blocks that have the given name, and they will all be renamed.
5. Click Preview to see what changes will be made, then click OK to apply them.

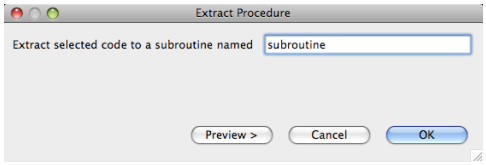
- Example:



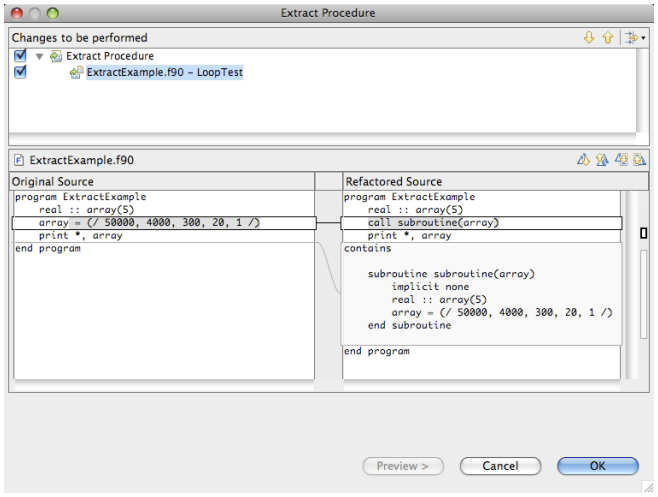
1. Dummy subprogram arguments cannot be renamed
2. Components of derived types cannot be renamed
3. Intrinsic subprograms and type-bound procedures (Fortran 2003) cannot be renamed

Extract Procedure

- Description: Extract Procedure removes a sequence of statements from a procedure, places them into a new subroutine, and replaces the original statements with a call to that subroutine. Any local variables used by those statements will be passed as parameters to the new procedure. This refactoring is generally used to make long procedures shorter.
- Applies To: A sequence of one or more action statements inside a procedure or main program.
- Operation:
  1. Select a sequence of one or more action statements in the editor. Be sure to include the newline following the last statement in the selection.
  2. Click Refactor ►Extract Procedure... The Extract Procedure dialog will appear. Extract Procedure dialog



  3. Enter a name for the new procedure that will be created.
  4. Click Preview to see what changes will be made, then click OK to apply them.
- Example:



Extract Local Variable

- Description: Extract Local Variable removes a subexpression from a larger expression and assigns that subexpression to a local variable, replacing the original subexpression with a reference to that variable. This refactoring is usually used to eliminate duplicated subexpressions or to introduce explanatory variable names into complex expressions.

- Caveats: The refactoring will only be allowed to proceed if extracting the subexpression will preserve the associativity and precedence of the original expression. This refactoring assumes that the extracted expression has no side effects; it does not check whether moving the computation of the extracted expression will change the behavior of the program.

- Operation:

  1. Select an expression in the editor.
  2. Click Refactor ►Extract Local Variable... The Extract Local Variable dialog will appear.
  3. Enter the type and name for the new local variable that will be created.
  4. Click Preview to see what changes will be made, then click OK to apply them.

3.2 Subprogram Refactorings

Add Subroutine Parameter

- Description: This refactoring will add a parameter to a subroutine.
- Applies To: A subroutine.
- Operation:
  1. Select the subroutine name or subroutine statement
  2. Click Refactoring ►Subprogram ►Add Subroutine Parameter (or press Shift+Alt+P)
  3. Use the input page to choose the type, intent, and name of the parameter.
  4. Type in numbers for the default value and position.
  5. Click preview to view the changes without applying, or OK to apply the changes.
- Example:

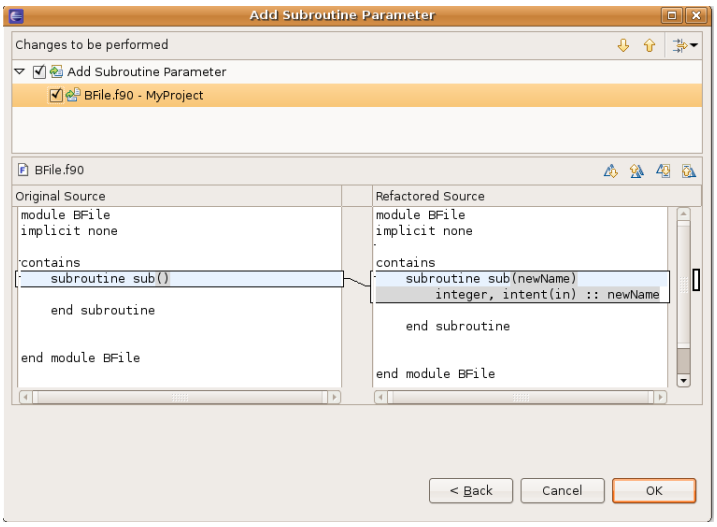


Figure 3.1:

Example of Add Subroutine Parameter refactoring.

Permute Subprogram Arguments

- Description: This refactoring will change the order of the arguments to a subroutine and adjust all call sites accordingly.
- Applies To: All files calling the selected subroutine.
- Operation:
  1. Select the subroutine name or subroutine statement
  2. Click Refactoring ►Subprogram ►Change Subroutine Signature.
  3. Use the input page's up and down buttons to rearrange the order of the arguments for the subroutine.
  4. Click preview to view the changes without applying, or OK to apply the changes.
- Example: Example of Change Subroutine Signature refactoring.

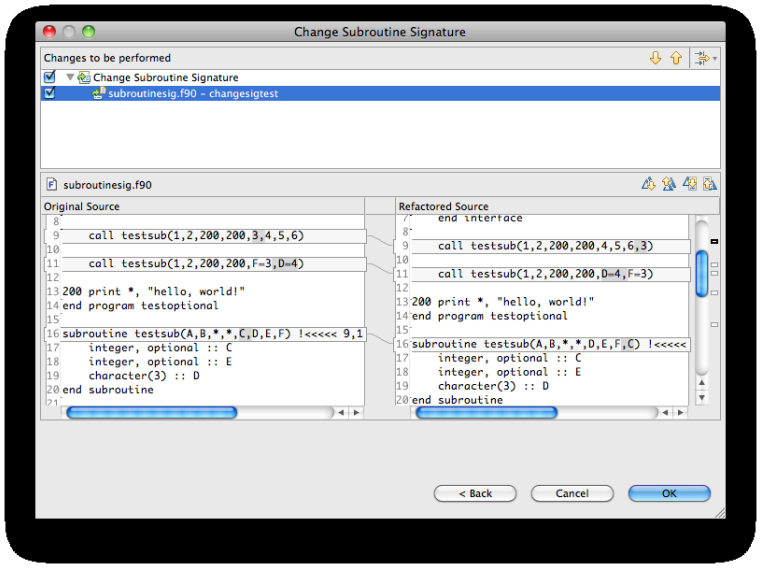


Figure 3.2:

Safe Delete

- Description: This refactoring will remove all Internal Subprograms from a given Host. The refactoring fails if there are any references to the subprogram.
- Applies To: A file.
- Operation:
  1. Select the subroutine name or subroutine statement to remove
  2. Click Refactoring ►Subprogram ►Safe Delete
  3. Click preview to view the changes without applying, or OK to apply the changes.
- Example: see figure 3.3

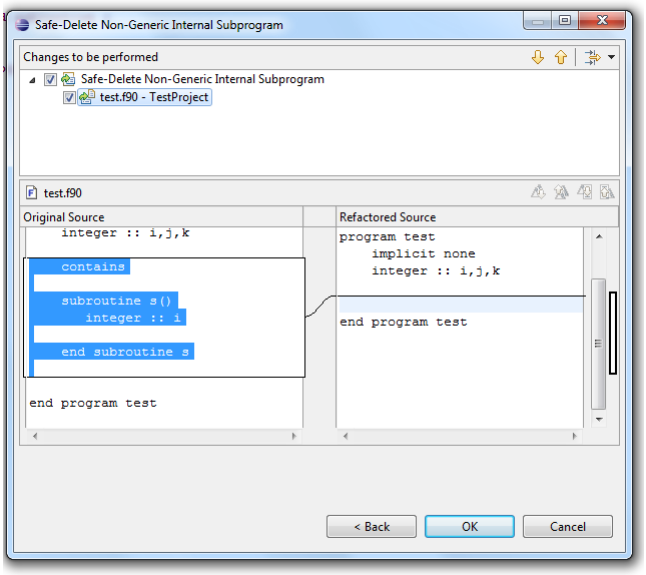


Figure 3.3: Example of the Safe-Delete Internal Subprograms refactoring.

3.3 Module Refactorings

Make Private Entity Public

- Description: Changes a module variable or subprogram from PRIVATE to PUBLIC visibility, and checks that it won't conflict with any existing name where that module is USEd.
- Applies To: Variables, subroutines, functions.
- Does Not Apply To: Intrinsics, Externals, Interfaces.
- Operation:
  1. Select the name of the private entity you wish to make public.
  2. Choose Refactor ►Make Private Entity Public.
  3. Click Preview to see what changes will be made, then click OK to apply them.
- Example: see figure 3.4 and 3.5

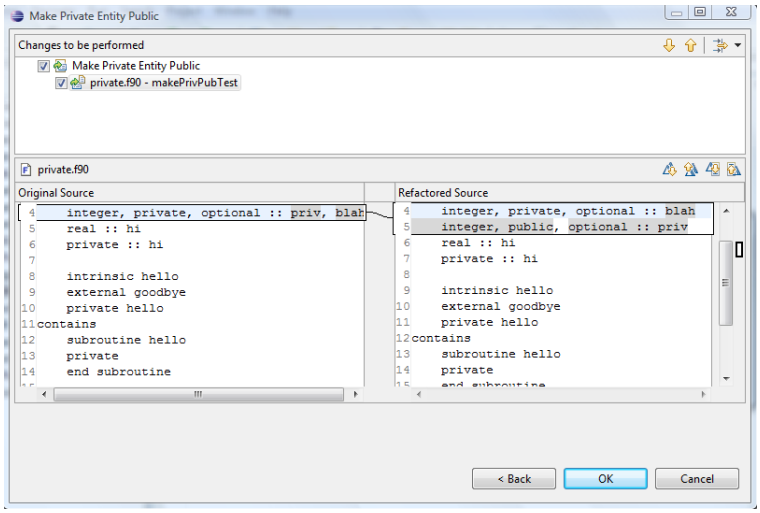


Figure 3.4: Example of make private entity public refactoring.

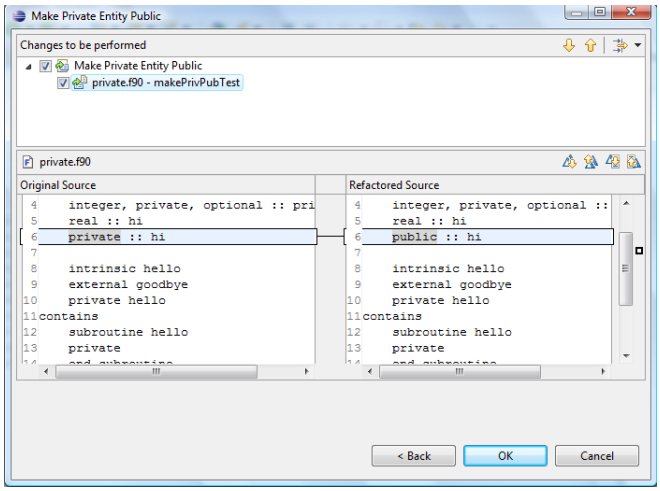


Figure 3.5: Example 2 of make private entity public refactoring.

Encapsulate variable

- Description: Encapsulate variable creates getter and setter methods for the selected variable in the module where it is defined and changes variable's visibility to private. It also replaces all uses of that variable in all files to use getter and setter method calls1.
- Applies To:
  1. Variables defined in the module.
  2. Variables of all basic as well as user-defined types
- Does Not Apply To:
  1. Arrays
  2. Variables that are not defined in a module
  3. Parameters (i.e. integer, parameter :: CANNOT_ENCAPSULATE)
  4. Pointers (i.e. real, pointer :: CANNOT_ENCAPSULATE)
  5. Targes (i.e. integer, target :: CANNOT_ENCAPSULATE)
- Operation:
  1. Click on or select the name of variable you want to encapsulate.
  2. Click Refactor ►Encapsulate Variable. The Encapsulate Variable dialog will appear.

     Getter and setter name menu for encapsulating variable temp

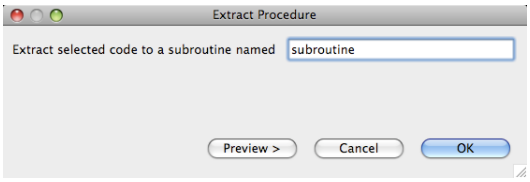


Figure 3.6:

3. Enter names for getter and setter methods. You will be warned if the names that you want to assign to your getter and setter methods will be conflicting with some other identifier in any of the involved files.
4. Click Preview to see what changes will be made, then click OK to apply them.

- Example: see figure 3.7 and 3.8

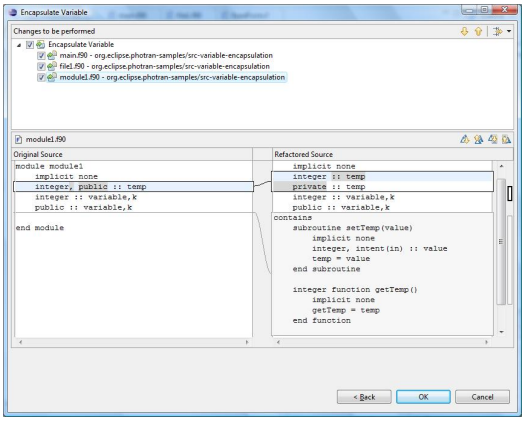


Figure 3.7: Simple example of Encapsulate Variable refactoring

1. If a variable is used as a parameter to a function/subroutine call, and that function changes the value of the variable as a side-effect, that change will not be preserved.

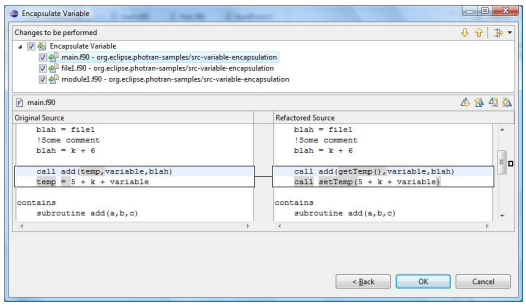


Figure 3.8: Another simple example of Encapsulate Variable refactoring

3.4 Use Statement Refactorings

Add ONLY Clause to USE Statement

- Description: Creates a list of the symbols that are being used from a module, and adds it to the USE statement.
- Applies To: All modules containing public definitions.
- Does not apply to: Empty modules or modules with only private entities.
- Operation:
  1. Select the name of the module in the USE statement you wish to add an ONLY clause to.
  2. Choose Refactor ►Add ONLY clause to USE statement.
  3. Select which module entities you wish to include in the ONLY list. Any entities in an existing ONLY list will already be selected and can be deselected to be removed. Add ONLY to USE dialog

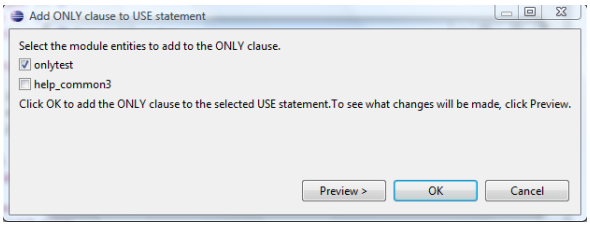


Figure 3.9:

  4. Click Preview to see what changes will be made, then click OK to apply them.
- Example: see figure 3.10

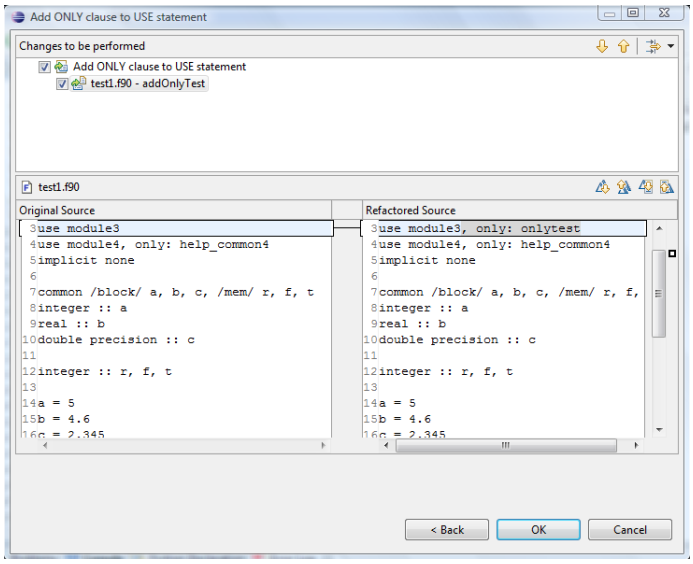


Figure 3.10: Example of Add ONLY clause to USE statement refactoring.

Minimize ONLY List

- Description: Deletes symbols that are not being used from the ONLY list in a USE statement.
- Applies To: USE statements with an ONLY clause.
- Operation:
  1. Select the name of the module in the USE statement you wish to minimize the ONLY list for.
  2. Choose Refactor ►Minimized ONLY list for Selected module.
  3. Click Preview to see what changes will be made, then click OK to apply them.
- Example: see figure 3.11

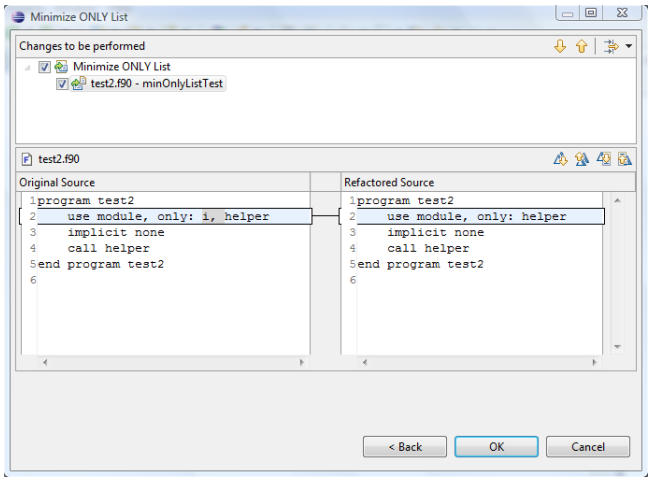


Figure 3.11: Example of Minimize ONLY list refactoring.

3.5 Common Block Refactorings

Make COMMON Variable Names Consistent

- Description: Fortran allows different definitions of a COMMON block to give the same variable different names. This is confusing. This refactoring gives the variables the same names in all definitions of the COMMON block.
- Applies To: All COMMON blocks with a valid name.
- Does not apply to: COMMON blocks with a NULL name.
- Operation:
  1. Select the name of the COMMON block in the editor which you wish to make variable names consistent for.
  2. Choose Refactor ►Make COMMON Variable Names Consistent form the menu bar.
  3. Enter the new names which you wish to give the COMMON variables. The default new names are the original names in the selected block with "_common" appended. Make common var names consistent dialog
  4. Click Preview to see what changes will be made, then click OK to apply them.
- Example Example of Make COMMON variable names consistent refactoring.

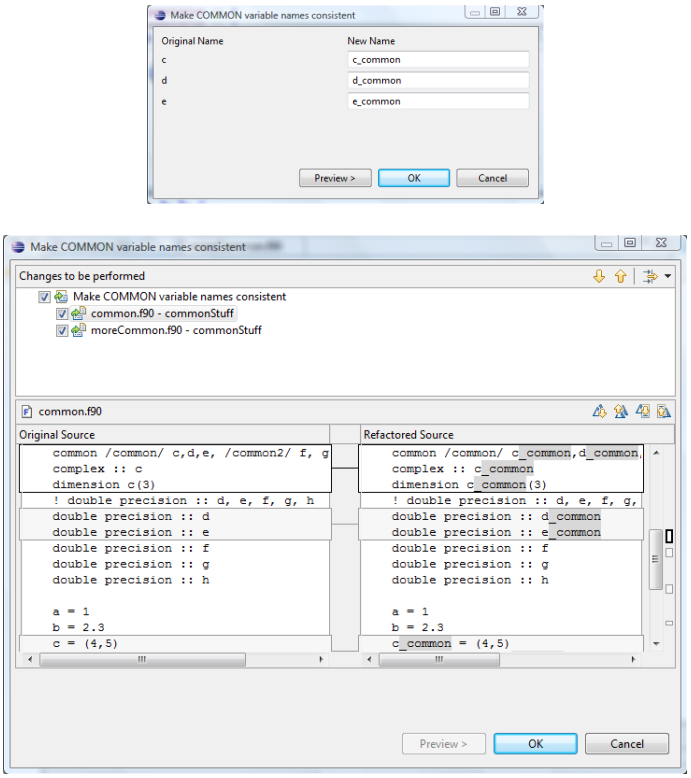


Figure 3.12:

Move Saved Variables to Common Block

- Description: Move Saved Variables to Common Block creates a common block for all "saved" variables of a subprogram. Declarations of these variables in the subprogram are transformed such that they are no longer "saved". The generated common block is declared both in the main PROGRAM and in the affected subprogram. Variables placed in the common block are renamed such that they do not conflict or shadow other variables. The current implementation assumes that the subprogram is in the CONTAINS section of the PROGRAM.
- Applies To: Subprograms.
- Operation:
  1. Click on the declaration statement of a subprogram.
  2. Click Refactor ►Move Saved Variables to Common Block.
  3. Click Preview to see what changes will be made, then click OK to apply them.

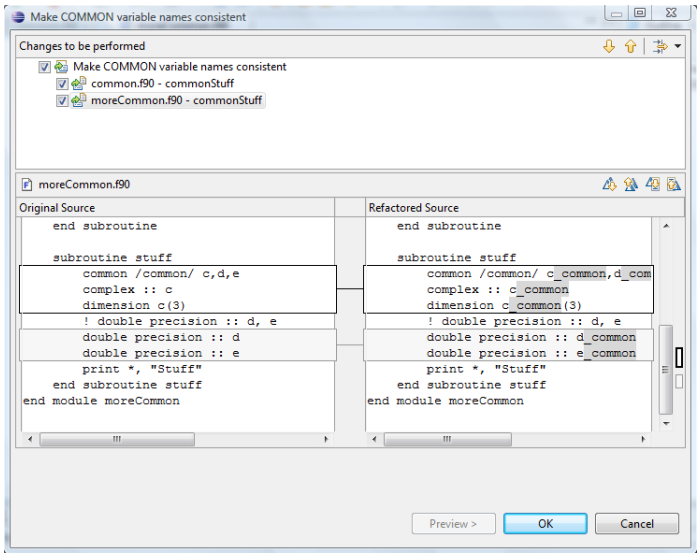


Figure 3.13: Example of Make COMMON variable names consistent refactoring.

- Example:

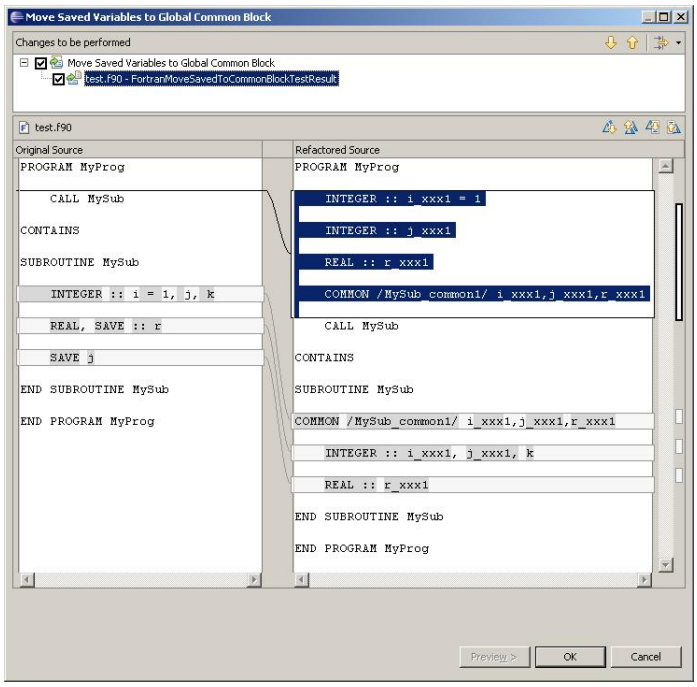


Figure 3.14:

3.6 Loop Refactoring

Interchange loops

- Description: Swaps inner and outer loops of the selected nested do-loop1. This refactoring merely switches the inner and outer do-headers. It will not make any changes to the body of the loop.
- Applies To: Selected nested do-loop
- Operation:
  1. Select the nested loops you wish to interchange
  2. Click Refactor ►Interchange Loops. The Interchange loops dialog will appear.
  3. Click Preview to see what changes will be made, then click OK to apply them.
- Example:

  1. In order for refactoring to work correctly, there must be no statements before the second loop. If such statements exist, correctness of the refactoring is not guaranteed.

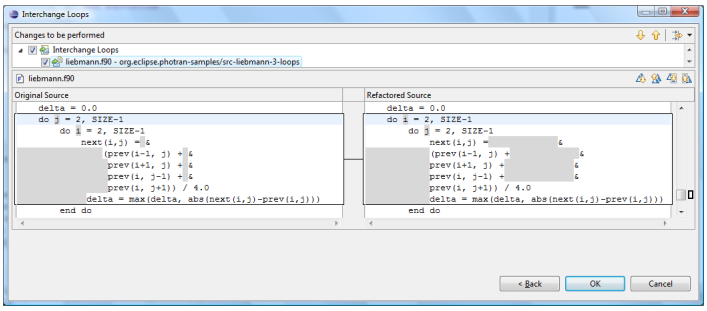


Figure 3.15: Simple example of Interchange Loops refactoring

Fuse Loops

- Description: Takes two do-loops, normalizes their bounds, and finally puts the loop bodies in a single do-loop.

- Applies To: Find the two do-loops you want to fuse, and select only the first one1. If there are any statements in between the two loops, the statements will remain after the fused do-loop. In order for the loops to be compatible, the loops need to have the same number of iterations (the bounds do not need to be the same, just the number of times the body of the loop is accessed).

- Operation:

  1. Select the first do-loop to be fused.

  2. Click Refactor ►Loop Fusion.

  3. Click Preview to see a comparison view of the changes made, and OK to apply the changes.

- Example: 1. The refactoring will find the next listed do-loop in your code (even with other lines of code in between them).

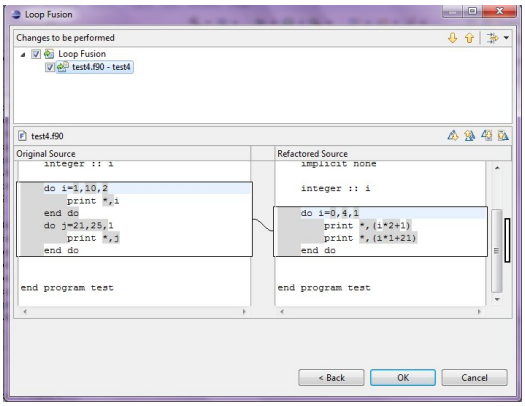


Figure 3.16: Example of Loop Fusion with loops of different bounds

Reverse Loop

- Description: Takes an incrementing or decrementing loop, swaps the lower and upper bounds, and negates the step.
- Applies To: Selected do-loop
- Operation:
  1. Select full do-loop or do-loop header
  2. Click Refactor ►Reverse Loop.
  3. Click Preview to see a comparison view of the changes made, and OK to apply the changes.
- Example:

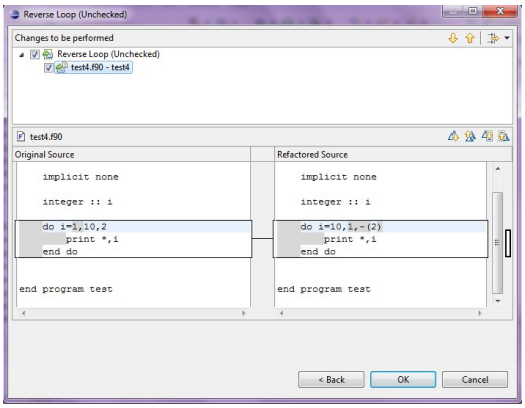


Figure 3.17: Example of Reverse Loop Refactoring

Unroll Loop

- Description: Takes the selected do-loop and either completely or partially unrolls it. This will also optionally include a conditional statement to make sure the loop stays in bounds.
- Applies To: A do-loop that doesn't contain labels, and never writes to the index variable (Ex: read(* ,*) indexVar)
- Operation:
  1. Select the loop to unroll (or just the header).
  2. Click Refactoring ►Loop Unrolling.
  3. Select either the number of times you would like to unroll the loop, or check the "Complete unrolling" box. Uncheck the "Include bounds checking" box if you don't want a conditional statement to ensure proper loop bounds in numbered loop unrolling.
  4. Click Preview to see the changes this refactoring will make, and OK to make the changes.
- Example(Numbered):

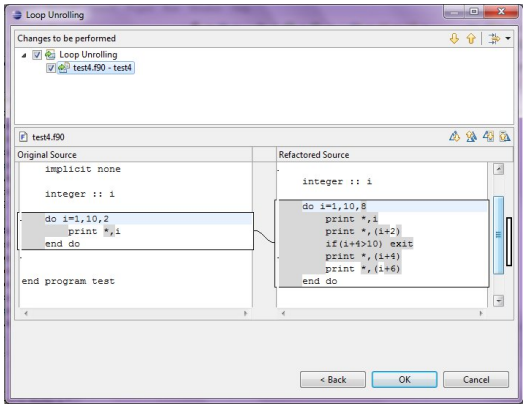


Figure 3.18: Preview of unrolling a loop four times including a bounds check

- Example(Complete):

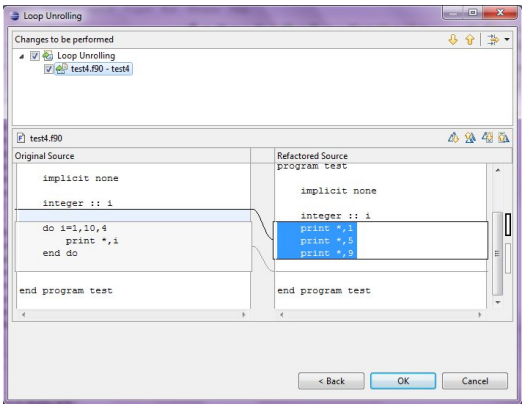


Figure 3.19: Preview of completely unrolling a loop

Tile Loop

- Description: This refactoring takes a double nested do-loop, and creates a nested do-loop with four levels of depth. Instead of iterating through a two dimensional array (for example) by going through each row, it will loop over smaller tile blocks
- Applies To: A double nested do-loop where both step values are 1.
- Inputs: o Tile Size: The size of the accessing block or tile. So, if the tile size is 3, an array will be accessed in 3x3 blocks. o Tile Offset: Adjusts where the blocks start (all the data will always be covered no matter where it starts).
- Operation:
  1. Select the whole nested do-loop, or the header of the top nested do-loop.
  2. Click Refactoring ►Loop Tiling.
  3. Enter a tile step and tile offset, then click preview to view changes, or OK to apply changes.
  4.
- Example:

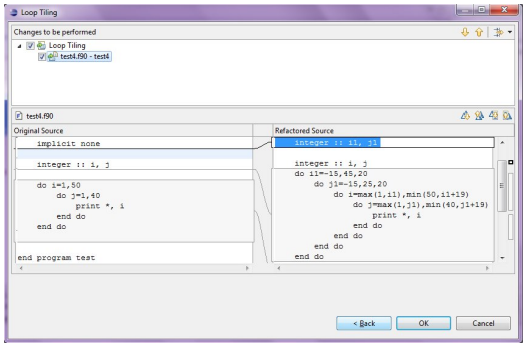


Figure 3.20: Example of loop tiling with a tile size of 20 and a tile offset of 5.

3.7 Refactorings to Remove Obsolete Language Features

Remove Arithmetic If Statements

- Description: This refactoring will remove all arithmetic if statements, a feature that became obsolete in Fortran 90, from a single file or all the files in a project.
- Applies To: A project or file.
- Operation:
  1. Select the project or file with arithmetic if statements to remove.
  2. Click Refactoring►Obsolete Language Features ►Remove Arithmetic If Statements
  3. Click preview to view the changes without applying, or OK to apply the changes.
- Example:

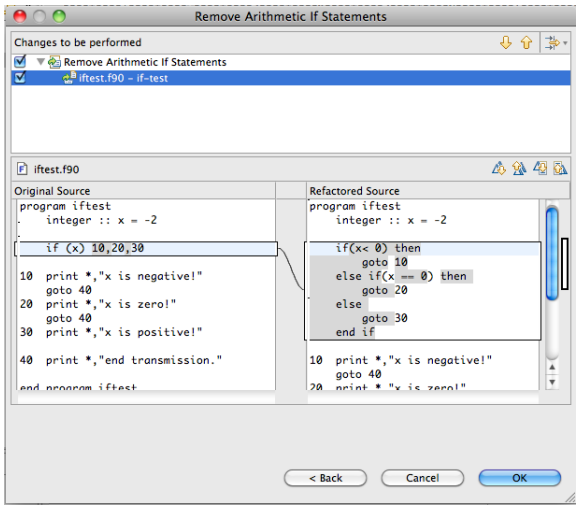


Figure 3.21: Example of the Remove Arithmetic If Statement refactoring.

Remove Assigned Goto

- Description: This refactoring will remove all assigned goto statements and replace them with case blocks.
- Applies To: All assigned goto statements.
- Operation:
  1. Click Refactoring ►Obsolete Language Features ►Remove Assigned Goto.
  2. Choose Yes if you want to add a default case, otherwise choose No.
  3. Click preview to view the changes without applying, or OK to apply the changes.
- Example: Example of the Remove Arithmetic If Statement refactoring.

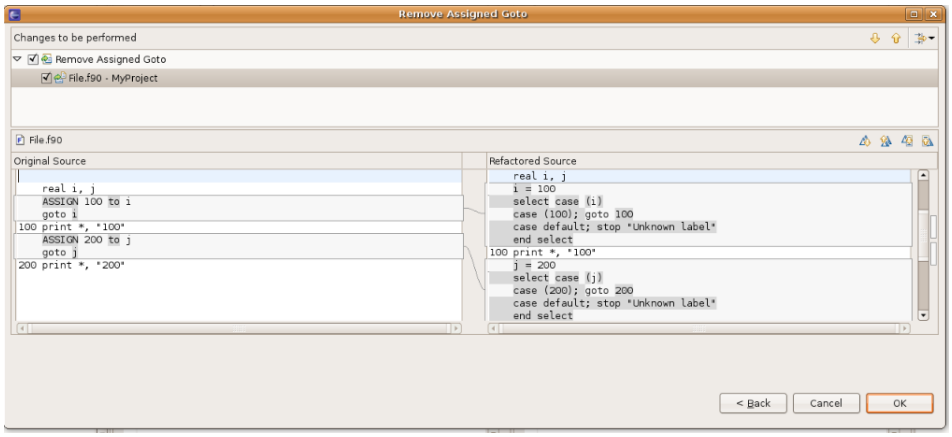


Figure 3.22:

Remove Branch to End If

- Description: Removes the branch to END IF statements. The GOTO statements carry outbranching. Branching to end if is replaced with branching to CONTINUE statement that immediately follows the END IF statement.
- Applies To: Selected end if statement
- Operation:
  1. Select an end if statement with a statement label
  2. Click Refactor ►Obsolete Language Features ►Remove Branch to End If
  3. Click Preview to see what changes will be made, then click OK to apply them.
- Example:

  Simple example of Remove Branch to End If refactoring

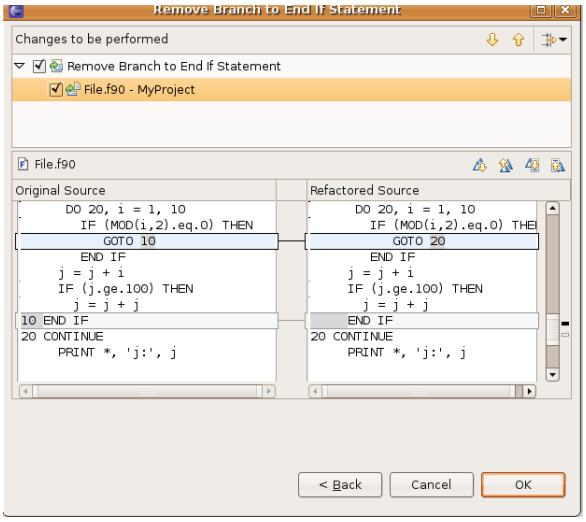


Figure 3.23:

Replace Character* with Character(len=)

- Description: Replaces character*n declaration with character(len=n) declaration.
- Applies To: All character declarations
- Operation:
  1. Select one of the character declaration statements
  2. Click Refactor ►Obsolete Language Features ►Replace Character* with Character(len=)... The Replace Character* with Character(len=) will appear.
  3. Click Preview to see what changes will be made, then click OK to apply them.
- Example:

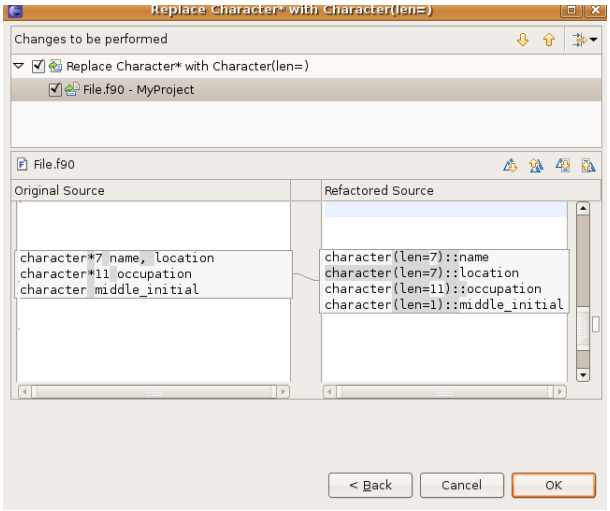


Figure 3.24: Simple example of Replace Character* with Character (len=) refactoring

Remove Computed Goto

- Description: Replaces computed goto statement with select case statement
- Applies To: Selected computed goto statement
- Operation:
  1. Select the computed goto statement you wish to remove
  2. Click Refactor ►Obsolete Language Features ►Remove Computed Goto. The Remove Computed Goto dialog will appear.
  3. Click Preview to see what changes will be made, then click OK to apply them.
- Example:

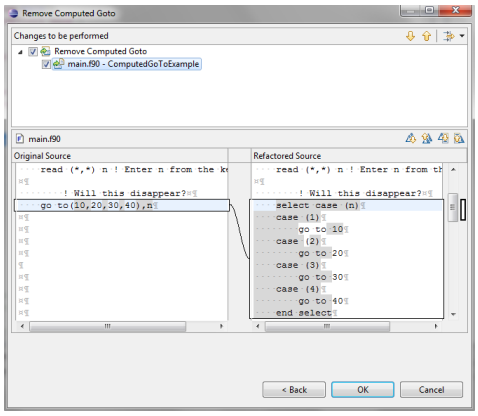


Figure 3.25: Simple example of Remove Computed Goto refactoring

Replace Obsolete Operators

- Description: Replace Obsolete Operators replaces all uses of old-style comparison operators (such as .LT. and .EQ.) with their newer equivalents (symbols such as ¡ and ==). and adds explicit declarations for all variables that were previously declared implicitly.
- Applies To: All uses of the following operators in one or more files: .LT. .LE. .EQ. .NE. .GT. .GE.
- Operation:
  1. This is a multiple-file refactoring.
     - To Replace Obsolete Operators in a single file, open the file in the editor and choose Refactor ►Obsolete Language Features ►Replace Obsolete Operators from the menu bar.
     - To Replace Obsolete Operators in multiple files, select the files in the Fortran Projects view, right-click on any of the selected filenames, and choose Refactor ►Obsolete Language Features ►Replace Obsolete Operators from the popup menu.
  2. Click Preview to see what changes will be made, then click OK to apply them.
- Example:Example of the Replace Obsolete Operators refactoring

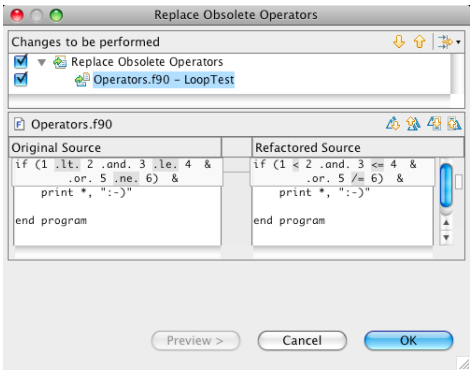


Figure 3.26:

Replace Old-Style Do Loops

- Description: Replaces old-style do loops with the modern style of do loops
- Applies To: All old-style do loops
- Operation:
  1. Click Refactor ►Obsolete Language Features ►Remove Old-Style Do Loops.
  2. Click Preview to see what changes will be made, then click OK to apply them.
- Example:

  Simple example of Remove Old-Style Do Loop refactoring

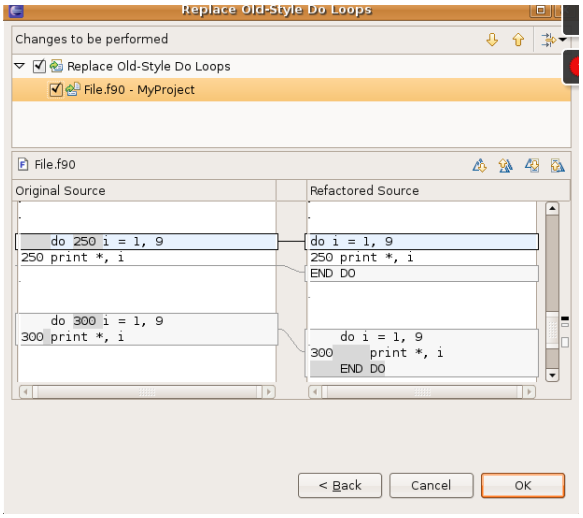


Figure 3.27:

Remove Pause Statement

- Description: Replaces pause statement with empty print and read statements
- Applies To: Selected pause statement
- Operation:
  1. Select the pause statement you wish to remove
  2. Click Refactor ►Obsolete Language Features ►Remove Pause Statement.
  3. Click Preview to see what changes will be made, then click OK to apply them.
- Example: Simple example of Remove Pause Statement refactoring

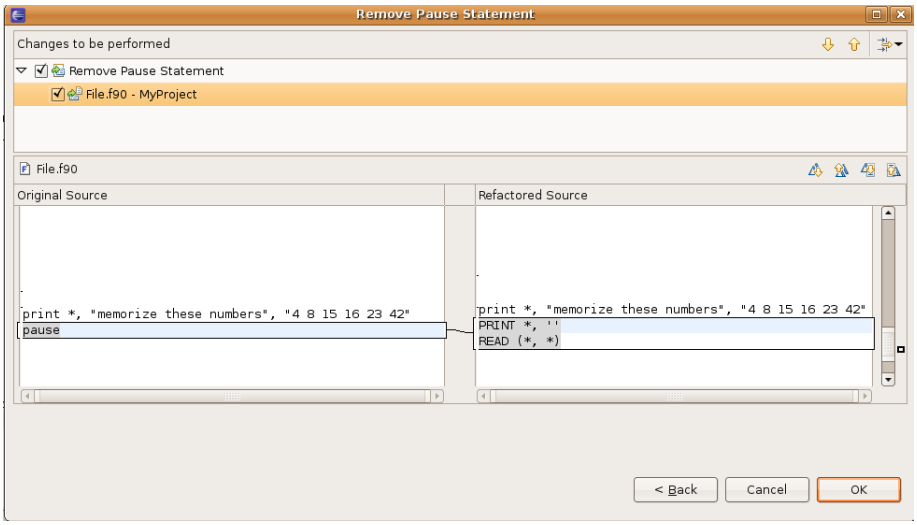


Figure 3.28:

Remove Real/Double Precision Loop Counter

- Description: Transforms a do or do while loop with control to a do or do while loop without control.
- Applies To: Selected loop
- Operation:
  1. Select the loop you wish to transform
  2. Click Refactor ►Obsolete Language Features ►Remove Real/Double Precision Loop Counter.
  3. Choose either do loop or do while loop.
  4. Click Preview to see what changes will be made, then click OK to apply them.
- Example: Simple example of Remove Real/Double Precision Loop Counter refactoring

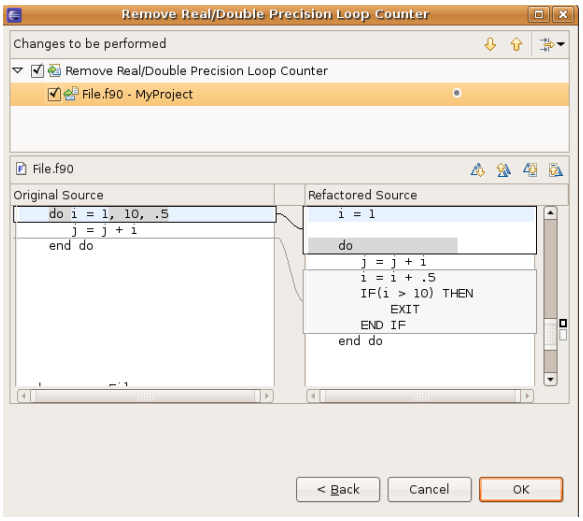


Figure 3.29:

3.8 Refactorings to Improve Coding Style

Add Identifier to End Statement

- Description: This refactoring will add the program/subprogram name to an end statement.
- Applies To: A file or project.
- Operation:
  1. Select the file or project.
  2. Click Refactoring ►Coding Style ►Add Identifier to End Statement
  3. Click preview to view the changes without applying, or OK to apply the changes.
- Example: Example of Add Identifier to END statement refactoring.

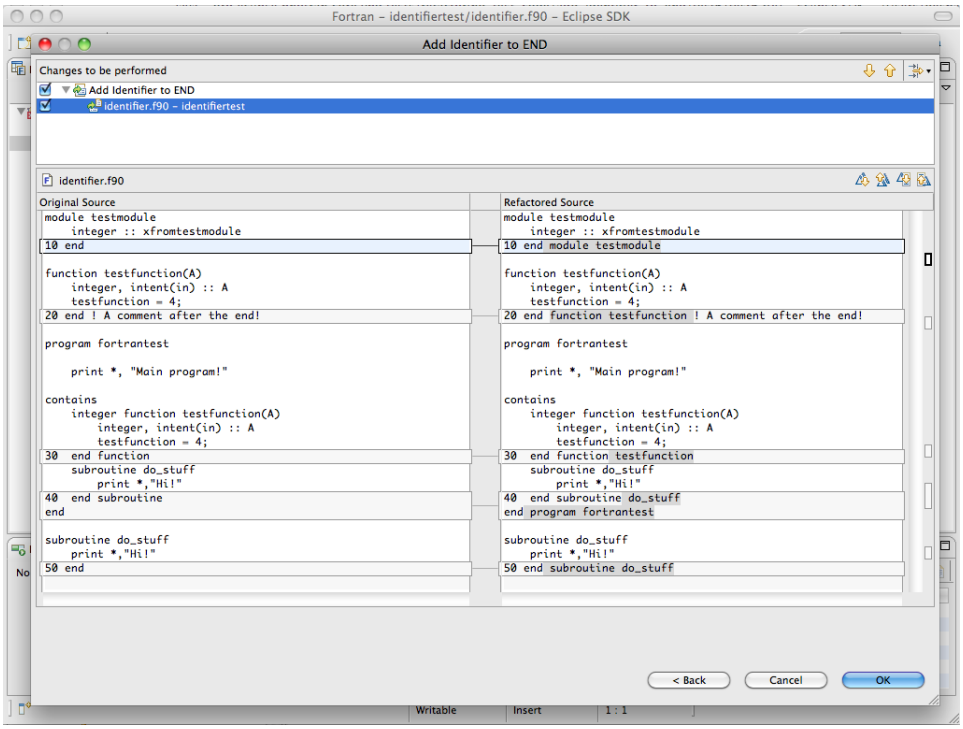


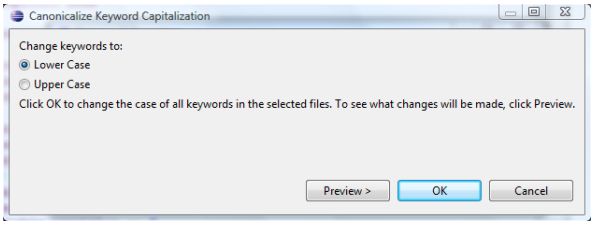
Figure 3.30:

Change Keyword Case

- Description: Makes all applicable keywords the same case throughout the selected Fortran program files.
- Applies To: All keywords except those listed below.
- Does not apply to:
  1. Identifiers
  2. All constants except for integer constants and real constants
- Operation:
  1. This is a multiple-file refactoring.
     - To Change Keyword Case in a single file, open the file in the editor and choose Refactor ▶Canonicalize Keyword Capitalization from the menu bar.
     - To Introduce Implicit None in multiple files, select the files in the Fortran Projects view, right-click on any of the selected

filenames, and choose Refactor ►Change Keyword Case from the popup menu.

2. Select Upper or Lower Case Canonicalize keyword capitalization dialog.



3. Click Preview to see what changes will be made, then click OK to apply them.

- Example Example of Change Keyword Case refactoring.

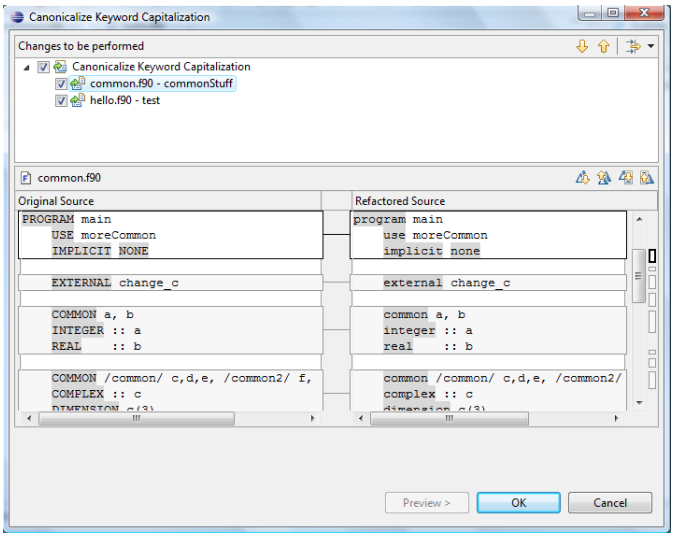


Figure 3.31:

Convert Between If Statement and If Construct

- Description: Converts a simple if statement to an if construct with a then block and possible else block.
- Applies To: Selected if statement.
- Operation:
  1. Refactor ►Coding Style ►Convert Between If Statement and If Construct
  2. Select whether or not you want an empty else block.
  3. Click Preview to see what changes will be made, then click OK to apply them.
- Example: Example of the Convert Between If Statement and If Construct refactoring

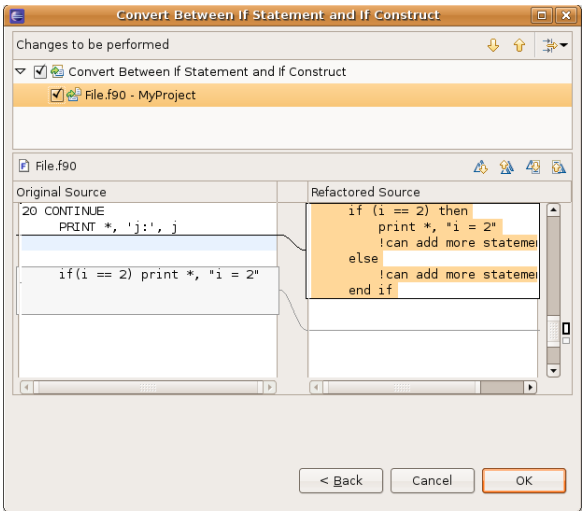


Figure 3.32:

Convert Data Statement to Parameter Statements

- Description: When a variable declared in a DATA statement is intended to be a constant, Data to Parameter can be used to change it to a variable with the PARAMETER attribute. Using the PARAMETER attribute makes it more clear which variables are constants and which ones are not; it can also result in performance gains, since it may allow an optimizing compiler to replace some variable accesses with the constant value.

- Applies To: All main programs, subprograms, and modules in one or more files.

- Operation:

  1. This is a multiple-file refactoring.
     - To transform variables declared as data in variables declared with parameter attribute in a single file, open the file in the editor and choose Refactor ►Data To Parameter from the menu bar.
     - To transform variables declared as data in variables declared with parameter attribute in multiple files, select the files in the Fortran Projects view, right-click on any of the selected filenames, and choose Refactor ►Data To Parameter from the popup menu.
  2. Click Preview to see what changes will be made, then click OK to apply them.

- Example: Example of the Data To Parameter refactoring

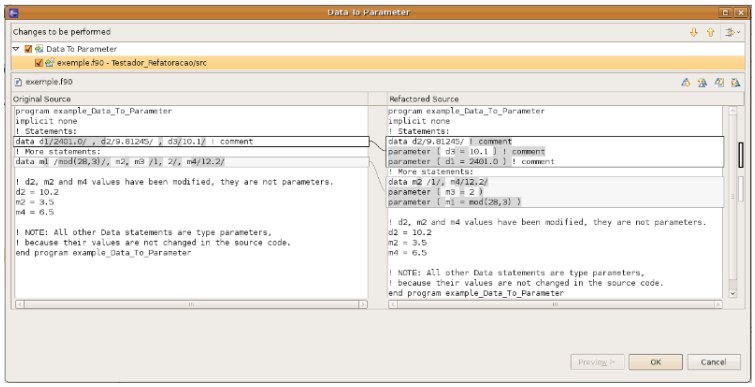


Figure 3.33:

Introduce Implicit None

- Description: Adds IMPLICIT NONE statements to a file and adds explicit declarations for all variables that were previously declared implicitly.
- Applies To: All main programs, subprograms, and modules in one or more files.
- Operation:
  1. This is a multiple-file refactoring.
     - To Introduce Implicit None in a single file, open the file in the editor and choose Refactor ►Introduce Implicit None from the menu bar.
     - To Introduce Implicit None in multiple files, select the files in the Fortran Projects view, right-click on any of the selected filenames, and choose Refactor ►Introduce Implicit None from the popup menu.
  2. Click Preview to see what changes will be made, then click OK to apply them.
- Example: Example of the Introduce Implicit None refactoring

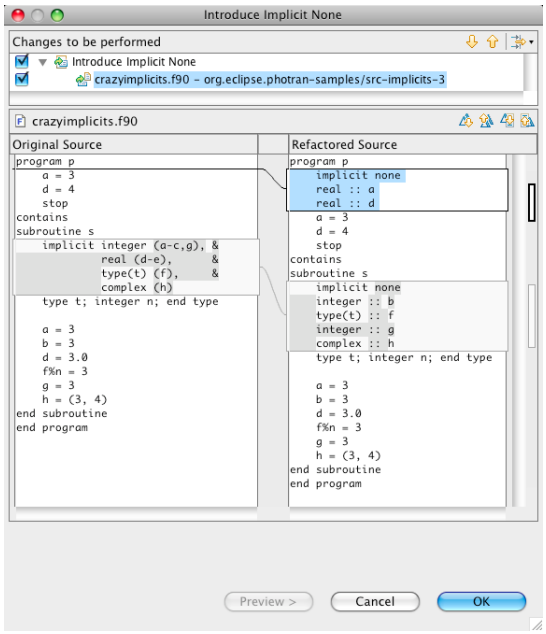


Figure 3.34:

Make Save Attributes Explicit

- Description: Makes Save Attributes Explicit.
- Applies To: The entire program.
- Operation:
  1. Refactor ►Coding Style ►Make Save Attributes Explicit
  2. Click Preview to see what changes will be made, then click OK to apply them.
- Example: Example of the Make Save Attributes Explicit refactoring

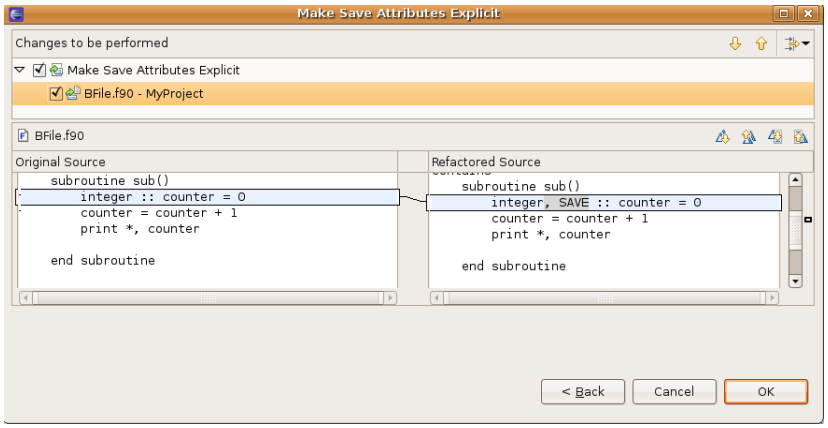


Figure 3.35:

Remove Unreferenced Labels

- Description: Removes any unreferenced labels.
- Applies To: All unreferenced labels.
- Operation:
  1. Refactor ►Coding Style ►Remove Unreferenced Labels
  2. Click Preview to see what changes will be made, then click OK to apply them.
- Example: Example of the Remove Unreferenced Labels refactoring

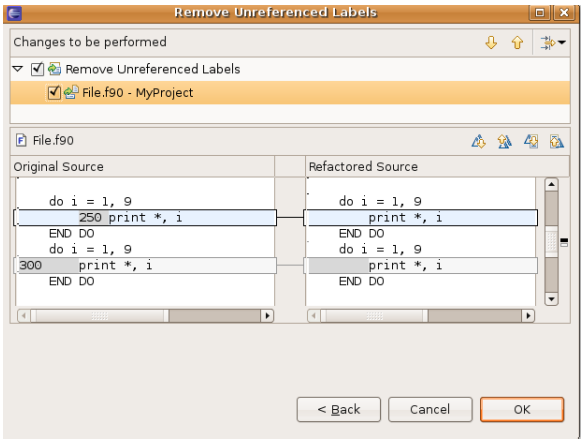


Figure 3.36:

Remove Unused Variables

- Description: Remove Unused Variables removes declarations of local variables that are never used.
- Applies To: All main programs, subprograms, and modules in one or more files.
- Operation:
  1. This is a multiple-file refactoring.
     - To Remove Unused Variables in a single file, open the file in the editor and choose Refactor ►Remove Unused Variables from the menu bar.
     - To Remove Unused Variables in multiple files, select the files in the Fortran Projects view, right-click on any of the selected filenames, and choose Refactor ►Remove Unused Variables from the popup menu.
  2. Click Preview to see what changes will be made, then click OK to apply them.
- Example: Example of the Remove Unused Variables refactoring

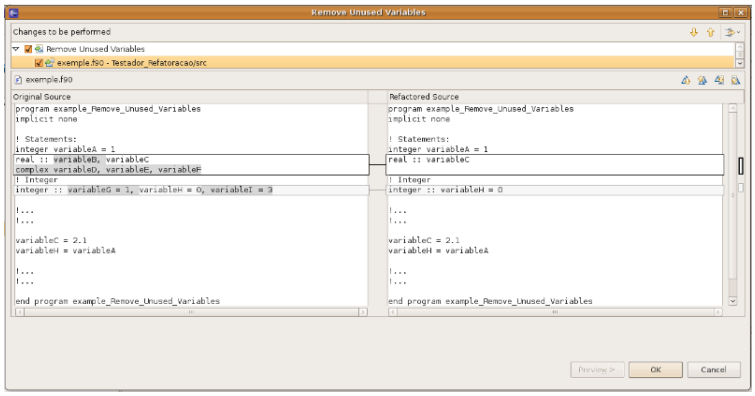


Figure 3.37:

Standardize Statements

- Description: Standardize Statements rewrites all variables declarations, so that
  - there is only one variable declaration per line, and
  - every variable declaration contains a double colon (::).

  This is intended to make the code more readable.
- Applies To: All main programs, subprograms, and modules in one or more files.
- Operation:
  1. This is a multiple-file refactoring.
     - To Standardize Statements in a single file, open the file in the editor and choose Refactor ►Standardize Statements from the menu bar.
     - To Standardize Statements in multiple files, select the files in the Fortran Projects view, right-click on any of the selected filenames, and choose Refactor ►Standardize Statements from the popup menu.
  2. Click Preview to see what changes will be made, then click OK to apply them.
- Example: Example of the Standardize Statements refactoring

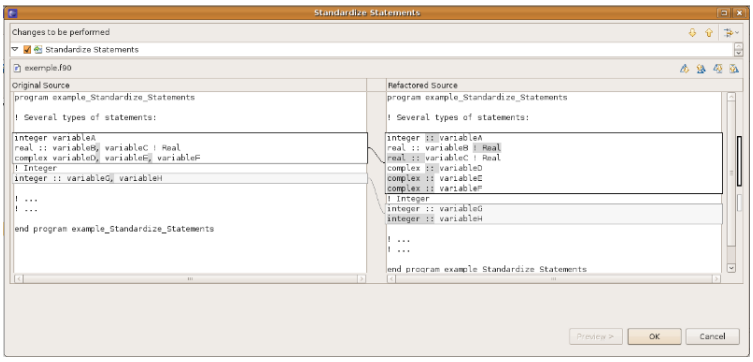


Figure 3.38:

3.9 Debugging

These refactorings are used to debug your program.

Highlight Variable Accesses

- Description: Creates a file of your code where the variables are highlighted according to whether they are read or write variables.
- Applies To: The entire program.
- Operation: Refactor ►Debugging ►Highlight Variable Accesses
- Example: Example of the Highlight Variable Accesses refactoring

Display Symbol Table for Current File

- Description: Creates file with a list of all the variables. Lists the name and type of variable and the line where the variable was declared.
- Applies To: The entire program.
- Operation: Refactor ►Debugging ►Display Symbol Table for Current File
- Example: Example of the Display Symbol Table for Current File refactoring

Find All Declarations in Scope

- Description: Gives a list of all the declarations that are in the scope of the line you're on.
- Operation: Refactor ►Debugging ►Find All Declarations in Scope You will see a list of all the variables that are in scope. You can type in a variable to see if it is in scope.
- Example: Example of the Find All Declarations in Scope refactoring

Select Enclosing Scope

- Description: Highlights the entire range of source text corresponding to a token?s enclosing ScopingNode.
- Operation: Click on a token in the editor. Refactor ►Debugging ►Select Enclosing Scope

Display Error/Warning Log

- Description: Creates file with a log of errors and warnings.
- Operation: Refactor ►Debugging ►Display Error/Warning Log

Resolve Interface Binding

- Description: Resolves all interface binding.
- Operation: Refactor ►Debugging ►Resolve Interface Binding

Find Matching Interface Declarations

- Description: Finds matching interface declaration for an identifier in a subprogram declaration.
- Operation: Refactor ►Debugging ►Find Matching Interface Declarations

Display Binder Statistics

- Description: Creates file with a list of average and maximum times for ReferenceCollider, ModuleLoader, etc.
- Operation: Refactor ►Debugging ►Display Binder Statistics
- Example: Example of the Display Binder Statistics refactoring

Reset Binder Statistics

- Description: Resets the Binder Statistics. If you Display Binder Statistics, there will be no data under Average Times.
- Operation: Refactor ►Debugging ►Reset Binder Statistic Display Database Statistics
- Description: Creates file with stats on how many times methods such as getIncomingEdgesTo, getAnnotation, and isOutOfDate have been called.
- Operation: Refactor ►Debugging ►Display Database Statistics
- Example: Example of the Display Database Statistics refactoring

Reset Database Statistics

- Description: Resets the Database Statistics. If you Display Database Statistics, there will be no data.
- Operation: Refactor ►Debugging ►Reset Database Statistic

Display Database

-
- Description: Creates file with database statistics on dependencies, edges, and modification stamps.
- Operation: Refactor ►Debugging ►Display Database
- Example: Example of the Display Database refactoring

Display Edge Model of Current File

- Description: Creates file that has info on all of the edges.
- Operation: Refactor ►Debugging ►Display Edge Model of Current File
- Example: Example of the Display Edge Model of Current File refactoring

Ensure Database is Up-to-Date

- Description: Makes sure that the database is up-to-date.
- Operation: Refactor ►Debugging ►Ensure Database is Up-to-Date

Clear and Rebuild Database

- Description: Clears and rebuilds the database.
- Operation: Refactor ►Debugging ►Clear and Rebuild Database

Browse VPG Database

- Description: Allows you to view all edges, dependencies, and annotations for all files.
- Operation: Refactor ►Debugging ►Browse VPG Database Select a file and switch between edges, dependencies, and annotations on the right side of the box.
- Example: Example of the Browse VPG Database refactoring

3.10 Not Yet Available

The following refactorings have been submitted but are not yet included in Photran.

Add Use Of Named Entities

- Description: This refactoring adds the statement use M, only: ... to another module, if a similar statement does not already exist. It fails if this will result in a naming conflict, the introduction of circular dependencies between modules, or if a statement use M already exists but renames any named entities.
- Applies To: A list of named entities.
- Operation: Move your cursor to the module you want to "use". Click Refactoring ►Add Use Of Named Entities to Modules. Select the named entities you would like to use, and the module you want to use them in. Click preview to view the changes, and OK to apply the changes.
- Example: Example of the Add Use Stmt refactoring.

Move Module Entities

- Description: This refactoring moves a set of named entities from one module to another, and updates all use statements that reference these modules.
- Applies To: A list of named entities.
- Operation: Move your cursor to the module with the entities you would like to move. Click Refactoring ►Move Module Entities. Select the named entities you would like to move, and the module you want to move them to. Click preview to view the changes, and OK to apply the changes.
- Example: Move Module Entities example.

Change Subroutine to Function

-
- Description: This refactoring will change a subroutine to a function. The refactoring fails if there are no intent out arguments.
- Applies To: A file.
- Operation: Select the subroutine name or subroutine statement that needs to be changed to a function Click Refactoring ►Change Subroutine To Function. Click preview to view the changes without applying, or OK to apply the changes.

- Example: Example of Change Subroutine to Function refactoring.

Convert Specification Statement to Declaration Attribute

TODO

Toggle End Name

TODO

Standardize Input Output Format

TODO