Photran 4

Photran is a graphical interface for Fortran. It may be used to edit, compile, run, and debug Fortran programs. It is based on the Eclipse open source software (`http://www.eclipse.org`). Photran itself consists of plugins for eclipse developed at the University of Illinois (`http://www.eclipse.org/photran`). Photran also is open source software. In addition, some enhancements have been provided by The Fortran Company.

4.1 Introduction to Using Photran

You can always run Fortran programs from the command line (Section 2), but if you want to use the Photran graphical interface to edit, compile, run, and debug Fortran programs, follow the instructions in this section. But first a little jargon so you can read additional documentation about Eclipse and Photran.

When using Photran, your code is organized into *workspaces* and *projects*. A project usually will contain the code for one complete program, consisting of a main program and possibly some modules. These files, and others used by the Photran system, usually are stored in one directory whose name is the name of the project. A workspace consists of projects; it uses a directory whose name is the name of the workspace to store the project directories. A workspace might contain only one project.

To use Photran for Fortran programs, you create a project, which is part of some workspace. You then add source code to the project, either by copying existing files into the project or by creating new source files and typing in the code. Then, using Photran, the project can be built, run, and debugged.

4.2 Starting Photran

1. If you created a shortcut to Photran, select it. Otherwise, type `photran`. If this does not work, in Windows, you can use `Explore` to find `eclipse.exe` and select it for execution.

2. If the screen contains only some logos, select the curved arrow labelled Workbench to start running some Fortran programs.

3. You will be asked to select a workspace. It is probably a good idea to select a directory different from the location of the software installed from the CD. Don't select a workspace with a space in the path name. C:\Photran\workspace might be a good choice.

4.3 Creating a New Project

Here are the steps to create a new project. As with most other Photran operations, there are several ways to do it. Here is one.

1. Select the File tab at the upper left corner of the screen, then New, then Project. In the New Project dialog, expand Fortran and then select Fortran Project. Then Next.

2. On the !Fortran Project! screen, enter a name for the project, such as a_simple_project. The default workspace directory (e.g., C:/Photran/workspace) should be a good choice for this example. Select Makefile Project (don't expand it). Then select Next (not Finish).

3. On the Select Configurations screen, Default should be checked (OK). Click on Advanced Settings. On the Properties screen, expand Fortran Build and select Settings.

4. For the Binary Parsers tab, select PE Windows Parser.

5. For the Error Parsers tab, check the following parsers and uncheck the rest:

```
CDT GNU Make Error Parser
CDT GNU C/C++ Error Parser
CDT GNU Assembler Error Parser
CDT GNU Linker Error Parser
Photran Error Parser for G95 Fortran
```

6. Click on Apply and OK.

7. Now select Finish. in the Select Configurations dialog.

8. Select the + symbol to the left of the project name in the `Navigator` view and you will see files that have been put there by Photran. They contain information about the project.

If projects have been created previously, some or all of these settings may already be in place.

4.4 Importing Existing Files

Right click on the project name and select `Import` from the list of options. Expand `General` and select `File System` from the next screen and then `Next`. On the `File System` screen, use the `Browse` button to find the directory Cygwin/`usr/local/fortrantools/lib`. Select `Makefile`. Then `Finish` to copy it to the current project. This file needs to be present in every project unless you create your own Makefile.

Or you can simply copy `Makefile` into the directory containing your project files.

4.5 Create a New Source File

To create a new source file, select `File` in the upper left corner of your screen, then `New`, then `Source File`. There are also icons to do this; determine which ones by putting your curser over the icons to see what they do. Enter a name (use `sine.f95` for example) for the source file and `Finish`.

4.6 Editing a Source File

1. Double click on the file in the `Navigator` view. This will display the contents of that file in the editor view, which occupies the upper central portion of the screen.

2. If you make a change to the file (this will be indicated by an asterisk by the file name just above the edit view), save the file by selecting the save (floppy disk) icon near the upper left corner of your screen.

   To provide a simple example, enter the following program:

```
program sine
   print *, “The sine of 0.5 is”, sin(0.5)
end program sine
```

Note the syntax highlighting of Fortran code by Photran. Comments, character strings, and keywords appear in different colors so that they may be identified readily.

Line numbers do not appear in the edit window, but the line number and character position within the line of the cursor are displayed below the edit window.

Here is another nice feature of Photran. If you want to comment a whole block of statements, it is necessary to put the comment symbol (`!`) at the beginning of each statement. To do this using Photran, select the lines to be commented (or uncommented), right click in any open space in the `Edit View` and select `Comment` (or `Uncomment`).

4.7 Building a Project

To build the program, select the *project name* in the `Navigator View` and select `Project.` Then select `Build Project` from the pulldown menu. If the `Build Project` option cannot be selected, uncheck the `Build Automatically` option and try again. For `a_simple_project`, something like the following should appear in the `Console` view near the bottom of your screen (make sure the Console tab is highlighted).

```
make all
perl /usr/local/fortrantools/lib/mkmf.pl -t \
   /usr/local/fortrantools/lib/mkmf_args \
   -p RUN -m f_Makefile -x
make[1]: Entering directory \
   `/home/walt/FortranTools/workspace/test'
g95 -g -Wall -fbounds-check -ftrace=full \
   -I/usr/local/fortrantools/lib \
   ./sine.f95
g95 trig_plot.o -o RUN \
   -L/usr/local/fortrantools/lib -lfortrantools \
   -lslatec -lmatrix -lg2c
make[1]: Leaving directory \
   `/cygdrive/c/workspace/a_simple_project'
```

If nothing happens, select the project name again in the `Navigator` view, select the `Project` pulldown menu, and select `Clean`. Select the `Clean selected projects` button and `OK`.

The important steps are those that begin with `g95`. The first of these compiles the program `sine.f95` and the second cre-

ates the executable file RUN. You will notice some new files appearing in the project in the Navigator view, including the file RUN.exe.

4.8 Running a Program

A run configuration must be established before any program can be run from Photran. To check if this has been done, select the Run, then Run As tabs above the edit window. If a tab showing Local Fortran Application appears, then simply click it and the program should begin execution.

If (none applicable) appears, go back and select Run and then Open Run Dialog. A window in which a run configuration can be established should appear. C/C++ Local Application should appear in the window to the left (it should say Fortran, but it doesn't). Select the leftmost icon; holding your cursor over the icon will reveal a baloon that says New lanuch configuration. Click on the icon. New configuration appears in the left pane and a new dialog appears.

Select the Main tab. Enter a configuration name; the name of the project might be a good choice. Enter RUN.exe as the C/C++ Application.

Next, select the Environment tab and then New. Enter the name LD_LIBRARY_PATH and the value C:\Cygwin\bin (or use the appropriate disk drive letter). Click OK.

Next select the Debugger tab and select Debugger: GDB Debugger. Uncheck the box Stop on startup at.

Then select Apply and either Run or Close.

Once the run configuration has been set up, instead of selecting Run tab and Run As, click on the Run button (the little green arrow). After the program has been run successfully, selecting the Run button may cause it to be recompiled if the source code has been changed.

The Console view is used for read * and print *; make sure that view is selected when typing.

4.9 Make Files

When a Fortran program is complicated, it may be necessary to write your own Makefile. This can be done by simply editing the file named Makefile in your project. The Makefile that is provided executes a Perl script (mkmf.pl) which builds another

make file (f_Makefile) based on the organization of modules and use statements in the project. Then that file is used to build the executable program. This is the Makefile provided.

```
FT_LIB = /usr/local/fortrantools/lib
all:
   perl $(FT_LIB)/mkmf.pl \
      -t $(FT_LIB)/mkmf_args -p RUN \
      -m f_Makefile -x
clean:
   rm -f *.mod *.o RUN* f_Makefile
```

Note that the lines executing perl and rm begin with a tab character, not spaces.

This process can be modified in a few simple ways by editing the file mkmf_args (make makefile arguments) located in /usr/local/fortrantools/lib. The one provided is:

```
FC = g95
FFLAGS = -g -Wall -fbounds-check -ftrace=full \
   -I/usr/local/fortrantools/lib

LD = g95

LDFLAGS = -L/usr/local/fortrantools/lib \
-lfortrantools -lslatec -lmatrix -lg2c
```

The first and third lines indicate that g95 is to be used to compile and load the program. The second line provides options to the compiler. -g is used for debugging, -Wall says to check for as many errors as possible (subscripts out of bounds, for example), and -I tells the compiler where to find some modules provided with Fortran Tools. When the program is ready for production use, this line might be changed to

```
FFLAGS = -O -I/usr/local/fortrantools/lib
```

to turn off error checking and turn on optimization. The -I option can be deleted from FFLAGS and the -L and -l options can be deleted from LDFLAGS if no Fortran Tools modules are being used.

Documentation for mkmf.pl is in the doc directory of the Fortran Tools distribution.

To see how the make file system works, let's go through an example provided by the `test_make` project.

1. Create a new project named `test_make` and import the file `Makefile` in the directory `/usr/local/fortrantools/lib`.

2. Import the files `m1.f95`, `m2.f95`, `m3.f95`, and `p.f95` from the examples directory of the distribution.

3. Look at the source files. There are three modules and a main program. Module `m1` contains declarations of the parameters `pi` and `e`. `m2` contains a subroutine `s` that uses module `m1` and prints the value of `pi`. `m3` uses `m1` and `m2` and contains a subroutine `s3` that calls `s` to print `pi` and also prints `e`. The main program `p` uses `m1` so it can print the values of `e` and `pi`. It contains a subroutine `ss` that uses `m3` and calls its module procedure `s3`.

4. Build the program. Note the compile commands that are executed and the order in which they are executed.

5. Experiment by changing one of the source files and then rebuilding the program. For example, change `m2` so that the subroutine prints `2*pi`. Don't forget to save the changed file and select the project before selecting `Build Project`. Note the compile commands when the project is rebuilt. Change the subroutine in `m2` and rebuild. Then change the value of `e` in `m1` and rebuild. In all cases, only the files that need to be recompiled are recompiled. This is not important for such a small program, but is for a big complicated one.

4.10 Deleting a Project

To delete a project, right click on the project name in the `Navigator` view and select `Delete`. The next screen gives you the option of keeping or deleting the contents of the project directory when the project is deleted.

4.11 Debugging

Programs can be debugged using the same Photran interface that is used to edit, build, and run the programs. The debugger has a lot of features, some of which take some effort to learn, but if all you use it for is a replacement for debugging by in-

serting `print` statements, learning just the simplest features to do that will be well worth the effort.

Let's learn about some of the features with an example.

1. Create a new project named `buggy` in your workspace.
2. Import the file `buggy.f95` in the `examples` directory of the Fortran Tools distribution. Take a look at it if you like.
3. Build the project.
4. If you have not done so already, create a run configuration (4.8) and run the program. There appears to be a problem; if you can figure it out, great, but if not we need to do some debugging.
5. Make sure that you have unchecked the box labelled `Stop at main(0) on startup`. when creating the run configuration (`Run`, `Open Run Dialog`, `Debugger` tab.
6. Set a breakpoint: with the source file `buggy.f95` in the edit window, place the cursor in the left margin of the edit window to the left of the statement

   ```
   j = 1
   ```

   Double click or right click and select `Toggle Breakpoint`. Notice that a small blue circle appears in the margin to indicate the presence of the breakpoint. If you don't set a breakpoint, execution of the program may hang and you will have to terminate the program `gdb` by other means (see 4.12).
7. Select the project name; select the `RUN` tab near the top of the screen; then `Debug As`; then `Local C/C++ Application`. Or click on the debug icon to the left of the Run icon. You will be asked if you want to change the perspective; answer `Yes`. The arrangement of views changes significantly.
8. In the upper right corner of the screen, there is a little window that says `Debug`. This used to say `Fortran`. With the little icon to the left of this window you can change the perspective (the arrangement of the views) to `Debug` or `Fortran`. Try it.

9. With the perspective set at `Debug`, the program appears in a view near the center of the screen. The program is suspended at the breakpoint as indicated by the little arrow in the left margin pointing to the `program` statement.

10. To determine the problem, we want to execute a few statements and then see how things look. One way to do this is to use the icons above the `Debug` view. Move your cursor over them to see what they do. `Restart` begins execution of the program from the beginning. `Resume` continues execution from the current place in the program until it hits a breakpoint. `Terminate` (the red square) stops the program. `Step Into` executes one Fortran statement; if it involves a function evaluation or a subroutine call, it stops at the beginning of the procedure invoked. `Step Over` executes one statement, but does not stop inside a procedure that is invoked. Another similar operation is `Run to Line`; there is no icon for this, but can be performed by right clicking in open space in the `Debug` view and selecting it; it causes the program to run to the point where a line is selected with the cursor.

11. Use `Step Over` or `Step Into` to run to the first `if` statement. check the value of the variables `i` and `j` by examining the `Variables` view in the upper right portion of the screen.

12. Perform `Step Over` several times to watch the loop get executed three or four times. Look at the `Variables` view and notice that each time `j` changes, it turns yellow. In fact, since the loop exits only when `i > n`, and `i` never changes during the loop, that explains the problem. Fix it by changing the test to use `j` instead of `i`. Probably the easiest way to do this is to terminate the program by selecting the red square, edit the source file, and rebuild the project.

13. We have fixed the bug, but let's try a few more things with the debugger to see how they work. After rebuilding the project, set a breakpoint at the first `print` statement and start the debugger again

14. When the program stops at the breakpoint, look in the `Variables` view. `j` is 11, as it should be. To see the values of `my_array`, select the + symbol to its left. Note that the el-

ements of the array are numbered from 0. This is because the debugger is derived from a C debugger. To see the value of the character string `c`, do the same thing. It is treated as an array by the debugger because in C, a character string is treated as an array of characters.

15. Use `Step Into` until you get to the `call` statement. Be sure to use `Step Into` again (maybe a couple of times) to enter the subroutine `SubA`.

16. Place a breakpoint on the line

```
zed(i) = FuncB(y)
```

and select Resume to run to the breakpoint. Note that the variables local to the subroutine have been added to the *bottom* of the list. Also, there are variables `i` with two different values; one is the `i` declared local to the subroutine and the other is the `i` in the main program.

17. `Step Into FuncB`. Notice that the variables local to the function (e.g., `xx` and `B_result`) have been added to the `Variables` view.

18. Suppose we think all is OK in `FuncB`. `Step Return` to complete execution of `FuncB` and go back to `SubA`.

19. Now suppose we suspect that something goes wrong during the last iteration or two of the `do` loop in `SubA`. It would be tedious step through the loop more than 300 times. Instead we can set a conditional breakpoint. First, set a breakpoint at the line

```
y = i
```

Then right click on the blue circle and select `Breakpoint Properties`. Alternatively, select the `Breakpoints` tab near the upper right corner of the screen, right click on the break point just created and select `Properties`. Select `Common` and in the `Condition` field, type

```
i > 357
```

Another option would be to type something like 355 in the `Ignore` field so that the breakpoint would be passed 355

times before the program is stopped. Select OK. Note the ? over the blue circle representing the breakpoint.

Resume to run to the breakpoint just set. Look at the Variables window and check to be sure that the loop was executed until the breakpoint condition was met. If this doesn't work, remove the breakpoint at the print statement and start the debugging process over again

20. Select the array zed to look at some of its values. Note that you can select portions of the array, which is very convenient if the array is large.

21. Select the red square to terminate the program. Return to the Fortran perspective.

4.12 Terminating a Program

Usually, an executing program can be stopped by clicking on the red square.

Sometimes, an instance of RUN or gdb may be left running when you thought everything was terminated. This happens especially during debugging. For example, the compiler may not be able to create a new version of RUN if the program is running. If this appears to be a problem, terminate all instances of the programs RUN and gdb. Ctrl-Alt-Del to get the task manager.

4.13 Other Sources of Information

1. With Photran running, select the Help tab and Help Contents. The leads you to the *Workbench Users Guide*.

2. The *Workbench Users Guide* is also available in the doc directory of the Fortran Tools distribution and at

   http://www.eclipse.org/documentation/main.html

Unfortunately, at this time, there is no additional documentation specific to Photran.

3. In the doc directory of the distribution, the file cdt.pdf contains the *C/C++ Development Toolkit User Guide*. Because some of the Photran software was developed from this

toolkit, there is a lot of information that is applicable to Photran, particularly the debugger information.