Photran 3.1 Developer's Guide

J. Overbey

Contents

Chapter 1

Introduction

Last Updated 4/4/07

Photran is a IDE for Fortran 90/95 and Fortran 77 that is built on top of Eclipse. It is structured as an Eclipse feature, in other words, as a set of plug-ins that are designed to be used together. Starting with version 3.0, it is an extension of CDT, the Eclipse IDE for C/C++. Previous versions of Photran were created by hacking a copy of the CDT to support Fortran instead of C/C++, but now we have developed a mechanism for adding new languages into the CDT, allowing the Fortran support code to be in its own set of plug-ins.

Our purpose in writing Photran was to create a refactoring tool for Fortran. Thus, Photran has a complete parser and program representation. Photran adds a Fortran editor and several preference pages to the CDT user interface, as well as a Fortran Managed Make project type.

This document explains the design of Photran so that you could fix a bug or add a refactoring. You should know how to use Photran and how the CDT works. You need to understand Eclipse and Eclipse plug-ins before you read this document. We recommend *The Java Developer's Guide to Eclipse* for Eclipse newcomers.

1.1 CDT Terminology

The following are CDT terms that will be used extensively when discussing Photran.

- **Standard Make projects** are ordinary Eclipse projects, except that the CDT (and Photran) recognize them as being "their" type of project (as opposed to, say, projects for JDT, EMF, or another Eclipse-based tool). The user must supply their own Makefile, typically with targets "clean" and "all." CDT/Photran cleans and builds the project by running `make`.
- **Managed Make projects** are similar to standard make projects, except that CDT/Photran automatically generates a Makefile and edits the Makefile automatically when source files are added to or removed from the project. The **Managed Build System** is the part of CDT and Photran that handles all of this.
- **Binary parsers** are able to detect whether a file is a legal executable for a platform (and extract other information from it). The CDT provides binary parsers for Windows (PE), Linux (ELF), Mac

OS X (Mach), and others. Photran does not provide any additional binary parsers.

- **Error parsers** are provided for many compilers. CDT provides a gcc error parser, for example. Photran provides error parsers for Lahey Fortran, F, g95, and others. Essentially, error parsers scan the output of `make` for error messages for their associated compiler. When they see an error message they can recognize, they extract the filename, line number, and error message, and use it to populate the Problems view.
- CDT keeps a **model** of all of the files in a project. The model is essentially a tree of **elements**, which all derive from a (CDT Core) class `ICElement`. It is described in the next section.

1.2 The Model

The Fortran Projects view in Photran is essentially a visualization of the CDT's *model*, a tree data structure describing the contents of all Fortran Projects in the workspace as well as the high-level contents (functions, aggregate types, etc.) of source files.

Alain Magloire (CDT) described the model, A.K.A. the `ICElement` hierarchy, in the thread "Patch to create ICoreModel interface" on the cdt-dev mailing list:

```
So I'll explain a little about the ICElement and what we get
out of it for C/C++.

The ICElement hierarchy can be separated in two:
(1) - how the Model views the world/resources (all classes above ITranslationUnit)
(2) - how the Model views the world/language (all classes below ITranslationUnit).

How we(C/C++) view the resources:
- ICModel  --> [root of the model]
    - ICProject --> [IProject with special attributes/natures]
        - ISourceRoot --> [Folder with a special attribute]
            - ITranslationUnit --> [IFile with special attributes, for example extensions *.c]
            - IBinary --> [IFile with special attributes, elf signature, coff etc...]
            - IArchive --> [IFile with special attributes, "<ar>" signature]
            - ICContainer -> [folder]

There are also some special helper classes
    - ILibraryReference [external files use in linking ex:libsocket.so, libm.a, ...]
    - IIncludeReference [external paths use in preprocessing i.e. /usr/include, ...]
    - IBinaryContainer [virtual containers regrouping all the binaries find in the project]

This model of the resources gives advantages:
- navigation of the binaries,
- navigation of the include files not part of the workspace (stdio.h, socket.h, etc ...)
- adding breakpoints
- search
- contribution on the objects
etc.....

[...]
```

```
(2) How we view the language.

Lets be clear this is only a simple/partial/incomplete view of the language.
For example, we do not drill down in blocks, there are no statements(if/else conditions) etc ....
For a complete interface/view of the language, clients should use the __AST__ interface.
```

From another one of Alain's posts in that thread:

```
Lets make sure we are on the same length about the ICElement hierarchy.
It was created for a few reasons:

- To provide a simpler layer to the AST.  The AST interface is too complex
  to handle in most UI tasks.
- To provide objects for UI contributions/actions.
- The glue for the Working copies in the Editor(CEditor), IWorkingCopy class
- The interface for changed events.
- ...

Basically it was created for the UI needs: Outliner, Object action contributions,
C/C++ Project view and more.

The CoreModel uses information taken from:
- the Binary Parser(Elf, Coff, ..)
- the source Parser(AST parser)
- the IPathEntry classes
- the workspace resource tree
- The ResolverModel (*.c, *.cc extensions), ...

to build the hierarchy.
```

1.3 The CDT Debugger and `gdb`

- The so-called CDT debugger is actually just a graphical interface to `gdb`, or more specifically to `gdb/mi`. If something doesn't work, try it in `gdb` directly, or using another `gdb`-based tool such as DDD.
- The debugger UI "contributes" breakpoint markers and actions to the editor. The "set breakpoint" action, and the breakpoint markers that appear in the ruler of the CDT (and Photran) editors are handled entirely by the debug UI: You will *not* find code for them in the Photran UI plug-in.
- `gdb` reads debug symbols from the executable it is debugging. That is how it knows what line it's on, what file to open, etc. Photran has *nothing* to do with this: These symbols are written entirely by the compiler. Moreover, the compiler determines what shows up in the Variables view. If the debugger views seem to be a mess, it is the compiler's fault, not Photran's.

Chapter 2

Plug-in Decomposition

Last Updated 4/4/07

2.1 The Photran Feature

The following projects comprise Photran as it is distributed to users. All sources are Java 4.

- org.eclipse.photran-feature

  This is the Eclipse feature for Photran, used to build the Zip file distributed to users. (A feature is a grouping of related plug-ins.)

- org.eclipse.photran.cdtinterface

  This contains all of the components (core and user interface) related to integration with the CDT. It includes

  - ILanguage for Fortran (i.e., the means of adding Fortran to the list of languages recognized by the CDT)
  - Fortran model builder, model elements, and icons for the Outline and Projects views
  - Fortran perspective, Fortran Projects view, and new project wizards

- org.eclipse.photran.core

  This is the Photran Core plug-in. It contains most of the Fortran-specific "behind the scenes" functionality:

  - Utility classes
  - Error parsers for Fortran compilers
  - Fortran 95 parser
  - Workspace preferences

- org.eclipse.photran.core.errorparsers.xlf

  Error parser for the XLF compiler. Managed by Craig Rasmussen at LANL.

- org.eclipse.photran.managedbuilder.core, org.eclipse.photran.managedbuilder.gnu.ui, org.eclipse.photran.managedbuild
  org.eclipse.photran.managedbuilder.ui

  Support for Managed Build projects using the GNU toolchain. Managed by Craig Rasmussen at LANL.

- org.eclipse.photran.core.intel, org.eclipse.photran.intel-feature, org.eclipse.photran.managedbuilder.intel.ui

  Support for Managed Build projects using Intel toolchains. Maintained by Bill Hilliard at Intel.

- org.eclipse.photran.ui

  This contains the Fortran-specific components of the user interface:

  – Editor
  – Preference pages

2.2 The Refactoring Feature

The following projects support analysis and refactoring of Fortran sources. They are written in Java 5. The functionality in org.eclipse.photran.core.analysis should really be in org.eclipse.photran.core, and likewise with the analysis UI, although they are separate since the main Photran plug-in needs to be compilable with Java 4. (Someone originally wanted to compile a native version using gcj, as was done with CDT and JDT at Red Hat, and gcj does not currently support Java 5.)

- org.eclipse.photran.refactoring-feature

  The analysis and refactoring feature, distributed separately from the base Photran distribution.

- org.eclipse.photran.core.analysis

  Support for type checking and analysis of scope, bindings, etc., mainly to support refactoring. This also includes the Fortran Indexer, which caches to disk information about the externally-visible modules, functions, etc. in each Fortran file in the user's workspace.

- org.eclipse.photran.ui.analysis

  UI contributions (other than refactorings) that depend on the org.eclipse.photran.core.analysis plug-in. Currently, this is the Open Declaration action and a project property page where the user can customize the search path for Fortran modules.

  Some debug actions (e.g., displaying the index, dumping the symbol table, describing the type of a variable) are also included at the time of writing.

- org.eclipse.photran.refactoring.core, org.eclipse.photran.refactoring.ui, org.eclipse.photran.refactoring-feature

  Support for refactoring Fortran. In progress.

2.3 Other Plug-ins

The following projects are in CVS but are not distributed to users:

- org.eclipse.photran-dev-docs

  Developer documentation, including this document (`dev-guide/*`), CVS instructions (`dev-guide/cvs-instructions.p`
  the materials from our presentation at EclipseCon 2006 on adding a new language to the CDT, and a spreadsheet mapping features in the Fortran language to JUnit tests (`language-coverage/*`).

- org.eclipse.photran-samples

  A Photran project containing an assortment of Fortran code.

- org.eclipse.photran.cdtinterface.tests

- org.eclipse.photran.core.tests

- org.eclipse.photran.refactoring.tests

  JUnit tests for the like-named plug-ins.

  org.eclipse.photran.refactoring.tests must be run as a JUnit Plug-in Test. The others are run as ordinary JUnit tests.

Chapter 3

Parsing, Analysis, and Refactoring

Last Updated 4/4/07

3.1 Overview

TODO

Appendix A

Getting the Photran 3.1 Sources from CVS

Last Updated 4/4/07

Part I. Check out the CDT 3.1.2 sources from CVS

If you already have CDT 3.2 installed and do not need to edit the CDT source code, Part I can be skipped.

1. In Eclipse, switch to the CVS Repository Exploring perspective.
2. Right-click the CVS Repositories view; choose New, Repository Location
3. Enter the following information, then click Finish:

Host name:	dev.eclipse.org
Repository path:	/cvsroot/tools
Connection type:	pserver
Username:	anonymous
Password:	(no password)

4. Expand :pserver:anonymous@dev.eclipse.org:/cvsroot/tools, and then expand HEAD (in the CVS Repositories view)
5. Expand org.eclipse.cdt-build
6. Under org.eclipse.cdt-build, right click and check out all of the org.eclipse.cdt.* packages (it is OK to skip the ones ending in "tests")
7. Do the same with org.eclipse.cdt-core, org.eclipse.cdt-debug, org.eclipse.cdt-doc, and org.eclipse.cdt-launch
8. You now have the CDT source code. Make sure it compiles successfully (lots of warnings, but no errors).

Part II. Check out the Photran sources from CVS

9. In Eclipse, switch to the CVS Repository Exploring perspective.

10. Right-click the CVS Repositories view; choose New, Repository Location

11. Enter the following information, then click Finish:
    If you are a Photran committer:

Host name:	dev.eclipse.org
Repository path:	/cvsroot/technology
Connection type:	extssh
Username/passwd:	(your eclipse.org committer username and password)

    Otherwise:

Host name:	dev.eclipse.org
Repository path:	/cvsroot/technology
Connection type:	pserver
Username:	anonymous
Password:	(no password)

12. Expand the node for dev.eclipse.org:/home/technology, then expand HEAD (in the CVS Repositories view), then expand org.eclipse.photran

13. Right-click and check out all of the projects under org.eclipse.photran

    The sources should all compile (albeit with lots of warnings).

Note. *Some JUnit tests for the parser and refactoring engine require closed-source code that is not available in CVS. A warning will appear in the JUnit runner if this code is not available.*

Appendix B

Creating an Error Parser

Last Updated 4/4/07

Error parsers scan the output of `make` for error messages for a particular compiler. When they see an error message they can recognize, they extract the filename, line number, and error message, and use it to populate the Problems view.

For an example, see `IntelFortranErrorParser`. (It's a mere 74 lines.)

To create a new error parser, do the following.

- We will assume that your error parser class will be in the `errorparsers` folder in the org.eclipse.photran.core plug-in and added to the `org.eclipse.photran.internal.errorparsers` package.
- Define a class implementing `IErrorParser`
- Implement `public boolean processLine(String line, ErrorParserManager eoParser)` which should always return false because ErrorParserManager appears not to use the result in a rational way
- In org.eclipse.photran.core's `plugin.xml`, find the place where we define all of the Fortran error parsers. Basically, copy an existing one. Your addition will look something like this:

```
<extension
      id="IntelFortranErrorParser"
      name="Photran Error Parser for Some New Fortran Compiler"
      point="org.eclipse.cdt.core.ErrorParser">
   <errorparser
         class="org.eclipse.photran.internal.errorparsers.MyErrorParser">
   </errorparser>
</extension>
```

- Your new error parser will appear in the error parser list in the Preferences automatically, and it will be automatically added to new projects. For existing projects, you will need to open the project properties dialog and add the new error parser to the project manually.

Note. Error parsers to not have to be implemented in the Photran Core plug-in. In fact, they do not have to be implemented in Photran at all. If you create a brand new plug-in, you can specify org.eclipse.cdt.core as a dependency, include the above XML snippet in your plug-in's `plugin.xml`, and include your custom error parser class in that plug-in.