Photran Developer's Guide

Part I: General Information

June 29, 2012 Revision

N. Chen
J. Overbey

Contents

Checklist for New Developers

Revision: $Id: cha-checklist.ltx-inc,v 1.2 2011/06/22 19:29:48 joverbey Exp

- ◯ **Install Photran, read its documentation, and become a proficient user.** Generally speaking, you cannot fix or enhance a program you don't know how to use.
- ◯ **Check out Photran's source code from Git and run the test suite.** See Appendices A and B.
- ◯ **Join the ptp-dev mailing list.** Photran is a component of the Parallel Tools Platform (PTP); this is where discussions about PTP development (including Photran) take place. Questions from developers (like you), release planning, discussions about API changes, etc. all take place here. If you have questions that are not answered by the Photran Developer's Guide, you may ask them on the ptp-dev list.
- ◯ **Join the photran users' mailing list.** It is good to keep track of what Photran's users are discussing, and as a Photran developer you will probably be able to answer many of their questions as well.
- ◯ **Create a Bugzilla account.** You will need to create an account at bugs.eclipse.org if you don't already have one. When you eventually want to contribute code to Photran, you will need to do it by attaching a patch to a Bugzilla bug.
- ◯ **Get a book or two on Eclipse plug-in development.** Eventually, you'll want to buy (or check out from the library) a book on Eclipse plug-in development. The Java Developer's Guide to Eclipse by D'Anjou et al. is a good choice. Eclipse: Building Commercial-Quality Plug-ins is also recommended.
- ◯ **Learn the basics of Eclipse plug-in development.** If you get D'Anjou's book, read Chapters 7-9 and 21. After you figure out the basics of Eclipse development (like how plugin.xml works), it's best to just start fixing bugs or adding features, returning to the book as a reference when you have something more specific you need to do.
- ◯ **Read the Photran Developer's Guide.** Chapters 1, 2, and 3 apply to everyone. The other chapters are more specialized; for example, the chapters on Parsing and Refactoring only apply to people developing refactorings, and the Photran Editors chapter applies mostly to people adding features to the Fortran editor.

Chapter 1

Introduction

Revision: $Id: cha-intro.ltx-inc,v 1.1 2010/05/21 20:12:20 joverbey Exp - based on 2008/08/08 nchen

Photran is an IDE for Fortran 77–2008 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 C/C++ Development Tools (CDT), the Eclipse IDE for C/C++. The first version of Photran was created by hacking a copy of CDT to support Fortran instead of C/C++, but now we have developed a mechanism for adding new languages into 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 also adds a Fortran editor and several preference pages to the CDT user interface, as well as Fortran-specific project wizards and support for several Fortran compilers.

Photran is part of the Parallel Tools Platform (PTP) project at the Eclipse Foundation, which provides the Web site, Git repository, and Bugzilla repository. The Web site is http://www.eclipse.org/photran, and bugs can be submitted at https://bugs.eclipse.org under Tools > PTP by selecting one of the "Photran" components.

How To Read This Guide

The *Photran Developer's Guide* is intended to complement the source code in the repository; it is not a substitute for reading the actual source code. Rather, it contains high-level information that *can't* easily be derived simply by reading the source code.

The *Photran Developer's Guide* has been divided into two documents. This document is Part I: It contains general information that every developer should know. Part II gives more detail about specific components of Photran; it should be used as a reference, depending on what specific parts of Photran you want to work on.

Getting Started

If you have not already, work through the items on the "Checklist for New Developers" at the beginning of this manual. In particular, *please* join the ptp-dev mailing list, and use it to ask for help! Although Photran is better documented than many projects, you will probably encounter problems that you cannot resolve simply by reading the documentation or reverse engineering the code.

When your contribution is finished, follow the procedure in Appendix F to contribute your code to the "official" version of Photran.

Chapter 2

Plug-in Decomposition

Revision: $Id: cha-plugins.ltx-inc,v 1.1 2010/05/21 20:12:20 joverbey Exp - based on 2008/08/08 nchen

2.1 Introduction

This chapter presents a high-level overview of the different projects and plug-ins in Photran. It serves as a guide for developers reverse-engineering Photran to *guess-and-locate* where certain components are. It also serves as a guide for contributers on *where* to put their contributions.

2.2 The Rephraser Engine

There are two major components in Photran's Git repository: Photran itself, and another project called the Rephraser Engine. For the most part, you will probably be able to ignore the Rephraser Engine, but it is helpful to know what it is. (The Rephraser Engine will contribute a Rephraser Engine Plug-in Developer Guide to the runtime workbench; this provides more information about these plug-ins. Its sources are in the org.eclipse.rephraserengine.doc.isv project.)

While developing Photran, one of our research objectives has been to create a common infrastructure that can be reused in refactoring tools for different languages. The Rephraser Engine is one part of this infrastructure: It contains base classes for refactorings, the virtual program graph (VPG), etc. *None of the classes in the Rephraser Engine know about Fortran, Photran, or CDT.* They are completely language-independent. However, many of the classes in Photran's refactoring engine inherit from base classes in the Rephraser Engine: They are "specialized" to work with Fortran.

In addition to Photran, Ralph Johnson's research group at UIUC is developing prototype refactoring tools for PHP, Lua, and BC; all of these use the Rephraser Engine. Therefore, if you ever need to change a class in the Rephraser Engine, you should be sure that the change applies to all of these tools, not just Photran.

2.3 Photran's Architecture: Overview and Dependencies

Figure 2.1 illustrates the plug-in decomposition of Photran and the Rephraser Engine, as well as their relationship with CDT and the Eclipse Platform. *Pay careful attention to the dependencies in this diagram.* If you are implementing a feature but need to introduce a dependency that is not present, you are probably implementing the feature in the wrong

component. It may be necessary to add an extension point to add your feature in the "correct" way. This should be discussed on the ptp-dev mailing list.

The following sections are grouped by feature. A feature is a collection of related Eclipse plug-ins that the user can install as a whole.

2.4 Base Photran Feature: org.eclipse.photran-feature

The following projects comprise the "base" of Photran.

- **org.eclipse.photran.cdtinterface**

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

  - The FortranLanguage class, which adds Fortran to the list of languages recognized by CDT
  - Fortran model elements and icons for the Outline and Fortran Projects views
  - An extension point for contributing Fortran model builders
  - The Fortran perspective, Fortran Projects view, and other CDT-based parts of the user interface
  - New Project wizards and Fortran project templates

  More information about CDT integration is provided in Part II of the *Photran Developer's Guide.*

- **org.eclipse.photran.core**

  This is the Photran Core plug-in. It contains much of the "behind the scenes" functionality which allows Eclipse to support Fortran projects (although it does *not* contain the Fortran parser/analysis infrastructure).

  - Workspace preferences for Fortran projects
  - Error parsers for Fortran compilers
  - Utility classes

- **org.eclipse.photran.managedbuilder.gnu.ui**,
  org.eclipse.photran.managedbuilder.ui

  Support for Managed Build projects using GNU Fortran (gfortran). Originally created by Craig Rasmussen at Los Alamos National Lab.

- **org.eclipse.photran.ui**

  This contains the Fortran-specific components of the user interface that are not derived from CDT and do not depend on the Fortran parser. These are:

  - Fortran Editors
  - Preference pages

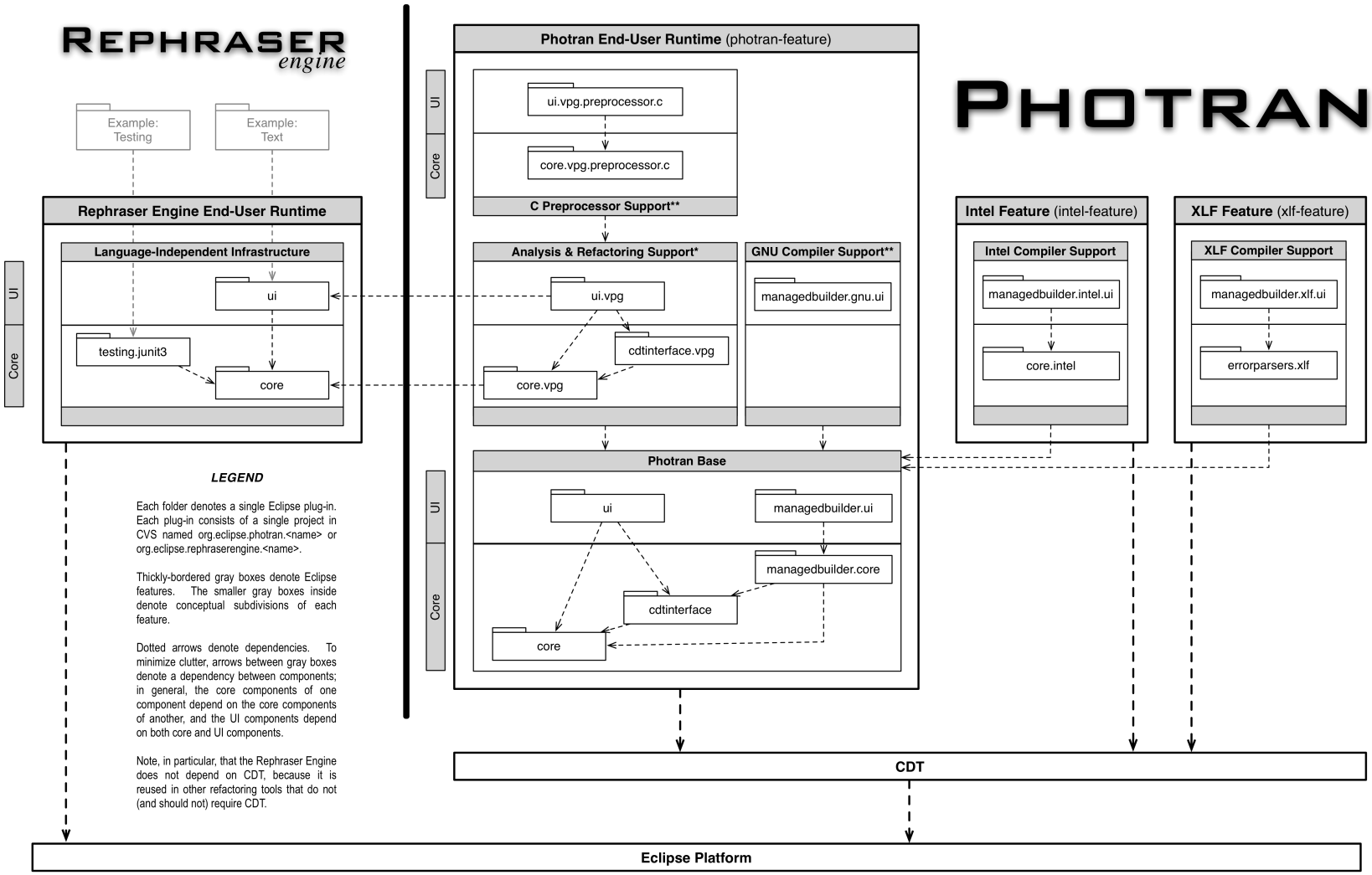

Figure 2.1: Plug-in decomposition of Photran and the Rephraser Engine

2.5 Virtual Program Graph (VPG) feature: org.eclipse.photran.vpg-feature

The following projects support parsing, analysis, and refactoring of Fortran sources. They are written in Java 5. The Virtual Program Graph is described in more detail in Part II of the *Photran Developer's Guide.*

- **org.eclipse.photran.core.vpg**

  This contains the parsing, analysis, and refactoring infrastructure.

  - Fortran parser and abstract syntax tree (AST)
  - Fortran preprocessor (to handle INCLUDE lines)
  - Parser-based model builder
  - Photran's Virtual Program Graph (VPG)
  - Utility classes (e.g., `SemanticError`, `LineCol`)
  - Project property pages
  - Name binding analysis (equivalent to symbol tables)
  - Refactoring/program transformation engine
  - Refactorings

- **org.eclipse.photran.core.vpg.tests**,
  org.eclipse.photran.core.vpg.tests.failing

  JUnit Plug-in tests for the VPG core plug-in.

  All tests in org.eclipse.photran.core.vpg.tests should pass. Test suites and test cases are placed in the "failing" plug-in project until all of their tests pass.

  These plug-ins *must* be run as a "JUnit Plug-in Test" (**not** a "JUnit Test"). In the launch configuration, the environment variable TESTING must be set to some non-empty value. (This ensures that the VPG will not try to run in the background and interfere with testing.)

- **org.eclipse.photran.ui.vpg**

  UI contributions that depend on the `org.eclipse.photran.core.vpg` plug-in. Currently, this includes the Open Declaration action, a project property page where the user can customize the search path for Fortran modules and include files, Fortran Search support, and all of the actions in the Refactor menu.

- **org.eclipse.photran.cdtinterface.vpg**

  This contributes the Fortran model builder which constructs the model seen in the Outline view and Fortran Projects view. It uses the Fortran parser contained in the org.eclipse.photran.core.vpg plug-in.

2.6 XL Fortran Compiler Feature: org.eclipse.photran.xlf-feature

The following are plug-ins to support the XL Fortran compiler.

- **org.eclipse.photran.core.errorparsers.xlf**,
  org.eclipse.photran.managedbuilder.xlf.ui

  Support for Managed Build projects using XL toolchains. Originally created by Craig Rasmussen at LANL.

2.7 Intel Fortran Compiler Feature: org.eclipse.photran.intel-feature

The following are plug-ins to support the Intel Fortran Compiler.

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

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

2.8 Non-plug-in projects

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

- **org.eclipse.photran-dev-docs**

  Developer documentation, including this document (`dev-guide/*`), Git instructions (`dev-guide/git-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.

Appendix A

Checking Out the Photran Sources from Git

Last modified June 29, 2012

This describes how to check out the source code for the current *development* version of Photran. At any given time, this is the version of Photran that will be released the following June.

The source code you check out using this document is *not* guaranteed to be stable. There are rare points in time when it may not even compile. So, if you have problems following the instructions in this document, please ask for help on the ptp-dev mailing list – https://dev.eclipse.org/mailman/listinfo/ptp-dev.

Before you begin...

1. Photran's source code *will not compile* if you are using the wrong version of Eclipse! You will need to install *Eclipse Classic*, also known as the *Eclipse SDK*, which includes the Eclipse Platform, Java development tools, and Plug-in Development Environment. (Since Photran is a set of Eclipse plug-ins, you need a distribution of Eclipse that includes the Plug-in Development Environment.) What version you need depends on the time of year.
   - Between January and June, you need to be running the most recent milestone build (in preparation for the upcoming release, which occurs in June). To download the latest milestone build, browse to
     http://download.eclipse.org/eclipse/downloads/
     and click on the link for "4.x Stream Stable Build." This will take you to a page where you can download the Eclipse SDK. (For example, in January–June, 2013, you will need the 4.3 Stable Stream Build for the upcoming Eclipse Kepler release train.)
   - Between July and December, you can use the latest release. Download Eclipse Classic from
     http://www.eclipse.org/downloads/
     (For example, in June–December, 2012, you will download Eclipse 4.2 Classic.)
2. You will need to be running a Java 6 or later Java Virtual Machine (JVM).
3. *(Optional)* You may want to install the following Eclipse plug-ins, although none of them are necessary:
   (a) **Subclipse** provides Subversion support – http://subclipse.tigris.org/
   (b) **FindBugs** is a static analysis/bug detection tool for Java – http://findbugs.sourceforge.net/
   (c) **EclEmma** is a tool which helps you determine the code coverage of your JUnit tests – http://www.eclemma.org/
   (d) **Metrics** provides code metrics for Java code – http://metrics.sourceforge.net/

Part I. Install the CDT SDK, EGit, and Photran

First, you will need to install the C/C++ Development Tools Software Development Kit (CDT SDK), EGit, and the most recent stable release of Photran itself. Photran reuses some parts of CDT, and the CDT SDK is required to view the source code for CDT. EGit provides support for the Git distributed version control system in Eclipse; it is required to access the Git repositories at the Eclipse Foundation which contain Photran and CDT's source code. Installing the latest release of Photran ensures that Fortran files will be properly syntax highlighted (among other things), which is helpful when writing tests. It is also required when setting an API baseline in Part II below.

1. Start Eclipse. Inside Eclipse, click on the "Help" menu, and choose "Install New Software..."
2. In the "Work with:" combo box, choose the update site for the latest Eclipse release train. As of January, 2012, this should be labeled "Juno – http://download.eclipse.org/releases/juno"
3. Expand the "Collaboration" category
4. Under "Collaboration," check the box next to "Eclipse EGit"
5. Expand the "Programming Languages" category
6. Under "Programming Languages," check the box next to "C/C++ Development Tools SDK" (Be sure the name ends with "SDK"!)
7. Under "Programming Languages," check the box next to "Fortran Development Tools (Photran)"
8. Click "Next"
9. The wizard will show the complete list of plug-ins to be installed; click "Next"
10. Accept the terms of the license agreement, and click "Finish"
11. Installation may take several minutes. Restart Eclipse when prompted.

Part II. Set an API Baseline

Eclipse plug-ins are required to follow very strict rules when making changes to their application programming interface (API). The Eclipse Plug-in Development Environment's API Tooling component enforces these rules. You must configure it with an API baseline so it can compare the current source code with the latest stable release, in order to detect API changes.

12. In Eclipse, if you are running Windows or Linux, click on the "Window" menu; if you are running Mac OS X, click on the "Eclipse" menu. Choose "Preferences" from the menu.
13. Expand the "Plug-in Development" category, and choose "API Baselines"
14. Click "Add Baseline..."
15. Enter "Default" for the name
16. Click "Reset"
17. Click "Finish"
18. Click "OK"
19. If prompted to "Do a full build now?", click "Yes"

Part III. Clone the Photran Git repository and check out Photran's source code

Important: *If you already have an earlier version of the Photran source code (e.g., if you checked it out from CVS before we moved to Git), you must delete the existing projects from your workspace. The Git import wizard will not overwrite them; it will give an error message and fail.*

20. Switch to the "Git Repository Exploring" perspective. (From the "Window" menu, choose "Open Perspective", and "Other..."; choose "Git Repository Exploring", and click "OK.")
21. From the "File" menu, choose "Import..."
22. Under "Git", choose "Projects from Git", and click "Next"
23. Select "URI", and click "Next"
24. In the URI field, enter one of the following.
    - Most people will enter
      `git://git.eclipse.org/gitroot/ptp/org.eclipse.photran.git`
    - If you are a committer at the Eclipse Foundation, enter
      `ssh://username@git.eclipse.org/gitroot/ptp/org.eclipse.photran.git`
      replacing `username` with your committer username.
25. Click "Next"
26. All branches will be selected by default; click "Next"
27. Make sure the local directory is OK (or change it if you want your local Git repository stored somewhere else); then click "Next"
28. The repository will be downloaded (it may spend several minutes "receiving objects").
29. Ensure that "Import existing projects" is selected and "Working Directory" is selected from the list; then click "Next"
30. All projects are selected by default; click "Finish"
31. Switch back to the Java perspective. The Package Explorer view should now contain several new projects with Photran's source code. There should be no compilation errors (although there will be a few warnings).

Appendix B

Running Photran's Automated Test Suite

Last modified April 28, 2010

1. In the Package Explorer view, select the `org.eclipse.photran.core.vpg.tests` project.
2. Right-click on that project and select Run As > Run Configurations.... A dialog will appear.
3. In that dialog, create a new **JUnit Plug-in Test** launch configuration. Call it "Photran-Tests".
4. For the configuration that you have just created, switch to the "Arguments" tab.
5. Change the "VM arguments" field to `-ea -Xms40m -Xmx512m`
6. Switch to the "Environment" tab.
7. *(Optional)* If you are running Linux or Mac OS X and have gfortran installed, some of Photran's refactoring unit tests can attempt to compile and run the Fortran test programs before and after the refactoring in order to ensure that the refactoring actually preserves behavior (and produces code that compiles). The following steps will enable this behavior. Note, however, that if the path to gfortran is incorrect, or if gfortran cannot be run successfully, it will cause the test suite to fail... so you might not want to do this the very first time you attempt to run the test suite.
   (a) Create a new environment variable called `COMPILER` with the full path to gfortran. This will be something like `/usr/local/bin/gfortran`
   (b) Create a new environment variable called `EXECUTABLE` with a path to some non-existent file in your home directory, e.g., `/Users/joverbey/a.out`. When gfortran is run, it will write the executable to this path.
8. Click the "Run" button to run the tests. It will take at least a minute to run the test suite. When it finishes, you should get a green bar in the JUnit view. If you get a red bar, some of the tests failed; the JUnit view will have details.
9. To run the tests again later, just launch the "Photran-Tests" configuration from the Eclipse Run menu.

   Note. *UIUC personnel: See the appendix "Additional Information for UIUC Personnel" in the Photran Developer's Guide for information on additional unit test cases.*

Appendix C

Creating a Launch Configuration for Photran

Last modified May 19, 2010

1. In the Package Explorer view, select the `org.eclipse.photran.core` project (or any other plug-in project).
2. Right-click on that project and select Debug As > Debug Configurations.... A dialog will appear.
3. In that dialog, create a new **Eclipse Application** launch configuration. Call it "Photran".
4. For the configuration that you have just created, switch to the "Arguments" tab.
5. Change the "VM arguments" field to:
   -ea -XX:PermSize=64m -XX:MaxPermSize=128m -Xms64m -Xmx768m
   (These arguments will enable assertions, increase the amount of PermGen space, and increate the amount of heap space available to Eclipse.)
6. *(Optional)* If you will be developing fixed form refactorings, or if you need fixed form refactoring enabled...
   (a) Switch to the "Environment" tab.
   (b) Create a new environment variable called `ENABLE_FIXED_FORM_REFACTORING` with a value of 1.
7. Click the "Debug" button. A new instance of Eclipse will open with the CDT and Photran plug-ins compiled from the code in your workspace.
8. To run it again later, just launch the "Photran" configuration from the Eclipse Run menu. Debug > Debug History > Photran will launch it in the debugger again (this will allow you to set breakpoints, watch expressions, etc.), while Run > Run History > Photran will launch it in a normal JVM (with debugging disabled).

Appendix D

Additional Information for UIUC Personnel

Revision: $Id: app-uiuc.ltx-inc,v 1.2 2010/04/28 18:12:52 joverbey Exp

The following information is only of interest to developers in the Department of Computer Science at the University of Illinois at Urbana-Champaign.

D.1 Photran Subversion Repository at UIUC

https://subversion.cs.uiuc.edu/svn/Photran/

A private Subversion repository for internal testing and research work is provided by TSG at the above URL. *You will only be able to access this repository if you have specifically been granted access.* The repository requires authentication using your NetID and Kerberos password. To access a Subversion repository from Eclipse, you must install an additional Eclipse plug-in (either Subclipse or Subversive). Currently, this repository contains the following.

- **Large Fortran test codes (photran-projects).** Checking these out into your workspace will enable about 6,000 additional test cases in the JUnit test suite. *These applications are PRIVATE. They may NOT be distributed and are intended only for testing Photran.*
- **Sample Obfuscation refactoring (sample-refactoring).** The sample refactoring described in Appendix **??**.
- **Refactoring tools for PHP, Lua, and BC.** Prototype Eclipse-based refactoring tools based on the Rephraser Engine.

D.2 Photran Web Site at UIUC

TSG provides a Web hosting infrastructure based on the popular CPanel system and has set up a web hosting account for Photran. This hosting environment supports PHP, MySQL databases, CGI scripts, and a number of other popular web technologies. Documentation for these features is provided in the control panel. You can access your web content via the control panel, SSH, or WebDAV. Here are the account details:

```
       web URL: http://photran.cs.illinois.edu/
control panel: https://cpanel.cs.illinois.edu:2083/
   SSH access: cpanel.cs.illinois.edu
     web root: ~/public_html
```

Appendix E

Checking Out the CDT Sources

Last modified June 28, 2012

This describes how to check out the source code for the current *development* version of CDT. At any given time, this is the version of CDT that will be released the following June.

In general, you should *not* have to check out CDT's source code from Git, unless you are planning to modify CDT. Photran developers generally do not need to modify/patch CDT; the instructions provided in Appendix A describe how to install the CDT SDK, which is usually sufficient for Photran development.

This information is provided for information only and may be incomplete or out of date. The "official" instructions for checking out CDT from Git are on the Eclipse wiki; see http://wiki.eclipse.org/Getting_started_with_CDT_development and http://wiki.eclipse.org/CDT/git.

1. Switch to the "CVS Repository Exploring" perspective. (From the "Window" menu, choose "Open Perspective", and "Other..."; choose "CVS Repository Exploring", and click "OK.")
2. Right-click anywhere in the white area of the "CVS Repositories" view (on the left). In the popup menu, click New > Repository Location...
3. In the dialog box, enter the following information, then click Finish.

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

4. In the CVS Repositories view
   - Expand ":pserver:anonymous@dev.eclipse.org:/cvsroot/eclipse"
   - Then expand "HEAD"
   - Then click on "org.eclipse.test.performance" to select it
   - Then right-click on "org.eclipse.test.performance" and, in the popup menu, click "Check Out". A dialog box may appear for a few moments while its source code is retrieved.
   - Scroll back to the top of the CVS Repositories view, and collapse ":pserver:anonymous@dev.eclipse.org:/cvsroot/eclipse"
5. Again, right-click anywhere in the white area of the "CVS Repositories" view, and in the popup menu, click New > Repository Location...

6. In the dialog box, enter the following information, then click Finish.
   Host name: dev.eclipse.org
   Repository path: /cvsroot/tools
   Username: anonymous
   Password: (no password)
   Connection type: pserver
7. In the CVS Repositories view
   - Expand ":pserver:anonymous@dev.eclipse.org:/cvsroot/tools"
   - Then expand "Versions"
   - Right-click on "org.eclipse.orbit", and from the popup menu, click "Configure Branches and Versions...".
8. In the "Configure Branches and Versions" dialog,
   - In "Browse files for tags", expand net.sourceforge.lpg.lpgjavaruntime, and then click on ".project"
   - Under "New tags found in the selected files", click the "Deselect All" button. Then, check v1_1 (at the top of the list), so it is the only item selected in the list.
   - Click "Add Checked Tags."
   - Under "Remembered tags for these projects," expand Branches, and make sure "v1_1" appears.
   - Click "OK" to close the dialog.
9. Back in the CVS Repositories view
   - Under ":pserver:anonymous@dev.eclipse.org:/cvsroot/tools", expand "Branches"
   - Then expand "v1_1"
   - Then expand "org.eclipse.orbit v1_1"
   - Then click on "net.sourceforge.lpg.lpgjavaruntime" to select it
   - Then right-click on "net.sourceforge.lpg.lpgjavaruntime" and, in the popup menu, click "Check Out". A dialog box may appear for a few moments while its source code is retrieved.
10. Switch back to the Java perspective. The Package Explorer view should now contain two new projects named org.eclipse.test.performance and net.sourceforge.lpg.lpgjavaruntime. There should be no compilation errors (although there may be a few warnings).
11. Switch to the "Git Repository Exploring" perspective. (From the "Window" menu, choose "Open Perspective", and "Other..."; choose "Git Repository Exploring", and click "OK.")
12. From the "File" menu, choose "Import..."
13. Under "Git", choose "Projects from Git", and click "OK"
14. Click "Clone..."
15. For the URI, enter `git://git.eclipse.org/gitroot/cdt/org.eclipse.cdt.git`
16. Click "Next"
17. All branches will be selected by default; click "Next"
18. Make sure the local directory is OK (or change it if you want your local Git repository stored somewhere else); then click "Next"
19. The repository will be downloaded (it may spend several minutes "receiving objects").

20. Select the (newly-added) org.eclipse.cdt repository in the list, and click “Next”
21. The defaults (“Import Existing Projects”) are OK; click “Next”
22. All projects are selected by default. Click “Finish”.
23. Switch back to the Java perspective. The Package Explorer view should now contain several new projects with CDT’s source code. There should be no compilation errors (although there will be a few warnings).

Appendix F

Contributing a New Feature or Bug Fix

Last modified April 28, 2010

1. Run Photran's automated test suite. All tests must pass.
2. If you are contributing a refactoring, program analysis, or similar complex feature, please include JUnit tests in your contribution. The parser and AST inevitably change over time, and this is how we will determine whether or not a change has broken your contribution.
3. Make sure you did not copy code from anywhere except Photran and CDT. If you copied code from books, mailing lists, Web sites, any other open source projects, other projects at your company, etc., **stop** and ask for further instructions. (If you do not know a Photran committer personally, ask on the ptp-dev mailing list.) It may or may not be possible to contribute your code.
4. Determine who owns the copyright to your code. Generally, if you are an employee and you were paid to write the code, it is the property of your employer. If you are a student, the code may be your personal property, or it may be the property of your university, depending on several factors. Always check with your employer or university to determine who the copyright owner is. The following examples assume that the code was written by John Doe as an employee of the XYZ Corporation.
5. Your code must be contributed under the terms of the Eclipse Public License (EPL). If the copyright is owned by your employer or university, make sure they will permit you to contribute your code under the EPL. (They will probably be asked for verification by the Eclipse Legal team.)
6. Ensure that *every* Java file you created or modified has an accurate copyright header.
   - If you created the file from scratch (i.e., it is not a modification of someone else's code), the copyright header must name the copyright owner and list them as the initial contributor. For example:

```
/*******************************************************************************
 * Copyright (c) 2010 XYZ Corporation and others.
 * All rights reserved. This program and the accompanying materials
 * are made available under the terms of the Eclipse Public License v1.0
 * which accompanies this distribution, and is available at
 * http://www.eclipse.org/legal/epl-v10.html
 *
 * Contributors:
 *    John Doe (XYZ Corporation) - Initial API and implementation
 *******************************************************************************/
```

   - If you modified an existing file from Photran or CDT, it must retain the *original* copyright notice, but you should add yourself as a contributor at the bottom. For example:

```
/*******************************************************************************
 * Copyright (c) 2004, 2008 IBM Corporation and others.
 * All rights reserved. This program and the accompanying materials
 * are made available under the terms of the Eclipse Public License v1.0
 * which accompanies this distribution, and is available at
 * http://www.eclipse.org/legal/epl-v10.html
 *
 * Contributors:
 *    IBM - Initial API and implementation
 *    John Doe (XYZ Corporation) - Photran modifications
 *******************************************************************************/
```

7. Every Java class should have a JavaDoc comment with a brief description of the class and an `@author` tag with your full name. Again, if you modified an existing class, add an `@author` tag with your name. For example:

```
/**
 * Refactoring which creates getter and setter subprograms for a module variable and replaces
 * variable accesses with calls to those subprograms.
 * <p>
 * This refactoring accepts a module variable declaration, makes that declaration PRIVATE, adds
 * getter and setter procedures to the module, and then replaces accesses to the variable outside
 * the module with calls to the getter and setter routines.
 *
 * @author Tim Yuvashev
 * @author Jeff Overbey
 */
public class EncapsulateVariableRefactoring
{ ...
```

8. Create a new bug in Bugzilla on one of the "Photran" components. You can do this by clicking the "Report a Bug" link on the Photran home page. If your code adds a new feature (rather than fixing a bug), set the severity of the bug to "Enhancement."

9. Create a patch with the files that you changed. (Highlight all of the projects you changed in the Package Explorer, right-click, and choose Team > Create Patch.) Attach the patch to the bug in Bugzilla. If you have any binary files (e.g., images), DO NOT include them in the patch; attach them to Bugzilla separately.

10. A Photran committer will review the code and may ask you to make changes. If so, you will need to create a new patch, attach it to the bug, and mark the old patch as "obsolete."

11. When the code review succeeds, the Photran committer will make the following comment on the bug:

    Please confirm that

    (a) you wrote 100% of the code without incorporating content from elsewhere or relying on the intellectual property of others,

    (b) you have the right to contribute the code to Eclipse, and

    (c) you have included the EPL license header in all source files?

    You can reply with something as simple as the following.

    I confirm that

    (a) I wrote 100% of the code without incorporating content from elsewhere or relying on the intellectual property of others,

    (b) I have the right to contribute the code to Eclipse, and

    (c) I have included the EPL license header in all source files.

    However, if you *did* incorporate content from elsewhere – e.g., if your contribution is based on code from CDT or elsewhere – DO NOT copy-and-paste this directly; change the first statement to note this explicitly.

12. Your code will be passed on to the intellectual property (IP) team at the Eclipse Foundation for a legal review. If there are any questions or concerns about the code, a member of the Eclipse IP team will contact you.

13. Once it passes IP review, the committer will commit your code to Photran's Git repository.
14. The committer will add your name to the "Contributors" page of the Photran Web site.

Eclipse Foundation References:
http://wiki.eclipse.org/Development_Resources#Everyone:_IP_Cleanliness
http://wiki.eclipse.org/Development_Conventions_and_Guidelines

Appendix G

Release History

Last modified June 29, 2011

Photran	Date	Platform	CDT	Notes
1.2	Jan 2005	2.1	1.2	First public version; hacked CDT clone
2.1	Feb 2005	3.0	2.1	First version available at eclipse.org
3.0b1	Aug 2005			
3.0b2	Nov 2005			
3.0	Jan 2006	3.1	3.0	CDT extension; required modified CDT
3.1b1	Jul 2006			
3.1b2	Apr 2007	3.1	3.1.1	Extension of stock CDT
4.0b1	Jun 2007	3.2.2	3.1.2	*Rename, intro implicit none*
4.0b2	Oct 2007	3.3.1	4.0.1	
4.0b3	Nov 2007	3.3.1.1	4.0.1	
4.0b4		3.4	5.0	First version release via an update site
4.0b5	Feb 2009	3.4	5.0.1	*Move saved vars to common*
	Mar 2009	3.4	5.0.1	First automated integration build under PTP
	Sep 2009	3.5	6.0.0	
5.0.0	Dec 2009	3.5	6.0.1	First official release at eclipse.org
6.0.0	Jun 2010	3.6	7.0.0	First release as part of the train (Helios)
7.0.0	Jun 2011	3.7	8.0.0	Indigo release train
8.0.0	Jun 2012	4.2	8.1.0	Juno release train
8.1.0	Jun 2013	4.3	9.0.0	Kepler release train (PLANNED)