CViMe: Viewing Conditionally Compiled C/C++ Sources Through Java Tooling

Nieraj Singh  
University of Victoria  
ECS Building  
3800 Finnerty Road, Victoria, BC V8P 5C2, Canada  
nierajsi@uvic.ca

Graeme Johnson  
IBM Ottawa  
2670 Queensview Drive  
Ottawa, Ontario K2B 8K1, Canada  
Tel: 1-613-820-1200  
Graeme_Johnson@ca.ibm.com

Yvonne Coady  
University of Victoria  
ECS Building, Room 610  
3800 Finnerty Road, Victoria, BC  
V8P 5C2, Canada  
Tel: 1-250-472-5715  
ycoady@cs.uvic.ca

Abstract
Today's large software systems and libraries are geared towards a broad range of platforms and environments, often relying on conditional compilation through preprocessor directives to generate specific builds for a given set of configuration flags. In spite of the well-documented benefits of using preprocessor directives for conditional compilation, heavy preprocessor presence can hinder code readability and affect maintenance and debugging. Although various existing preprocessor tools hide unwanted preprocessor conditionals, we present a more portable Eclipse-based solution called CViMe (Conditional-compilation Viewer and Miner) that relies on object-modeling to not only fold non-compiled code, but refactor conditionally compiled blocks into reusable units like macros, significantly reducing the presence of preprocessor directives at the editor level.

Categories and Subject Descriptors  D.2.3 [Software]: Coding Tools and Techniques - object-oriented programming, program editors.

General Terms  Design, Languages

Keywords  C preprocessor, conditional compilation, crosscutting concerns, Eclipse plug-ins, editors, Java object modeling, macros, refactoring, reusability, source views.

1. Introduction
Written in Java as an Eclipse plug-in, CViMe’s object modeling of C and C Preprocessor (CPP) allows the generation of specific views of a source based on a set of defined flags. CViMe hides non-compiled code through code folding techniques and annotates areas of interest such as common and conditionally compiled blocks. Navigation to CPP-controlled blocks specific to a subset of directives is possible from the CViMe perspective, which narrows the scope of source maintenance. Moreover, in an effort to further reduce the presence of preprocessor directives, CViMe provides the ability to refactor common, conditionally-compiled blocks into reusable units using macro substitution. CViMe runs on C/C++ Development Tooling (CDT) [3], and both share the same set of configuration flags such that the semantic index generated by CDT matches the CViMe view, and full language support is available in the editor.

At the moment, semantic language support via CDT is only available for the compiled sections of code.

2. Design
The underlying CViMe engine is a custom designed, single-pass, streamed compiler front-end written in Java that relies on object-modeling to not only fold non-compiled code, but refactor conditionally compiled blocks into reusable units like macros, significantly reducing the presence of preprocessor directives at the editor level.

Categories and Subject Descriptors  D.2.3 [Software]: Coding Tools and Techniques - object-oriented programming, program editors.

General Terms  Design, Languages

Keywords  C preprocessor, conditional compilation, crosscutting concerns, Eclipse plug-ins, editors, Java object modeling, macros, refactoring, reusability, source views.

1. Introduction
Written in Java as an Eclipse plug-in, CViMe’s object modeling of C and C Preprocessor (CPP) allows the generation of specific views of a source based on a set of defined flags. CViMe hides non-compiled code through code folding techniques and annotates areas of interest such as common and conditionally compiled blocks. Navigation to CPP-controlled blocks specific to a subset of directives is possible from the CViMe perspective, which narrows the scope of source maintenance. Moreover, in an effort to further reduce the presence of preprocessor directives, CViMe provides the ability to refactor common, conditionally-compiled blocks into reusable units using macro substitution. CViMe runs on C/C++ Development Tooling (CDT) [3], and both share the same set of configuration flags such that the semantic index generated by CDT matches the CViMe view, and full language support is available in the editor.

At the moment, semantic language support via CDT is only available for the compiled sections of code.

2. Design
The underlying CViMe engine is a custom designed, single-pass, streamed compiler front-end written in Java that relies on object modeling of both common blocks and the conditionally compiled sections of a C/C++ source. The non-object oriented source is converted into translation objects, and paths down a compacted abstract syntax tree correspond to different compilation branching for a given file in a C project. Omitted paths are folded in the C editor using Eclipse’s code folding features. The working set of files affected by a group of chosen configuration flags can be further narrowed by searching for conditionally compiled sections controlling similar code blocks. This allows the developer to refactor those sections into conditionally defined macros, provided they contain identical logic. This replaces scattered and tangled blocks with macro definitions, leading to cleaner code. This reduced scope additionally facilitates the consideration of alternate forms of modularization such as aspects [4].

3. Demonstration
The demonstration will be split into two parts: performance and features.

3.1. Performance
Performance test will include execution time for scanning and parsing C modules containing large sources for well-known software systems like Mozilla’s Firefox, and FreeBSD, as well as IBM’s port, thread, and native Java class libraries used in the open source Apache Harmony project [5].

3.2 Screenshot
The features portion of the demo will focus on a module containing a subset of one of the software systems presented in Section 3.1.

Figure 1 shows a screenshot of the CViMe perspective in Eclipse. The right hand side shows a configuration widget listing all of the CPP conditional flags used in a Mozilla Firefox module. Upon selecting a set of flags from the list, the plug-in updates the working set of files and lists only those containing the selected flags. Opening a file from that working set results in a view specific to the configuration selected from the widget.

Other features of CViMe are listed in section 3.3
3.3 Features

The demonstration will include:

1. **Opening a project**: CViMe will build translation units and populate an index table when an Eclipse project is opened in the CViMe perspective. All the flags parsed from the source are displayed in a widget, and can be selected to generate a particular view of the source (see Figure 1). Flags can also be grouped into specification categories and read from an external XML file.

2. **Configuration selection**: Upon selecting a configuration option from the configuration widget a reduced list of files are displayed as a working set in the Package Explorer.

3. **Opening a file**: Opening a particular file using an extension of the C Editor used by CDT will display all non-compiled code folded around the controlling preprocessor directives. CDT language support will be available for all compiled sections.

4. **Error detection**: CViMe’s enhanced error detection will be demonstrated, including the detection of unrecognized CPP directives, malformed CPP controlling expressions and blocks, like missing #endifs. Also provided are warnings for directives that do not start on a new line, such as non-comment tokens occurring after an #endif but prior to a newline character that are not detected as errors by the Microsoft compiler, but caught by CViMe.

5. **Refactoring**: An Eclipse view will indicate all syntactically similar, conditionally compiled C blocks within a user specified syntactic distance. The view can be expanded beyond the working set to include the entire C project. These blocks can be refactored into macros, or alternately, the developer can consider other technologies for separation of concerns like AspectC [4]. Future versions will use a more robust C analyzer that relies on semantic comparison.

About the Authors

Nieraj Singh is a graduate student at University of Victoria, and his research focuses on configuration of large systems, conditional compilation, and modularity of crosscutting concerns. He is the primary architect and developer of CViMe.

Yvonne Coady is an Assistant Professor in the Department of Computer Science at University of Victoria. Her research focuses on OS, JVM and middleware infrastructure and modularity, as well as Aspect-Oriented Programming and Software Development.

Graeme Johnson, Andrew Low, and Trent Gray-Donald are IBM developers and J9 JVM Development Leads based at the IBM Queensview Lab in Ottawa.

Presenters

Graham Johnson, IBM Ottawa
Andrew Low, IBM Ottawa
Trent Gray-Donald, IBM Ottawa

References


