Using Eclipse CDT for C/C++ Development

Sebastien Marineau-Mes
Outline

- Who we are
- CDT project: goals and challenges
- CDT architecture
- Feature set
- Roadmap
- Example integrations
CDT – A Bit of History

- CDT – C Development Tooling

- Project launched July 2002
  - Provide C/C++ development under Eclipse
  - Integrate with existing C/C++ command-line tools (compiler, debug etc)
  - Built as extensible and replaceable building blocks

- Milestones
  - CDT 1.0 Dec. 2002
  - CDT 1.1 (May)
  - CDT 1.2 (Oct)
  - CDT 2.0 planned for June 2004 (sync with Eclipse 3.0)

- CDT committers
  - QNX and IBM Rational
  - 20+ person-years of effort
Where We Are Today

- Downloads
  - Enthusiasts
  - Mostly Windows and Linux
- Adoption in commercial products
  - QNX Momentics development suite
  - IBM WSDD
  - Timesys Timestorm
  - Tensilica Xtensa Xplorer
  - Redhat Enterprise Linux
  - Montavista DevRocket
- IDE prototypes
  - Altera
  - PalmSource
  - Intel
  - Rockwell Collins

- Participating companies
  - TimeSys
  - Tensilica
  - Red Hat
  - Montavista
  - Intel
  - Rockwell Collins
  - Real-time Innovations
  - Altera
  - PalmSource
  - Ericksson
  - Nortel
  - Wind River
  - others…
CDT Project Goals

- First-class framework for C/C++ tooling in Eclipse
  - Platform-neutral framework to support variety of development scenarios
  - As full-featured as the JDT (!)
- Extensible and interoperable
  - Provide powerful base functionality and allow extending/replacing features
  - Well-defined APIs for interoperable extensibility
  - CDT common integration point for all C/C++ tooling
- Cooperative
  - Pooling of resources for base C/C++ tools components
  - Well-defined “value-add” from contributing companies
Where CDT is Being Targeted

- Traditional embedded
  - C/C++ development in host-target paradigm
  - CDT as integration point of embedded tooling
- Desktop/server
  - Linux-based self-hosted systems (non-Windows)
  - Opportunity for full-featured C/C++ IDE
- Deeply embedded
  - SW/HW co-design, soft cores, FPGA
  - Interest in C, assembly; very simulator-centric

Targeted environments drive default toolchain decisions
- Default CDT implementations target GNU
- gcc most widely used for build/compile
- gdb debugger widely available
- Provide additional integration hooks for other toolchains
Specific C/C++ IDE Challenges

- No “control” over back-end tools
  - Compiler, debugger, toolchains, build system
- C/C++ language challenges
  - Parsing challenges
  - Language variants
  - Complexity of C++
- Preprocessor
  - #defines can be in source, headers, or “inside” build system
  - Needed to properly parse source
General CDT Architecture

Eclipse platform

- **Workbench**
  - JFace
  - SWT
- **Team**
- **Debug**
- **Workspace**

Platform Runtime

- **CDT Core**
  - C/C++ editor
  - Parsing, outline, indexing
  - Code completion
  - Search
- **CDT Build**
  - Project builds
  - Make and Managed Make
- **CDT Launch**
  - Launch run/debug session
  - Glue between Core and Debug
- **C/C++ Debug**
  - C/C++ debug perspective
  - Memory, regs, Asm views
  - CDI interfaces
  - GDB/MI plugin

Using Eclipse CDT for C/C++ Development
Using Eclipse CDT for C/C++ Development

CDT Core Architecture (partial)

- Project files
  - hello.c
  - hello.h
  - hello.cpp

- Editor

- Outline View

- Compiler
  - gcc, g++, others

- Build Tools
  - Make scripts

- Code Assist

- Search

- Class Browser

- Refactor

- Build Console
  - Error Parsing

Outline View

Index AST

C/C++ Parser
CDT Core Features

- Editor
  - C/C++ syntax highlighting
  - Code completion
  - Hover help

- Parser
  - Parses source files in project to extract C/C++ elements
  - Information used for search, outline, code completion

- Search
  - API and extension points to allow extensibility

- C++ Development
  - Class creation wizards
CDT Build Features

- Standard Make
  - Re-uses existing makefiles
  - Simple integration with arbitrary build systems
  - Parsing of toolchain output to generate error markers
- Managed Make
  - Manages compiles and toolchain directly
  - No makefile editing
  - Fine control over compile, link settings
CDT Debug Features

- Portable source-level debugger
- Various views that extend Eclipse debug framework
  - Registers
  - Memory
  - Signals
  - Shared libraries
- CDI (C Debugger Interface)
  - MI plugin implementation (interface to GDB through machine-independent interface)
  - Support for MI level 1 and 2
  - Integrates with gdb version 5.2.1 and above
  - Allows targeting of a wide variety of CPU architecture
  - CDI APIs allow programmatic control over debugger
# CDT Roadmap

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1.2 GA</td>
<td>1.2.1 GA</td>
<td>2.0 GA</td>
<td>2.x</td>
</tr>
</tbody>
</table>

- **Improved parser**
- **Search/navigation**
- **Wizards (C++)**
- **Managed builds**
- **Debug updates**
- **GDB MI v.2**

- **Maintenance update**
- **Eclipse 2.1.x based**

- **Based on Eclipse 3.0**
- **Integration API**
- **CDOM, refactoring**
- **Managed build updates**
- **Class browser**

---

## Embedded Extensions

**Define other frameworks:**
- **Target abstraction**
- **Tracing**
- **Profiling**
CDT 1.2 Release Highlights

- Feature enhancements
  - Search, indexer, parser
  - First iteration of managed build
  - Debugger enhancements (inc. GDB MI/2)
- Improved testing
  - All platforms tested
  - Significant test coverage
- Documentation – user’s guide
- Improved project management
  - Planning and tracking
  - Builds
  - Web site
CDT 2.0 Plan – Key Themes

- Improved out-of-box experience for new users
- Eclipse 3.0 support and UI compliance
- Productization: I18N and accessibility
- Enhance capability of code modeling capabilities
  - Parser, indexer, AST/DOM
- Enhance capability of end user functionality
  - Search, content assist, managed build
  - Debug, breakpoints, expressions
- Performance improvements
  - Source management in debugger
  - Indexing and parsing
  - View bookkeeping in debugger
- New Features
  - Class browser, refactoring
  - Mixed source/assembly presentation in debugger
- ISV documentation
  - CDT core APIs
  - CDI debug API
CDT 2.0 Parser/AST/DOM

- Improve accuracy of parser
- Improve reporting of parse errors
- Improve coverage of gcc/g++ language extensions
- Support for selection search and content assist
- Complete AST, the beginning of the DOM
- Language variants – support through extension mechanism
C/C++ Search/Indexer/Content Assist

- Improve performance and scalability of indexer
- Support search based on selection in editor
- Support content-assist completion of C++ elements
  - Use parser for accuracy
  - Don’t forget C - link time scoping
- Support proper parsing in standard make build environment
  - Need build options
Managed Build

- Support multiple targets, build goals, tool chains, configurations per project

- Enhance usability
  - Setting of options on multiple configurations at a time
  - Separation of build goal from target
  - Reasonable defaults at project creation time (improve out-of-box)

- Support different types of tools (e.g. bison/flex, gcj).

- Support for different make utilities (maybe even non-make?)
Refactoring

- Similar to JDT
  - But with simpler framework

- Start with rename
  - Simply search and replace

- Move onto fancier refactorings once DOM is writable (post 2.0)

- Undo manager
  - To manage undo across multiple files
CDT 2.0 Debug Features

- **CDI**
  - Revision
  - Documentation
  - Abstract implementation
- **Breakpoints**
  - Deferred breakpoints
  - Thread-restricted breakpoints
  - Breakpoints in external files
- **Expressions**
  - Make persistent across workbench sessions
  - Add action to the Expressions view
- **Source Management**
  - Performance improvements
  - Duplicate files
  - Prefix mapping

- **Editors**
  - Mixed disassembly/source presentation
- **Memory View**
  - New design and implementation
- **Registers View**
  - Registers bookkeeping (query on demand)
- **Variables View**
  - Improve detail pane
- **UI Improvements**
  - Usability
Example Integration - Extending the Debugger

Debugger launch

- Standard debug launch for plain gdb, cygwin and gdb server
- May need to start session with own flavor of gdb, custom options
- May need to customize to perform additional steps, for instance:
  - Start simulator
  - Download code to target
  - Download additional files to target
  - Start extra tools
Example Integration – CDT Debugger

CDT Debug Core

Implementation of Eclipse Debug Core

Specific C / C++ Interface

CDI

CDT Launch

Extension point

GDB / MI

Cygwin / MI

My GDB flavor

My GDB flavor
CDT Debug Extension Points

You need to implement two debugger extension points:

```xml
<extension
    point="org.eclipse.cdt.debug.core.CDebugger">
    <debugger
        platform="native"
        name="%GDBDebugger.name"
        modes="run,core,attach"
        cpu="native"
        class="org.eclipse.cdt.debug.mi.core.GDBDebugger"
        id="org.eclipse.cdt.debug.mi.core.CDebugger">
    </debugger>
</extension>

<extension
    point="org.eclipse.cdt.debug.ui.CDebuggerPage">
    <debugPage
        class="org.eclipse.cdt.debug.mi.internal.ui.GDBDebuggerPage"
        id="org.eclipse.cdt.debug.mi.GDBDebuggerPage"
        debuggerID="org.eclipse.cdt.debug.mi.core.CDebugger">
    </debugPage>
</extension>
```

Code starting points:
- `org.eclipse.cdt.debug.mi.core` -> `GDBServerDebugger.java`
- `org.eclipse.cdt.debug.mi.ui` -> `GDBServerDebuggerPage.java`
- about ~200 lines
Example of integrating with CDT – Hover Text

What hover help does
- Provides additional pop-up help in C editor
- Can provide info, tips, documentation on API calls
- Information can come from indexer, or from external contribution

Why provide an external contribution
- API summary for binary libraries
- API docs for system libraries

How to do this
- Implement the “org.eclipse.cdt.ui.textHovers” extension point
  `<extension-point id="textHovers" name="$textHoversName"/>

Example
- Redhat built text hover plugin that provides help on Linux APIs
- Help info extracted from man pages
Where CDT is Going

- Long-term features
  - Integrated Java and C/C++ development
  - Non-gdb based debugger
  - Project templates
  - Parse errors and quick fix
  - Autoconf/automake support
- Increase adoption of CDT
  - Make it easier for ISVs to ship Eclipse/CDT
  - Encourage integrations with CDT
- Extend CDT to embrace embedded needs
- Provide “mix-and-match” platform for C/C++ tooling
Extending CDT for Embedded – Some Ideas

- Multiprocessing Extensions
- Code Analysis
- Language Extensions
- IDE Extensions
- GNU Extensions
- Optimizing Compilers
- GNU Tool Chain
- C/C++ Build
- C/C++ Code Development
- C/C++ Debugging
- GNU Extensions
- ICE/JTAG
- Other
- Simulator
- GDB-MI
- Modeling
- BSP builder
- Memory
- Tracing
- Application Profiler
- Embedded Java Tooling
- Integrated Java/C/C++
- C/C++ Developer Tooling
- Performance and Trace
- Target Abstraction
- Performance
- Java Developer Tooling

Using Eclipse CDT for C/C++ Development
Ultimate Deployment Scenarios

- Automated Test
- QNX MultiCPU Debugger
- XDE for C++
- Code Analysis
- SlickEdit Editor
- C/C++ Debugging
- C/C++ Code Development
- Timesys Extensions
- GNU Extensions
- GNU Compilers
- Intel Compilers
- Tensilica Simulator
- WIND JTAG
- GDB-MI
How to Contribute to CDT

- As a user of CDT
  - Download and use it for your C/C++ development
  - Provide feedback on features, usability
  - Suggest improvements
  - Report bugs
- As a developer of CDT
  - Provide patches and bugfixes
  - Implement features
  - CDT has no shortage of “hard” problems to solve
- Other areas
  - User documentation, How-to’s, FAQ
  - Example integrations
  - Plugins that extend CDT

Want to get Involved? Visit www.eclipse.org/cdt!
Conclusion

- A lot of community interest in CDT
- Several commercial products shipping with CDT
- CDT feature set/architecture evolving based on feedback and needs
- CDT 2.0
  - Significant feature enhancements
  - Brings CDT in sync with base Eclipse platform

- We would love for you to get involved
  - As a user and developer
  - Several areas in need of contributions and leadership