eclipseCON 2006

Adding a New Language to the CDT

Jeff Overbey & Spiros Xanthos
University of Illinois at Urbana-Champaign • Software Architecture Group
Photran Committers • http://www.eclipse.org/photran

 March 20, 2006

Welcome

- **Why we're talking about this:**
  - We are working on Photran, a CDT-based Fortran IDE
  - By building on the CDT rather than the basic Eclipse platform, one can take advantage of:
    - Outline view
    - Make builder
    - Makefile generator
    - The gdb-based debugger
    - Binary launcher, and
    - Compiler error extraction facilities

 |

Jeff Overbey

Jeff Overbey is a PhD student at the University of Illinois a Urbana-Champaign and a committer on Photran, an IDE and refactoring tool for Fortran. Photran recently changed from being a standalone Eclipse-based IDE to being a CDT add-in providing Fortran support. As part of this effort, he and the rest of the Photran team have been working closely with the developers of CDT to define a standard mechanism for adding additional languages to the CDT.

 |

Spiros Xanthos

Spiros Xanthos is a PhD student at the University of Illinois at Urbana-Champaign. His research interests are mainly in the areas of Software Engineering and Systems. He is an Eclipse committer on Photran, a Fortran IDE and refactoring tool based on the CDT. As a member of the Photran team works closely with the developers of CDT to define a standard mechanism for adding additional languages to the CDT.

 |

Introduction

- What CDT is :
  - A set of plug-ins that support C and C++ development
  - With out-of-the-box support for gnu tools (gcc, g++, gdb, binutils)
  - Platform for integrating other toolchains (compilers, debuggers, etc)
  - Platform for adding other C/C++ tooling (code analysis, documentation generation, unit testing)
- History of CDT multi-language support :
  - Started with Photran and PTP
  - Other attempts within the Eclipse community (ldt)

 |

We 'll cover

- The CDT architecture
- CDT changes for multi-language support
- Implement an actual IDE for *Eightbol*
- Managed Build System integration
- Refactoring support
- Current state and future directions

 |

When we finish...

- You should:
  - Understand the architecture of the CDT
  - Know what CDT components you can reuse
  - Know how to extend the CDT components for a new language
  - Know what need to add to create a full IDE
- You shouldn't:
  - Throw things to the presenters

 |

Before We Start...

- You will need:
  - A laptop
  - Eclipse >3.1
  - Java 5
  - Patience :)

 |

Schedule

- 8:30–8:45 Welcome, introduction, overview
- 8:45–9:00 CDT overview, architecture
- 9:00–9:30 CDT changes for multi-language support
- 9:30–9:45 *(break)*
- 9:45–11:00 Eightbol tutorial
- 11:00–11:30 Photran (case study) and refactoring support
- 11:30–12:00 Future directions, discussion, etc.

CDT Architecture 8:45–9:00

- The CDT consists of the following components:
  - Core
  - UI
  - Debug
  - Launch
  - Doc
  - Refactoring

 |

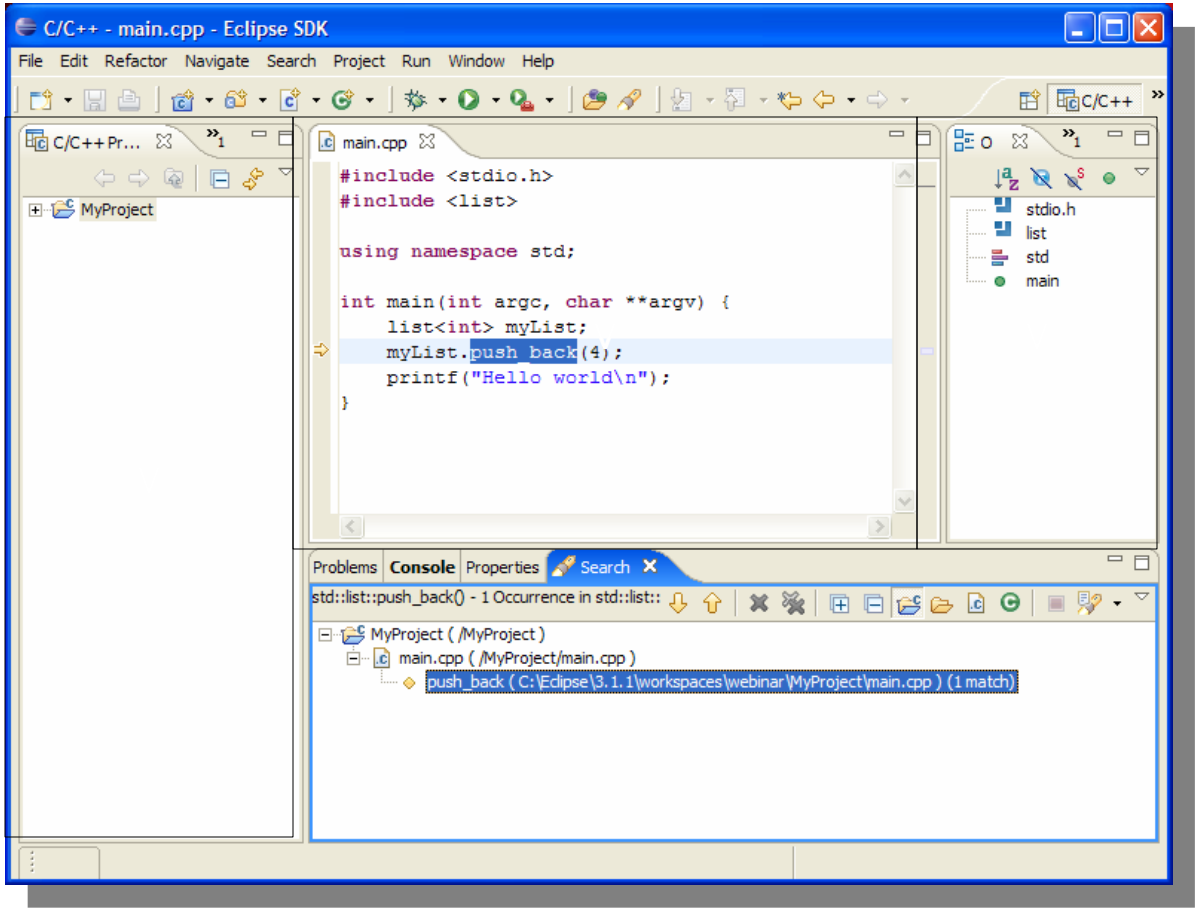
CDT Core

- Cmodel/CDOM
  - C/C++ AST
  - Used the Outline View and “Reactoring”
- Full blown C/C++ parser and Indexer
- Binary parser
  - Extracts symbol information from libraries, objects and archives
- Project builder:
  - Invokes external tools (e.g make) for building and
  - Parses external tool output for problems and errors

 |

CDT UI

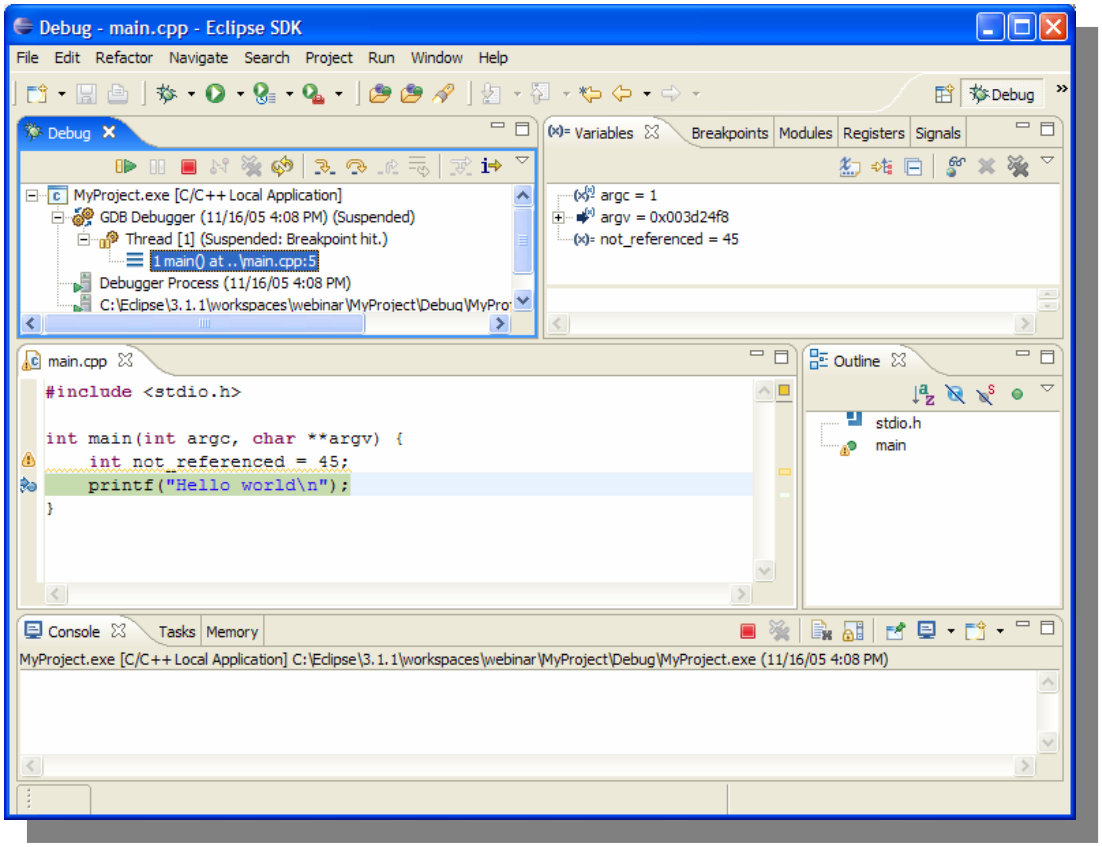
- C/C++ Projects View
  - C++-centric view of the project
- C/C++ Editor
  - syntax highlighting
  - code assist,
  - hover help
  - include assistance
  - Templates
- C Outline View
- C make view
- Wizards for creating C and C++ projects



 |

CDT Debug

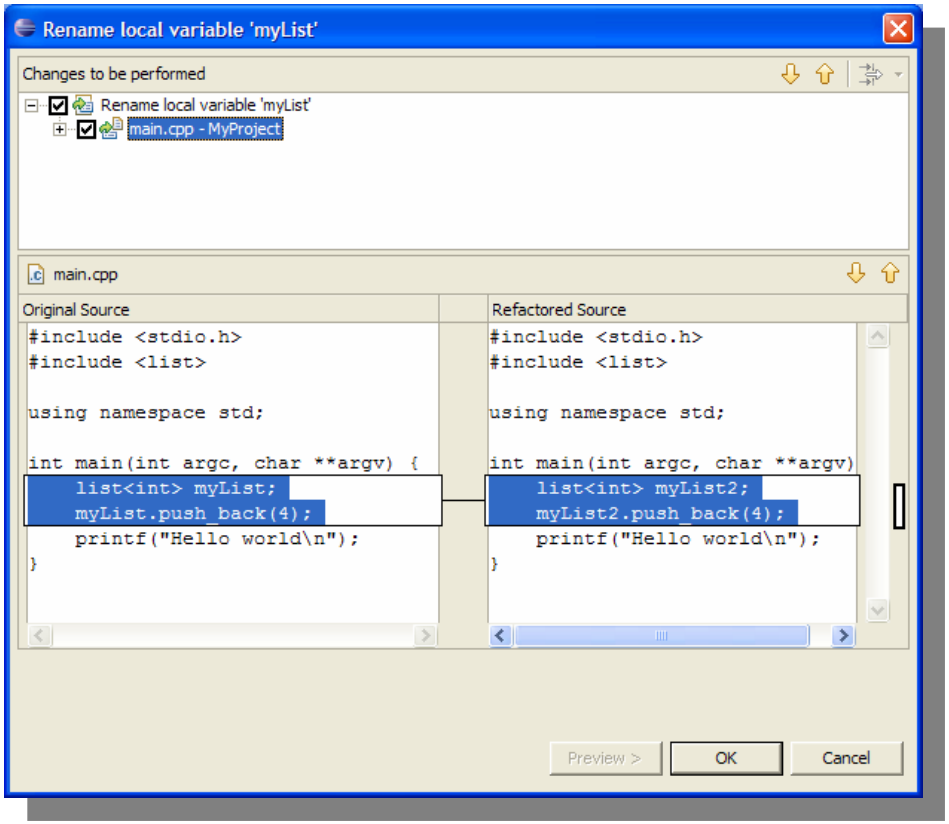
- Provides a C/C++ debug model which extends the debug model provided by the platform.
- Provides a debugger interface (CDI) which is used to "talk" to the actual debugger.
- Provides a default implementation for *gdb*



 |

CDT Refactoring

- Refactoring Support for the CDT
- Extends the ltk Refactoring framework
- Currently only *rename* refactoring



 |

...and CDT Launch

Which provides generic launch support for binaries

 |

9:00–9:30

CDT Changes for Multi-language Support

 |

Gee, This Is Convenient

Fortran programmers...

- Use CVS
- Use make
- Compile to binaries
- Debug using gdb

Gee, This Is Convenient

Fortran programmers...

- Use CVS
- Use make
- Compile to binaries
- Debug using gdb

CDT provides...

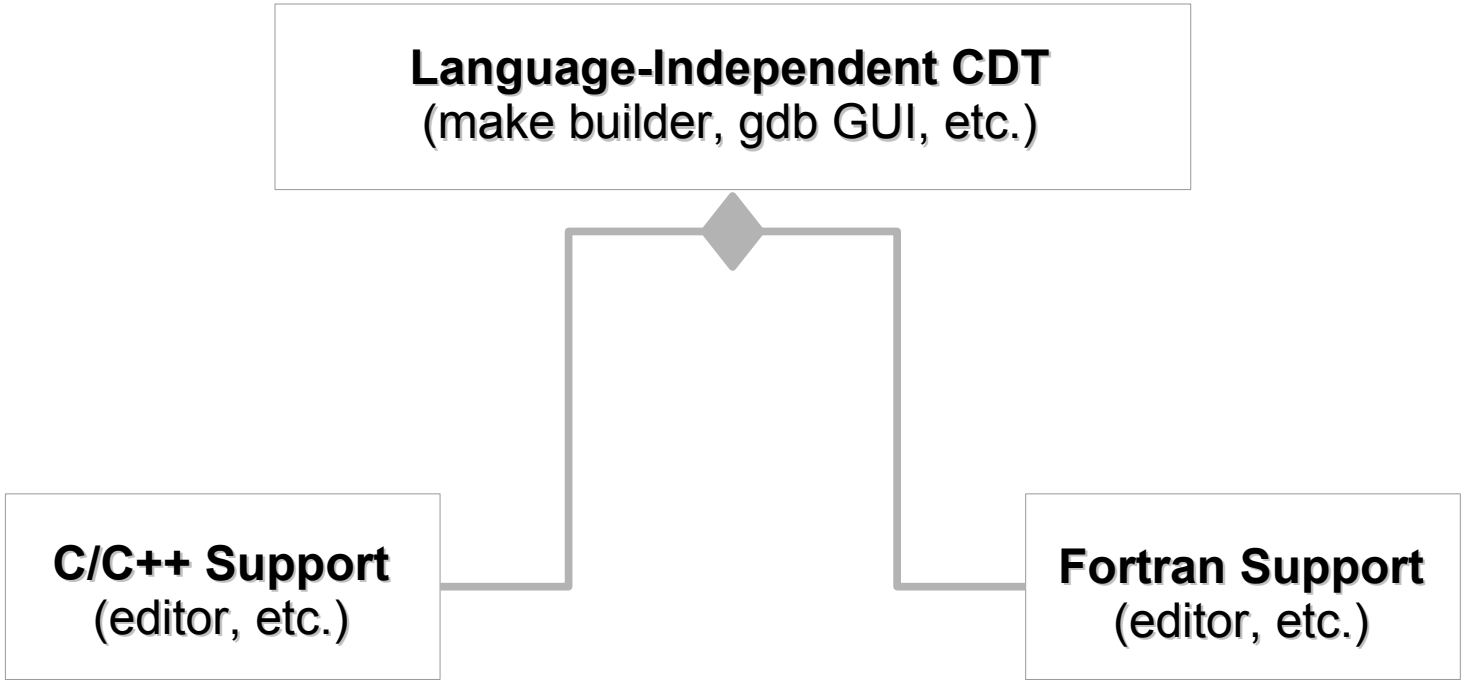
- CVS synchronization
- Make builder
- Binary launcher
- GUI for gdb

 |

The Photran Heuristic

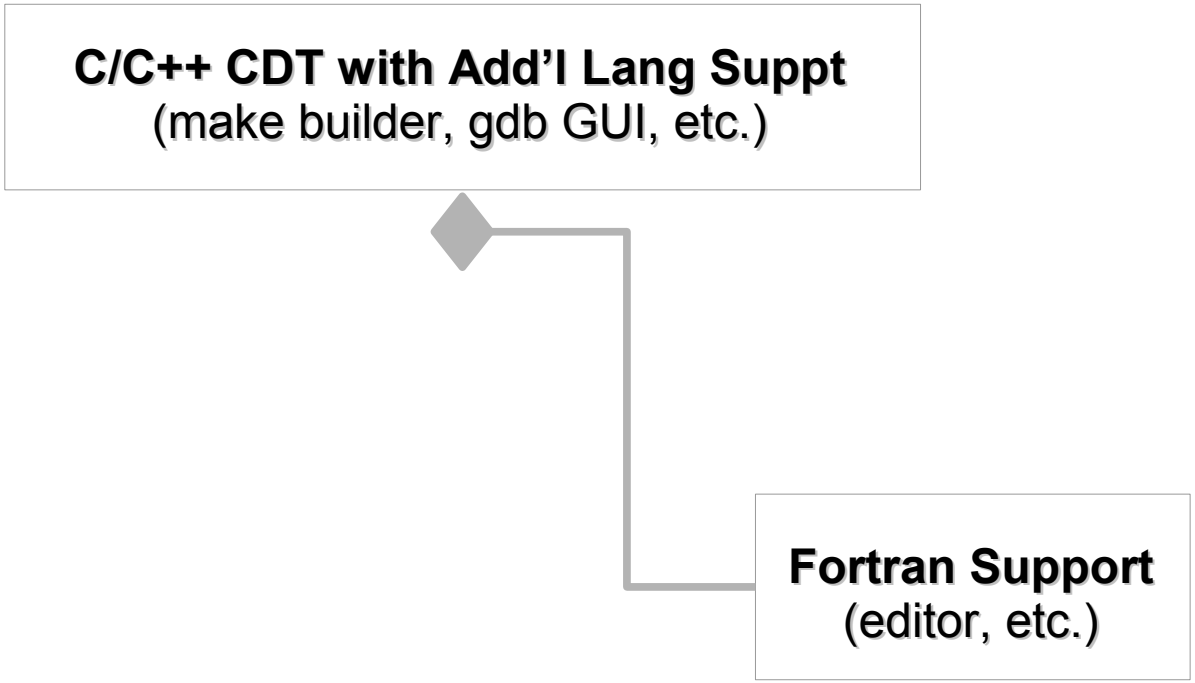
If your product development cycle involves cutting and pasting a quarter million lines of code every six months, there might be a better way to approach the problem.

What We Would Like



 |

What We Actually Have



 |

Basically...

If programs in a language are typically compiled with make and debugged with gdb, an IDE for that language can be created quickly by extending the CDT.

 |

Part One

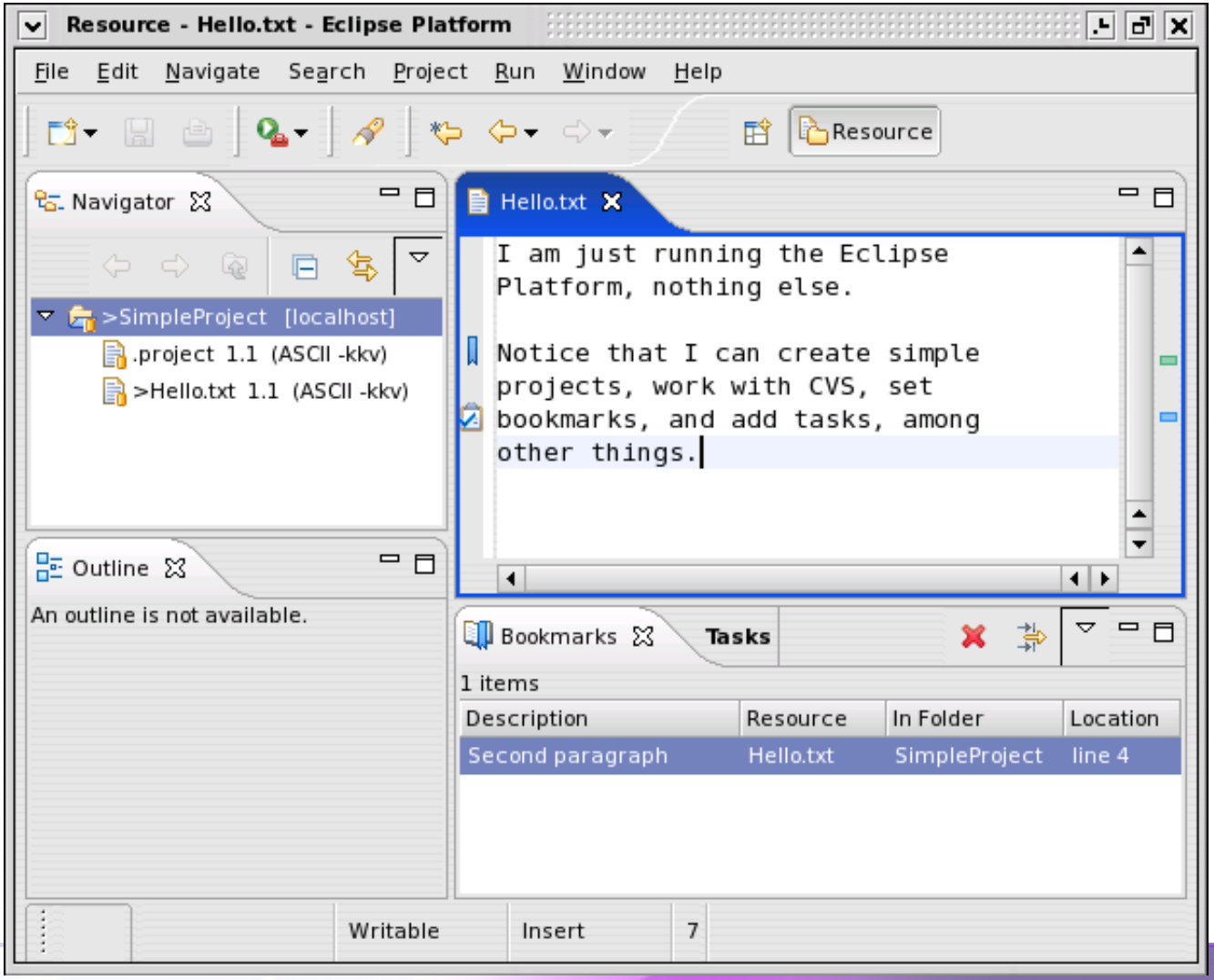
If programs in a language are typically compiled with make and debugged with gdb, an IDE for that language can be created quickly by extending the CDT.

 |

Eclipse vs. Eclipse+CDT

Eclipse provides...

- Projects
- Team/CVS
- Frameworks
  - editors
  - debuggers
  - etc.



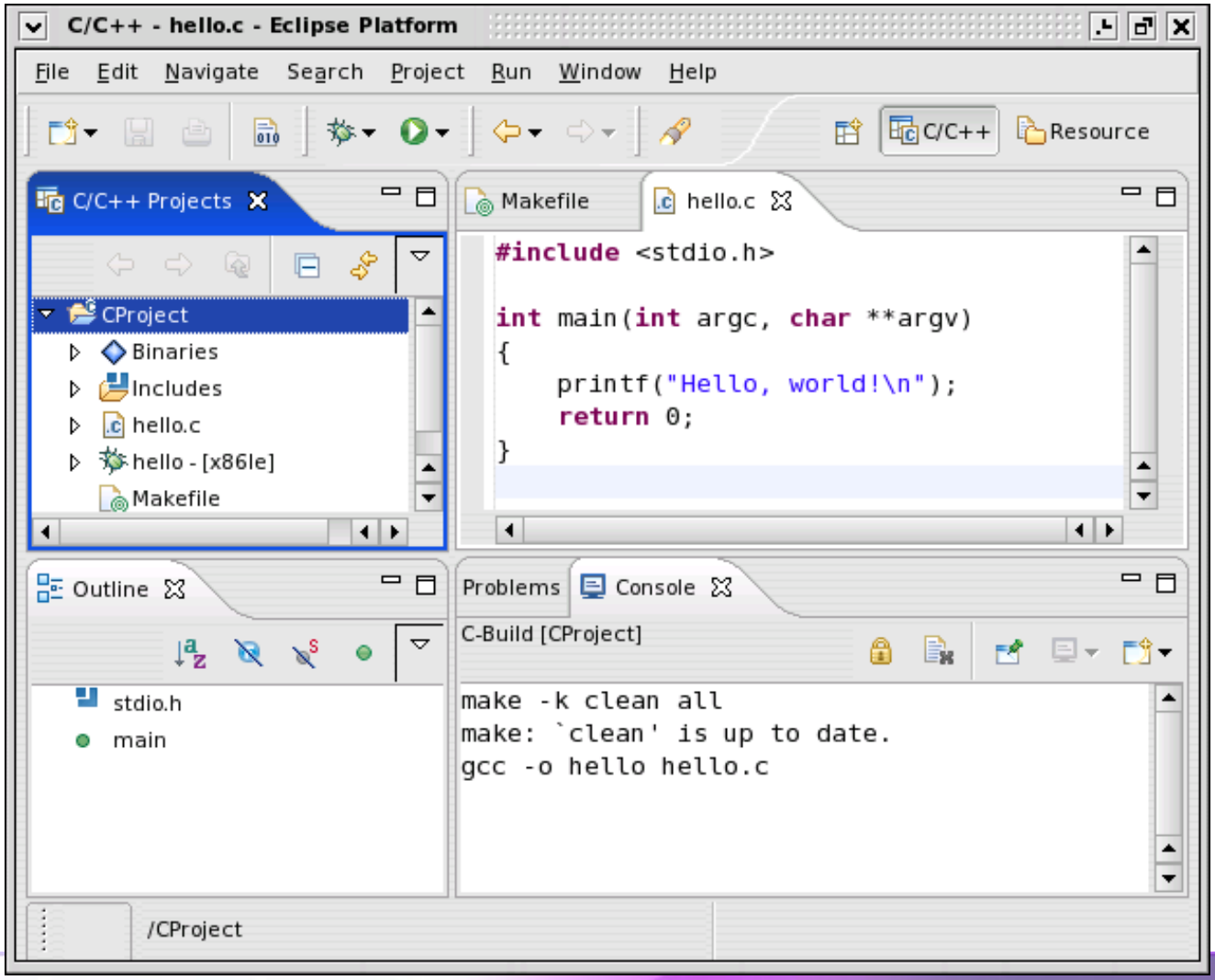
 |

Eclipse vs. Eclipse+CDT

CDT provides...

(among other things)

- Make builder
- Binary launch
- GUI for gdb



 |

Eclipse vs. Eclipse+CDT

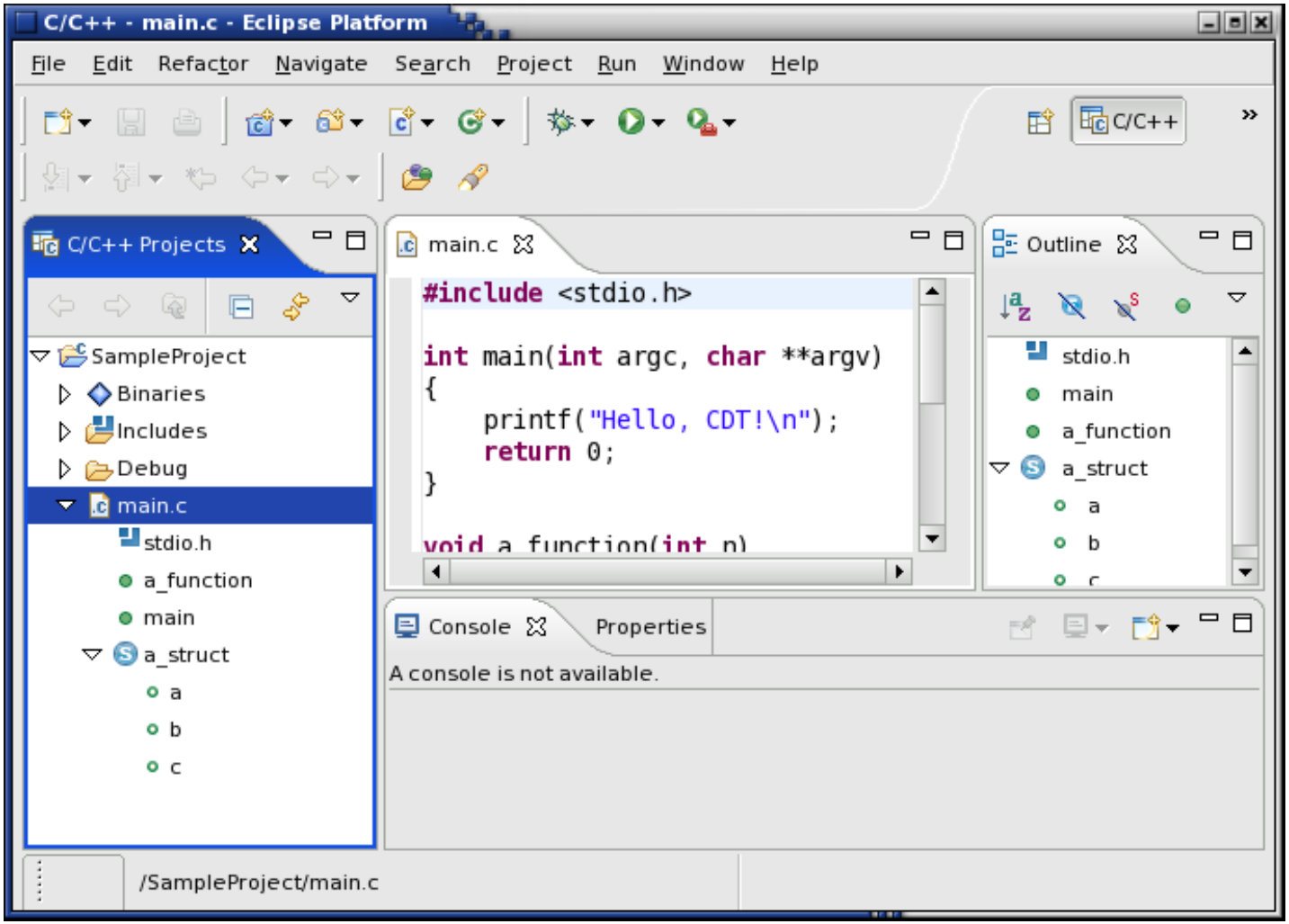
Eclipse provides...
- Project support
- Team support (CVS)
- Frameworks

CDT provides...
- Make builder
- Binary launcher
- GUI for gdb

Build on the CDT when programs in your language are typically compiled with make and debugged with gdb.

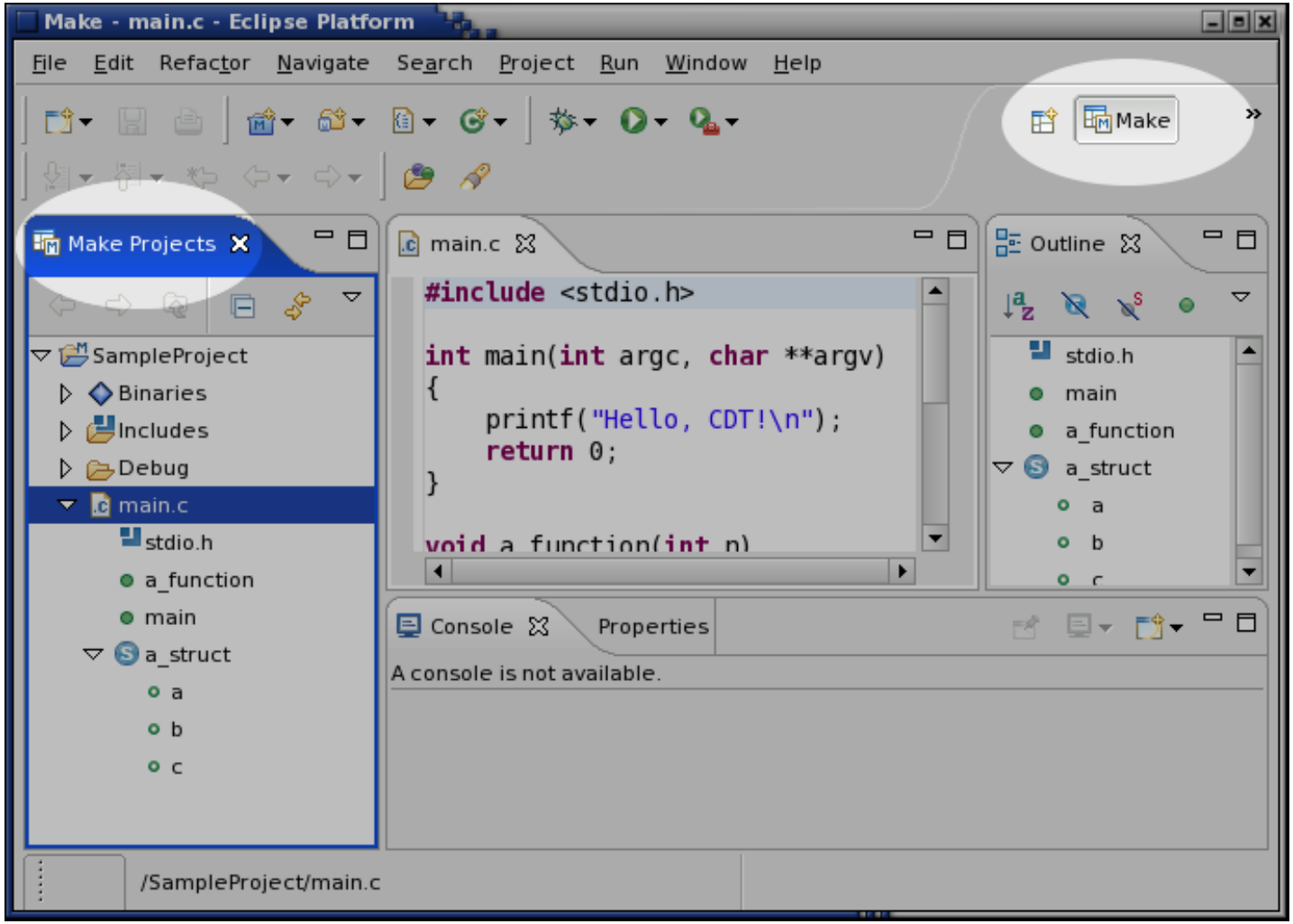
 |

CDT Changes: User Interface



 |

CDT Changes: User Interface



 |

CDT Changes: Core

- New extension point
  - Plug new languages into the CDT

 |

Part Two

If programs in a language are typically compiled with make and debugged with gdb, an IDE for that language can be created quickly by extending the CDT.

 |

How to Extend the CDT (you'll be doing this in a few minutes...)

- Create a “model builder”
  - Produces Outline view for files in your language
- Create an editor
  - Nothing CDT-specific, except...
  - Reuse the CDT Outline page
- Plug into the new extension point
  - List of content types (filename extensions)
  - Model builder

(more detailed descriptions follow)

 |

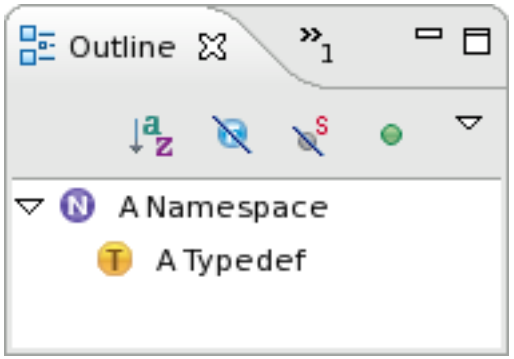
Step 1 of 3: Create a Model Builder

```
public class TrivialModelBuilder implements IModelBuilder
{
   public Map parse(boolean quickParseMode) throws Exception
   {
      Map newElements = new Map();

      // Create a namespace as a child of the translation unit
      Namespace ns = new Namespace(translationUnit, “A Namespace”);
      translationUnit.addChild(ns);
      newElements.put(ns, ns.getElementInfo());

      // Create a typedef as a child of the namespace
      TypeDef td = new TypeDef(ns, “A Typedef”);
      ns.addChild(td);
      newElements.put(td, td.getElementInfo());

      // No parse errors were encountered
      translationUnit.getElementInfo().setIsStructureKnown(true);
      return newElements;
   }
}
```



 |

Step 1.5 of 3: Support New Outline View Elements

```
public abstract class SampleElement extends SourceManipulation
    implements ICElement, IParent, ISourceReference, IAdditionalLanguageElement
{
    public SampleElement(Parent parent, String identifier)
    {
        super(parent, identifier, -1);

        // To set position information within the file:
        // setIdPos(offset, length);
        // setPos(offset, length);
        // setLines(startLine, endLine);
    }

    public Object getBaseImageDescriptor()
    {
        return MyPlugin.getImageDescriptor("icons/sample.gif");
    }
}
```

 |

Slide 2 of 3: Create an Editor

- Nothing special (just a regular old Eclipse editor), except...
- Reuse the CDT Outline page
  - Reuse the CDT's document provider
  - Set up syntax highlighting by overriding **doSetInput**
  - Integrate the Outline page by overriding **getAdapter**
  - Implement **ISelectionChangedListener** to “jump” to the correct location when the user clicks on elements in the Outline view
- Allow the user to set breakpoints in your editor
  - In the ctor, call **setRulerContextMenuId(“#CEditorRulerContext”)**

 |

Step 3 of 3: Plug into the New Extension Point

- Create a class that implements IAdditionalLanguage...

```
public interface IAdditionalLanguage
{
    public String getName();
    public Collection<String> getRegisteredContentTypeIds();
    public IModelBuilder createModelBuilder(
        TranslationUnit tu,
        Map<ICElement, CElementInfo> newElements);
}
```

- ...and plug it into the CDT Core.

```
<extension point="org.eclipse.cdt.core.AdditionalLanguages">
    <language class="com.mycompany.XYZLanguage" />
</extension>
```

 |

You May Also Want To

- Create some “error parsers” (easy)
  - An error parser scans the output of make for error messages from a particular compiler and displays them in the Problems view
- Integrate into the Managed Build System
  - Allows a makefile to be generated automatically

 |

Things We Haven't Considered

- Indexer (not very good yet)
- Open Type dialog (still C/C++-specific)
- Browsing perspective (still C++-specific)
- IPathEntry hierarchy (huh?)
- CModelDeltaBuilder (huh?)
- AST integration (even if it's possible, sounds like a bad idea...)
- Search (still C/C++-specific)
- Refactoring UI (need to integrate with it or disable it)

 |

9:30 - 9:45

Break

 |

9:45–11:00

Eightbol Tutorial

 |

The Plan...

1. **Learn the Eightbol language**
2. **Download and try out the Eightbol compiler**
3. **Build an entire IDE for Eightbol**

Make sure you have these files (in eightbol.zip):

- eightbol-site.pdf
- eightbol-compiler-linux.tgz
- eightbol-tutorial.pdf
- starter-workspace.zip

 |

11:00–11:30

- Photran (case study) and refactoring support

 |

Discussion / Future Directions 11:30 – 12:00