# Codan: a C/C++ Static Analysis Framework for CDT

Alena Laskavaia
September, 2010

# Agenda

Codan – stands for - “Code Analysis”

- An overview of the framework
- The user interface
- The development status
- How to create and integrate a simple internal checker
- How to integrate external tool such as lint
- What API provided to aid in writing a checker

# Goal

To create a common components and API that are shared between static analysis tools, such as:

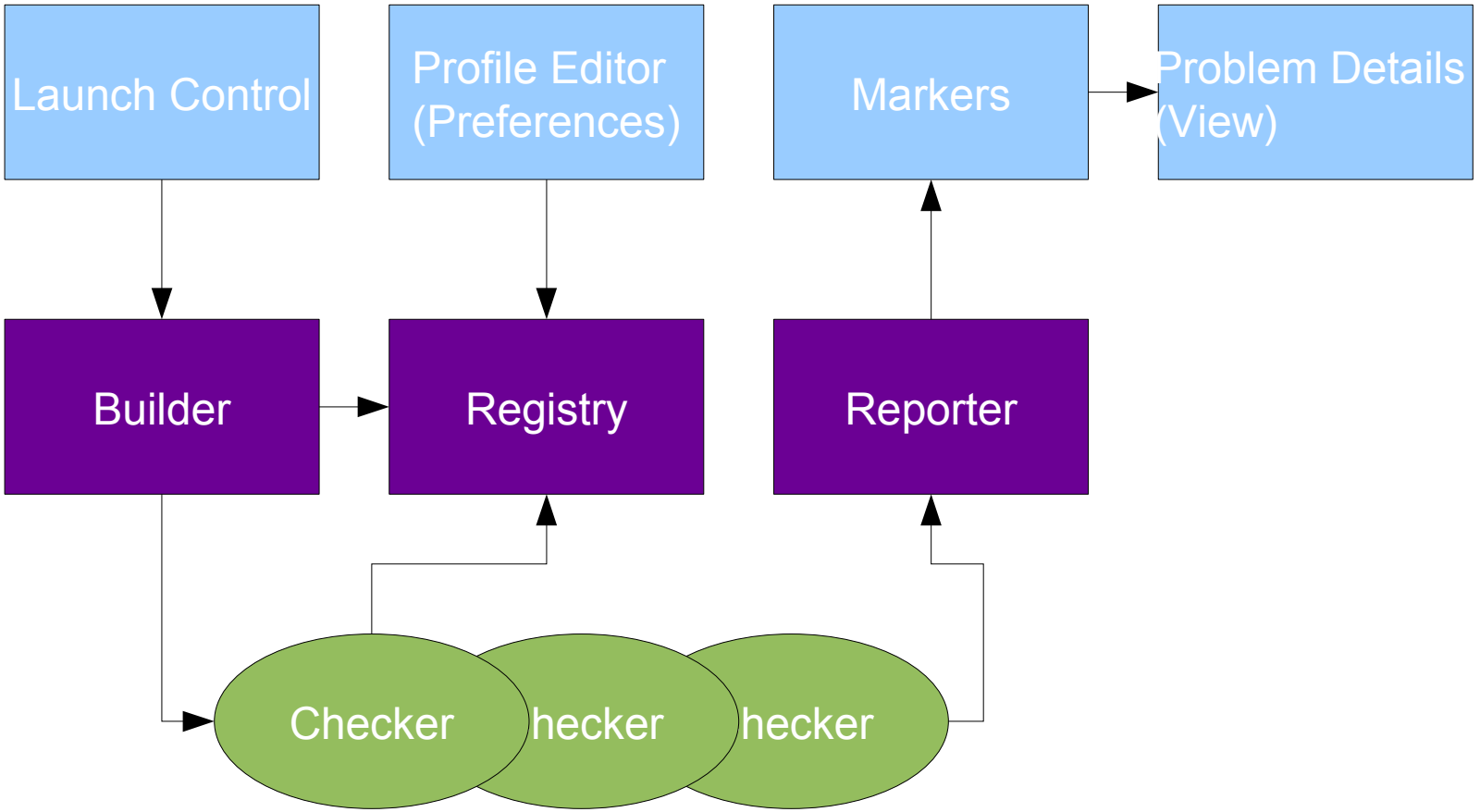
- User Interface to control the Problems enablement and parameters
- Different launch modes (as you type, on demand, as a builder)
- A view to display additional problem information
- Generic Marker type for problems with extra fields
- API to log the problems
- Abstract classes for checkers
- Sample checkers
- JUnit testing framework

Besides the framework there is also a “Checkers Feature” which has few implemented checkers and quick fixes

# Codan Users

- Tool Vendors
  - To create plugins containing end-user checkers and templates
- Software Architects, Process Enforcement people
  - To create customized new checkers, based on templates (no programming involved)
  - To create problems profiles
- Developer, Tester, Code Inspector
  - To check for errors as you type and have a quick way to fix them, during development
  - To find bugs, security violations, API violations, coding standard violations during debugging, testing, code inspection or before code commit

# Architecture



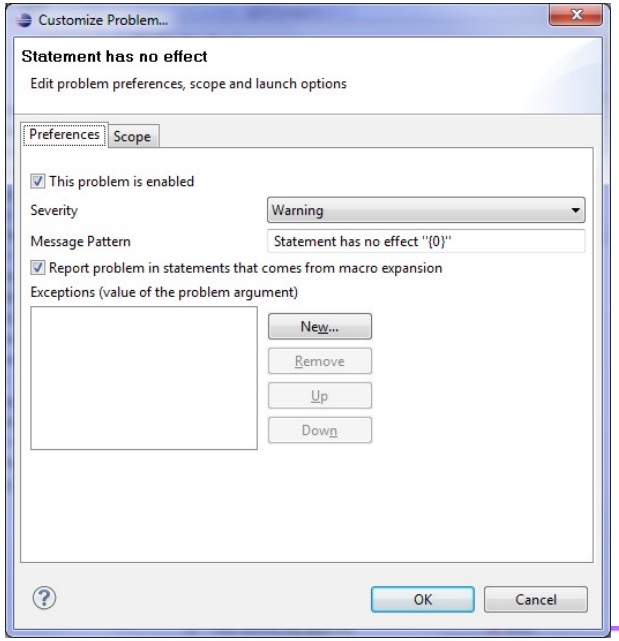
# Plugins

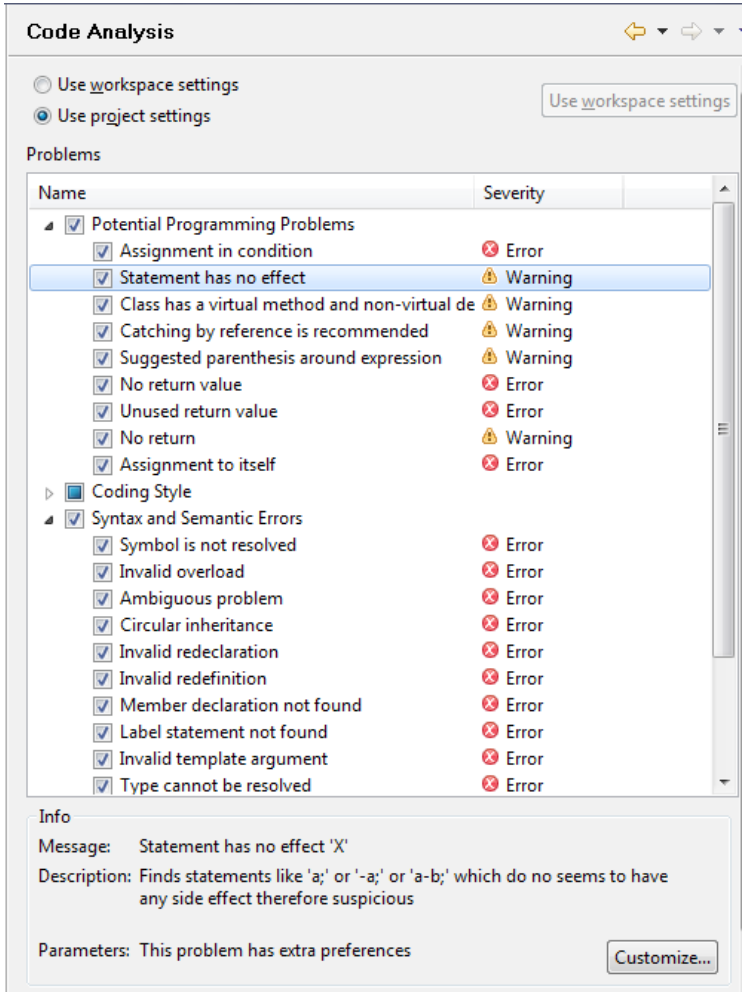
Plugins in /cvsroot/tools/org.eclipse.cdt/codan

- org.eclipse.cdt.codan-feature – codan feature
- org.eclipse.cdt.codan.core – generic core components, registry, builder, abstract checkers, all model interfaces
- org.eclipse.cdt.codan.core.cxx – C/C++ specific core components, abstract checkers, etc
- org.eclipse.cdt.codan.checkers – actual C/C++ cherckers
- org.eclipse.cdt.codan.checkers.ui – UI support for specific checkers, such as quick fix, parameter controls, detail view controls, etc
- org.eclipse.cdt.codan.ui – generic UI, preferences, launch, etc
- org.eclipse.cdt.codan.ui.cxx – specific C/C++ ui control, such CEditor listeners
- org.eclipse.cdt.codan.core.test – junit testing framework and tests for checkers
- org.eclipse.cdt.codan.examples – examples of checkers

# Profile Editor (Problem Preferences)

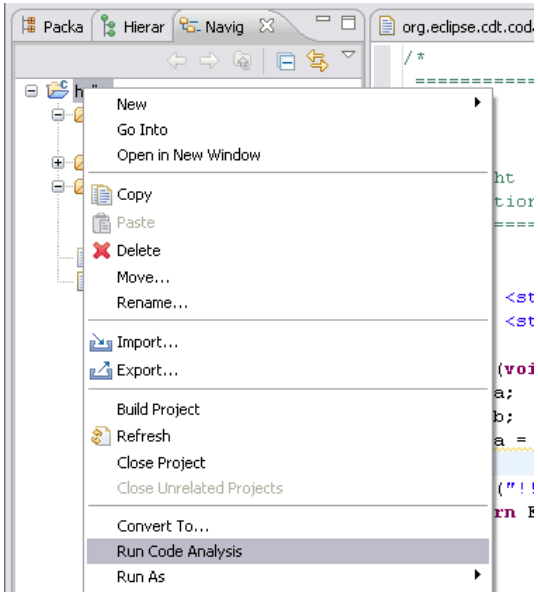
- Either checker enabled or not
- Severity of the Problem
- Info – description, message
- Customization: edit message, parameters, scope



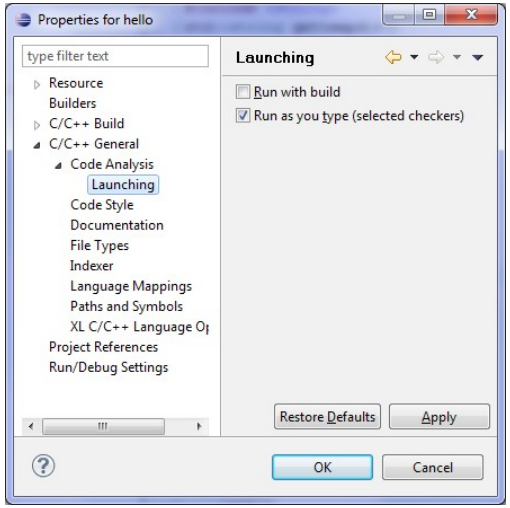



# Launch Control

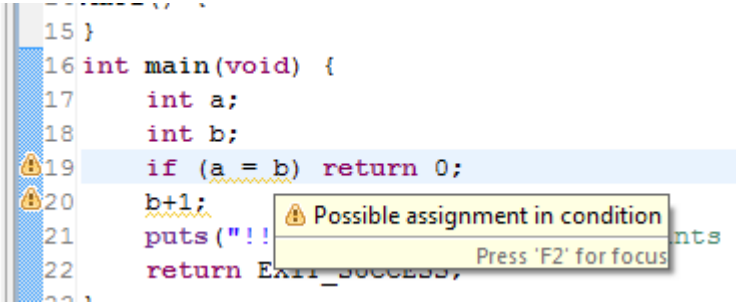
Run on demand from context menu



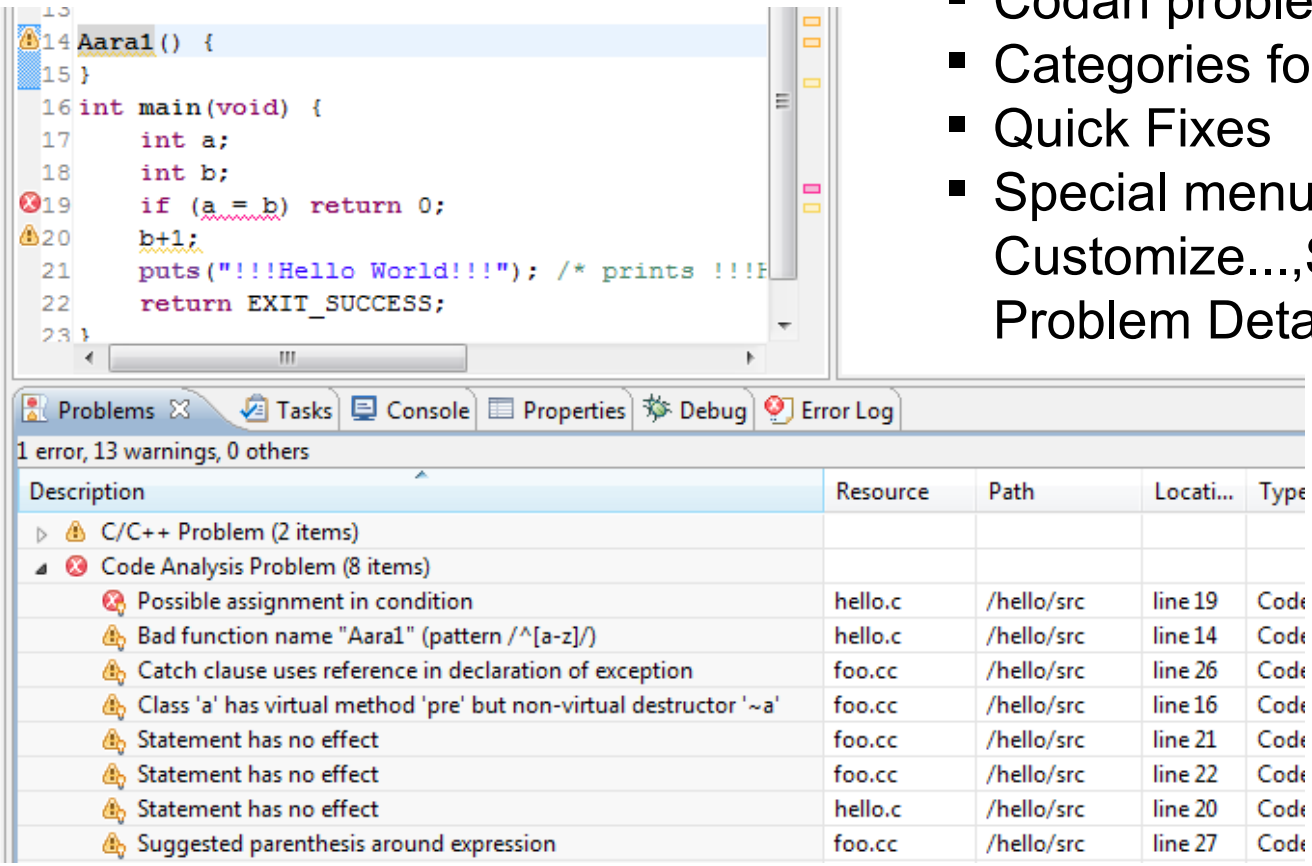

Run with Build




Run as you type



# Problem Markers & Quick Fix




- Codan problem markers
- Categories for grouping
- Quick Fixes
- Special menu commands: Customize...,Show in Problem Details view

# CDT 7.0

- Released in CDT 7.0 as optional feature
- Framework Features
  - Pluggable checkers, base classes for generic checker and C/C++ AST checkers
  - Customizable problems with severity, enablement, categories
  - Parametrized checkers (limited ui support)
  - Problem details view (extendable)
- Only handful of checker available for end-users
-

# Development Status CDT 8.0

- Framework
  - Generic framework for Quick Fix
  - Base classes for simplified junits (code samples are read from comments)
  - Better support for marker generation (tries to update markers instead of delete/insert)
  - Common scope filters for checkers (excluded/included files)
- Checker & Quick Fixes
  - Added Problem Binding checker (which produces dozens of problems) and Quick Fixes such as “Create Local Variable”, etc
  - Added assignment to itself checker

# How to create Internal Checker

- Define a problem
- Pick a model that can be used to find a problem, i.e. Index, AST, Control Flow Graph, Data Flow Graph, Call Graph
- Extend abstract checker that supports a given model, and implement a check (currently supported: No Model, Indexer, C/C++ AST, Control Flow Graph)
- Create extension to define your checker and problem(s) it can find, define a new category or assign to existing one
- Create a quick fix for the problem (optional)
- Create a documentation/description of a problem and integration into extension
- Creation extension to problem view (optional)
- Create a junit test cases

# Internal Checker – Example (Extension)

```
<extension
         point="org.eclipse.cdt.codan.core.checkers">
  </checker>
            <checker
            class="org.eclipse.cdt.codan.internal.checkers.StatementHasNoEffectChecker"
            id="org.eclipse.cdt.codan.internal.checkers.StatementHasNoEffectChecker"
            name="StatementHasNoEffectChecker">
      <problem
            category="org.eclipse.cdt.codan.core.categories.ProgrammingProblems"
            defaultSeverity="Warning"
            id="org.eclipse.cdt.codan.internal.checkers.StatementHasNoEffectProblem"
            name="Statement has no effect">
            messagePattern="Statement has no effect ''{0}''"
      />
  </checker>
</extension>
```

# Internal Checker – Example (Code)

```
public class StatementHasNoEffectChecker extends AbstractIndexAstChecker {
private static final String ER_ID =
"org.eclipse.cdt.codan.internal.checkers.StatementHasNoEffectProblem"; //$NON-NLS-1$

public void processAst(IASTTranslationUnit ast) {
    ast.accept(new CheckStmpVisitor());
}

class CheckStmpVisitor extends ASTVisitor {
    CheckStmpVisitor() {
        shouldVisitStatements = true;
    }

    public int visit(IASTStatement stmt) {
         if (stmt instanceof IASTExpressionStatement) {
                    if (hasNoEffect(((IASTExpressionStatement) stmt).getExpression())) {
                              reportProblem(ER_ID, stmt);
                    }
         }
    return PROCESS_SKIP;
}
…
```

See full code of this checker in codan project:
org.eclipse.cdt/codan/org.eclipse.cdt.codan.checkers/src/org/eclipse/cdt/codan/internal/checkers/StatementHasNoEffectChecker.java

# How to create External Checker

- If tool is integrated with a make base build system already
  - Either a) adjust tool output to match one of the recognized error patterns (such as gcc)
  - b) create an error parser for your tool using an API or UI
- If you want user to control problem profile for the tool
  - Create a “checker” for the tool you running
  - Register problems or group of problems tools creates
  - Add a listener for profile changes to generate external problems profile that can be used by the tool
- If tool is not integrated with make implement a checker as launcher of tool (get projects option such as includes and defines from CDT)
- Extend problems view to show addition problem information/documentation

# What API provided to aid in writing a checker

Usually static analysis checkers use one or more of the following software models:

- File Text – platform/java
- Token list – CDT - scanner and preprocessor
- Comments – CDT - scanner
- Language AST – CDT - parser
- Bindings – provided by CDT - indexer
- Control Flow Graph – CDT - codan
- Data Flow Graph – todo in codan
- Call Graph – CDT - indexer
- Abstract Checker classes – CDT - codan

# Questions?

http://wiki.eclipse.org/CDT/designs/StaticAnalysis