Scanner Configuration Usability Enhancements FDS

This document describes the proposed work items for the scanner configuration usability enhancements for the CDT 2.0 release.

Author : Vladimir Hirsl
Revision Date : 02/11/2004 - Version: 0.2.0
Change History : 0.1.0 - Document Creation
: 0.2.0 - Review update

Table of Contents

1. Introduction

This document provides an overview of the proposed enhancements to Standard Make Project's scanner configuration.

Currently user has to specify scanner configuration (include paths and symbol definitions) manually in the project's properties dialog or in the preferences for C/C++ New Make Projects. The goal of this feature is to automate as much of that process as possible so that user gets better 'out of the box' experience.

1.2 Glossary

This section defines terms commonly used in this document.

Build target	A target in the build sense as defined in makefiles (i.e. clean, all)
Target specific options	Set of compiler options that affect compiler's scanner info defined in a 'specs' file. For GNU C/C++ compilers these are: `mwin32`, `mno-win32`, `mno-cygwin`, `ansi` and `nostdinc`.
Scanner Info	Include paths and symbol definitions used by the compiler to process a compilation unit (C or

	C++ source file).
Scanner configuration	Scanner info collected, stored and managed on a project level (union of all project's compilation units' scanner info).
Discovered SC	Discovered scanner configuration - scanner configuration attained through scanner configuration discovery
Existing discovered SC	Existing discovered scanner configuration - previous value of discovered scanner configuration; initially empty
Project's scanner configuration	Sum of user specified and existing discovered scanner configuration. Persisted in the CDT project file. Accessible through GUI.
Indexer	Provider of information needed for efficient searching, code assist, text hover, refactoring, etc. Relies heavily on the parser/scanner.
Discovery	CDT component responsible for scanner configuration discovery.

2. Standard Make

2.1 Overview

Generally, build configuration consists of several configuration items i.e. preprocessor options, path and symbol information, optimization, debugging and warning options, etc. In a standard make project build configuration is defined in makefiles. The minimal information required by the IDE tool to enable proper indexer functionality and ultimately, features like search, content assist, etc, is path and symbol information. To help enable search features for the standard make scenario, we introduce the notion of **scanner configuration**. A scanner configuration is simply a set of path and symbol information. It is required for proper IDE functionality, but it does not affect standard make build system (which is fully defined by the makefiles).

One of the challenges that immediately arise when trying to extract path/symbol information from a makefile is that the desired information can change depending on the target being built. For that reason, the best way to extract info from the makefile (without actually parsing the makefile) is to parse the build output.

To be able to automate discovery of path and symbol information in a standard make project, scanner configuration and build target need to be associated in some way. As a starting point we suggest a single scanner configuration associated with build targets specified in *Incremental Build* and *Full Build* fields. The idea is to collect scanner configuration information when any of the targets is built. The main issue with a single scanner configuration is that building targets that are not part of *Incremental Build* and *Full Build* target list will not affect scanner configuration. Therefore, scanner configuration may be incomplete.

Ultimately we may want to allow multiple scanner configurations, each associated with possibly multiple build targets (some build targets may even belong to multiple scanner configurations). We would need a clever way of managing scanner configurations and build targets. Also switching among scanner configurations may cause reindexing of the project (time consuming). Obviously, this matter requires further investigation.

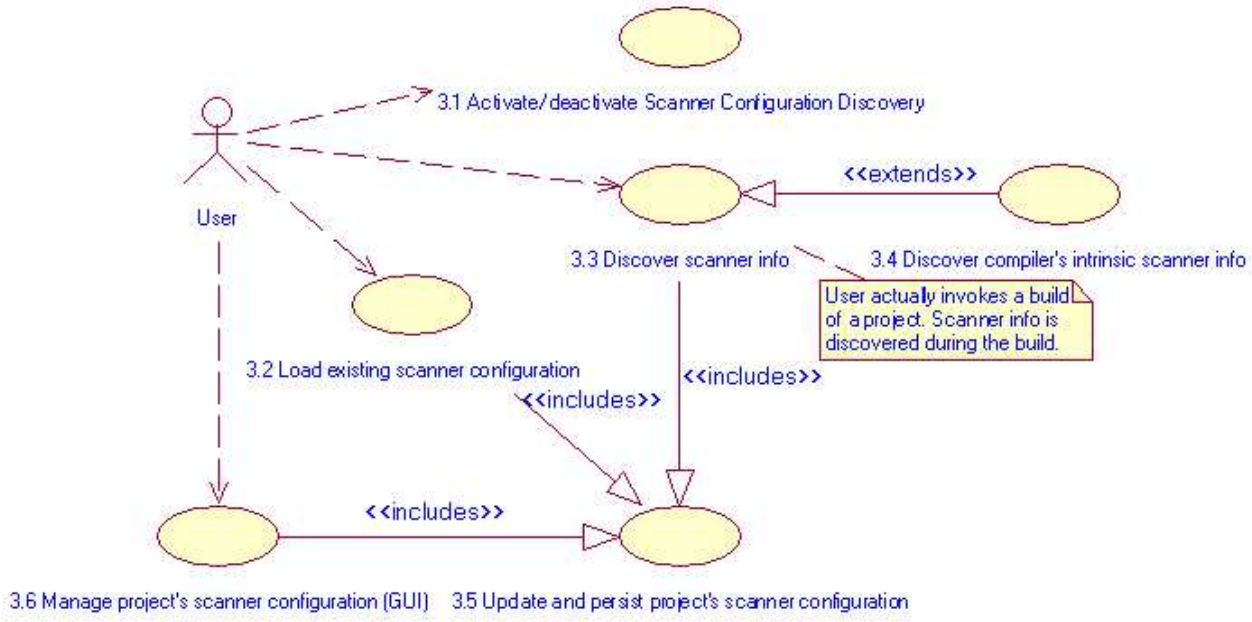
3. Use cases

The use cases assume GNU C/C++ compiler as an example. All compiler specific functionality will be generalized through extension points. The GNU compiler specific functionality will be contributed as extensions to these or already existing extension points.

There are three main use cases a user can exercise:

- activate or deactivate scanner configuration discovery,

- invoke a build of a project that will indirectly start scanner configuration discovery and update and
- manage (add, remove, edit, use variable extended paths,) project's scanner configuration.



3.1 Activate/Deactivate scanner configuration discovery

Brief description

User sets/clears "Automate Scanner Configuration Discovery" checkbox in Scanner configuration discovery page (see 4.2.2), of the:

- Preferences, C/C++, New Make Projects or
- New Standard Make C/C++ project wizard or
- Standard Make project property page.

To enhance user out-of-box experience, this option is going to be enabled by default.

Workflows

Preferences, C/C++, New Make project preference page

In the New Make project preference page user sets "Automate Scanner Configuration Discovery" in the Make Builder tab. All new C and C++ projects will have scanner configuration discovery activated. Existing C/C++ projects will have the option value as specified in the project's property page.

New Standard Make C/C++ project wizard

User creates a new C or C++ Standard Make project invoking New Project wizard. In the Make Builder tab of the New Project wizard user sets "Automate Scanner Configuration Discovery". The state of the option is reflected in the project's property page.

Standard Make project property page

In the C/C++ Make Project property page of the Standard Make C/C++ project user sets "Automate Scanner Configuration Discovery" in the Make Builder tab.

Post-conditions

Setting this option adds a new builder to the standard make projects that enables discovery and update of project's scanner configuration on a next build request.

3.2 Load existing scanner configuration

Brief description

This use case begins when user opens an existing C/C++ project. Project's scanner configuration is populated with the information from the .cdtproject file. The use case ends when the project's existing discovered scanner configuration is about to be updated.

Workflows

Getting information from .cdtproject file

The user opens a Standard Make project in the C/C++ Projects view. Information from the project's .cdtproject file is read and project's existing discovered scanner configuration is populated. The information from the .cdtproject file is contributed to the project's scanner configuration.

- Includes Use case 3.5 Update and persist project's scanner configuration.

3.3 Discover scanner info

Brief description

This use case begins when user requests a build or rebuild of a project. The build output is parsed for include paths and symbol definitions. The information is collected for all resources in the project and after the build is done, discovered scanner info is contributed to the project's *existing discovered scanner configuration*. The use case ends when the project's existing discovered scanner configuration is updated.

Workflows

All different ways of invoking a build on a project lead to the following workflow.

Project build

The user selects a Standard Make project in the C/C++ Projects view. They chose to build the project or project's make target.

Output of the build process is parsed for include paths, symbol definitions and target specific options for each compiled C or C++ file. Discovered scanner info for each compiled source file in a project is collected into the *project's discovered scanner configuration*. When the build of the project is done, following step is performed:

- XP1: Use Case 3.4 Discover compiler's intrinsic scanner info.

Project's *existing discovered scanner configuration* is then updated with the project's *discovered scanner configuration*. Entries in discovered scanner configuration not existent in the existing discovered scanner configuration are added to the existing discovered scanner configuration. Resulting changes are then contributed to the project's scanner configuration.

- Includes Use Case 3.5 Update and persist project’s scanner configuration.

Special requirements

Parsing build output for scanner info is compiler specific. The GNU toolchain compilers gcc and g++ use -I for include paths and -D for symbol definitions while Microsoft Visual C++ uses /I and /D respectively. Also, list of target specific options and scanner information contained in the specs file are compiler specific.

Limitations

Silent make (make –s) does not produce any build output and therefore makes this use case to fail. Note that the goal of this feature is not to solve all posible cases where scanner configuration is not correct but to help populating it with the information available to the compiler. Also, one would think that other than the build of Linux kernel, users would generally want as much information about the build of their projects as possible.

Pre-conditions

Option "Automate Scanner Configuration Discovery" is enabled.

Post-conditions

Project's existing discovered scanner configuration is updated. Changes to the existing discovered scanner configuration are propagated to the project's scanner configuration.

3.4 Discover compiler's intrinsic scanner info

Brief description

This use case starts when there is a change in project's target specific options or if compiler's intrinsic scanner info has not been discovered previously. The use case ends when the discovered compiler's intrinsic scanner info is contributed to the *project's discovered scanner configuration*.

The C/C++ compiler is invoked to generate 'specs' output. The output is parsed for include paths and symbol definitions.

Workflows

Parsing output of 'generate specs' command

The C/C++ compiler is invoked to generate 'specs' output based on all discovered target specific options. The output is parsed for include paths and symbol definitions. The discovered compiler's intrinsic scanner info is contributed to the project's discovered scanner configuration.

In case of an error in execution of generate specs command no scanner info is contributed to the project's discovered scanner configuration. An error class problem marker is generated in the Problems view with a Quick fix contribution to open a 'generate specs' command setup dialog.

Special requirements

For taking the successful path in this use case, it is necessary that the 'generate specs' command is in system's path and that it is the same command used for compiling source files by the Standard Make project build system.

For GNU C/C++ compilers 'generate specs' command is '**gcc -c -v *target_specific_options* {empty.c | empty.cpp}**'. For other C/C++ compiler there may be a different 'generate specs' command or a different way to retrieve compiler's intrinsic scanner info.

Pre-conditions

The 'generate specs' command is set up by default to '**gcc -c -v**'.

Post-conditions

Discovered compiler's intrinsic scanner info is contributed to the project's discovered scanner configuration.

3.5 Update and persist project's scanner configuration

Brief description

This use case starts when there is a change to the project's scanner configuration. A change in project's scanner configuration is represented with three sets of information: added entries, changed entries and removed entries. The use case ends when the project's scanner configuration is updated and persisted.

Project's scanner configuration consists of two parts: user specified scanner configuration and discovered scanner configuration. User can update both parts through GUI. The "Discovery" can update only discovered part of scanner configuration.

Workflows

Change in discovered portion of scanner configuration

Entries in the set of added entries are added to the project's scanner configuration. Entries in the set of removed entries are have their annotation changed to ‘Deleted’. Entries in the set of changed entries have their annotation changed. An entry in discovered portion of scanner configuration can have following annotations:

- Removed - user disabled the discovered entry (the entry is moved to the Removed list in GUI).
- Deleted - user removed the discovered entry permanently (the entry is removed from both GUI lists)
- Variable extended (for include paths only) - path variable is used to replace a portion of an include path.

Project's scanner configuration is persisted to the CDT project file.

Change in user specified portion of scanner configuration

Entries in the set of added entries are added to the project's scanner configuration. Entries in the set of removed entries are removed from the project's scanner configuration. Entries in the set of changed entries have their value or/and annotation changed. An entry in user specified portion of scanner configuration can have following annotations:

- Variable extended (for include paths only) - path variable is used to replace a portion of an include path.

Project's scanner configuration is persisted to the CDT project file (.cdtproject).

Post-conditions

The project's scanner configuration has been updated and persisted. The next time user selects project properties, Symbols and Paths tab, they will see updated scanner configuration information.

3.6 Manage project's scanner configuration

Brief description

This use case starts when user opens Paths and Symbols tab from one of the following:

- Preferences, C/C++, New Make Projects preference page or
- New Standard Make C/C++ project wizard or
- Standard Make project property page

The use case ends when the user presses Restore Defaults (the changes are discarded and the change set is empty) or Apply (the changes are in the change set). All the scenarios in this use case include Use Case 3.4 as the last step.

Workflows

Include paths management

User selects to manage include paths. A new "Manage include paths" dialog is displayed (see 4.2.3). User can:

- add new include paths,
- remove existing include paths,
- edit existing include paths,
- remove discovered include paths,
- restore removed discovered include paths.

The dialog is closed when the user presses Ok (the changes are applied and the Paths and Symbols tab reflects the changes) or Cancel (the changes are discarded and the Paths and Symbols tab remains unchanged).

User presses Restore Defaults or Apply to apply the change set to the project's scanner configuration. Includes Use Case 3.5 Update and persist project’s scanner configuration.

Using Variable extended paths

User chooses to apply variable extended paths to the entry in the list of include paths. A variable extended paths dialog is displayed. User can select from the list of existing variable extended paths or define a new variable extended path. Then user selects an include path prefix that is to be replaced with variable extended path. The variable extended paths display is closed and the include path entry is updated.

User presses Restore Defaults or Apply to apply the change set to the project's scanner configuration. Includes Use Case 3.5 Update and persist project’s scanner configuration.

Moving entries in the include paths list

User selects an entry in the include paths list and presses Up or Down button. An entry is moved accordingly in the list of include paths.

User presses Restore Defaults or Apply to apply the change set to the project's scanner configuration. Includes Use Case 3.5 Update and persist project’s scanner configuration.

Defined symbols management

User selects to manage defined symbols. A new "Manage defined symbols" dialog is displayed (see 4.2.4). User can:

- add new symbol definitions,
- remove existing symbol definitions,
- edit existing symbol definitions,
- remove discovered symbol definitions,
- restore removed discovered symbol definitions

The dialog is closed when the user presses Ok (the changes are applied and the Paths and Symbols tab reflects the changes) or Cancel (the changes are discarded and the Paths and Symbols tab remains unchanged).

User presses Restore Defaults or Apply to apply the change set to the project's scanner configuration. Includes Use Case 3.5 Update and persist project's scanner configuration.

Alternative flow - Moving entries in the defined symbols list

User selects an entry in the defined symbols list and presses Up or Down button. An entry is moved accordingly in the list of defined symbols.

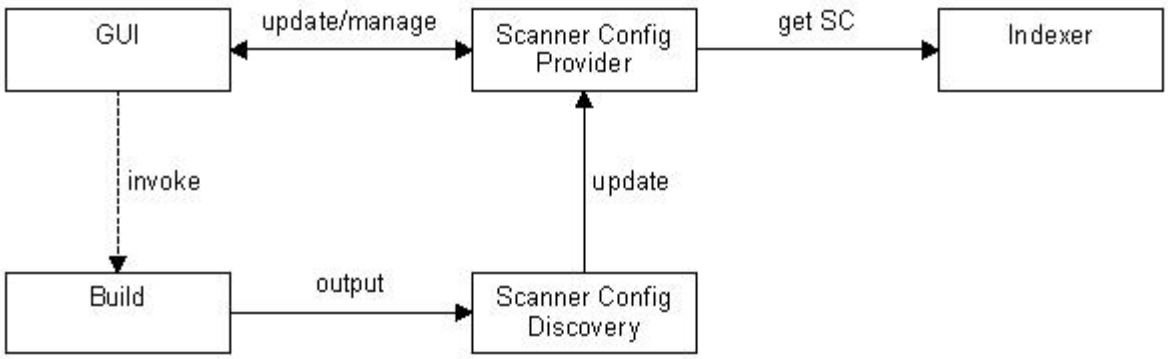
User presses Restore Defaults or Apply to apply the change set to the project's scanner configuration. Includes Use Case 3.5 Update and persist project's scanner configuration.

4. Design considerations

4.1 Architectural overview

Mechanism of scanner configuration discovery ("Discovery") resides in `org.eclipse.cdt.make.core` and `org.eclipse.cdt.make.ui` plugins.

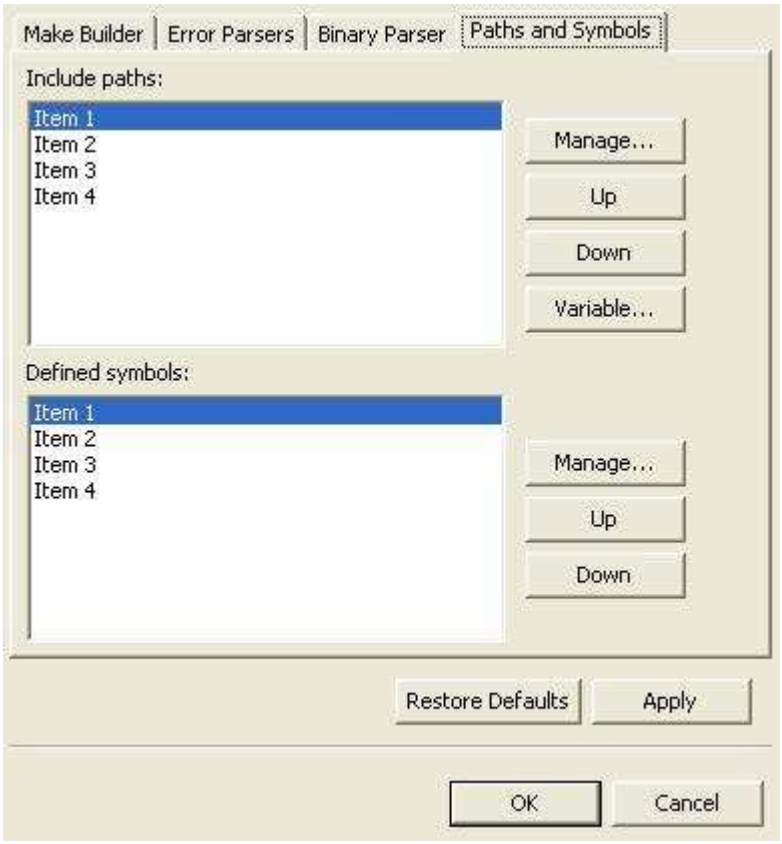
This is the block diagram of "Discovery" and its clients and providers:



4.2 GUI changes

This section contains some mock-ups of the UI elements that may be required for the new or enhanced features in the Standard Make build system. These do not represent final design decisions, but an attempt to elicit some feedback on how they could be improved.

4.2.1 Changes to Paths and Symbols tab

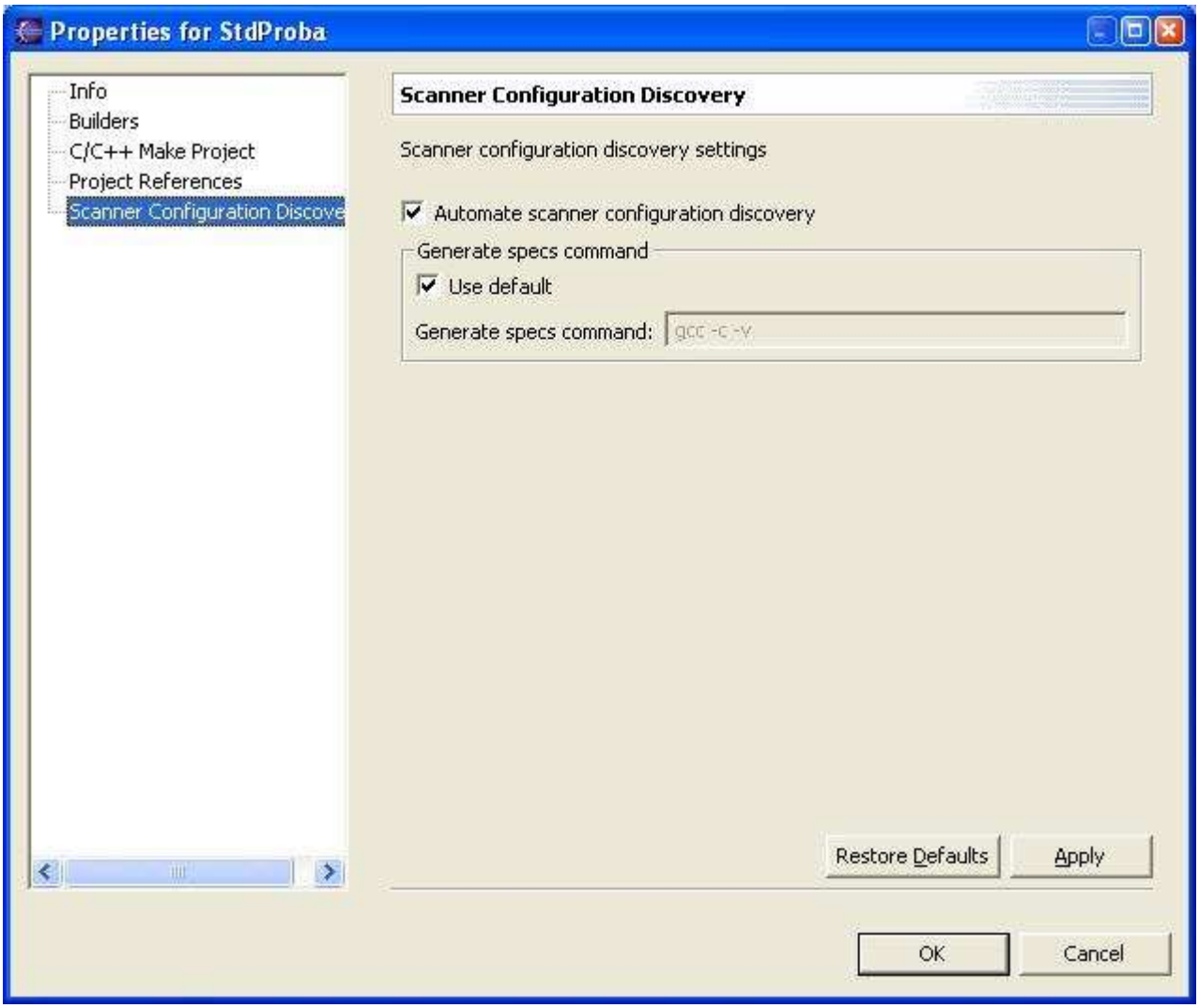


The Paths and Symbols tab has four new buttons:

- Manage... - for displaying new Manage include paths dialog,
- Variable... - for defining new and assigning variable path extensions to a selected include paths entry,
- Manage... - for displaying new Manage defined symbols dialog,

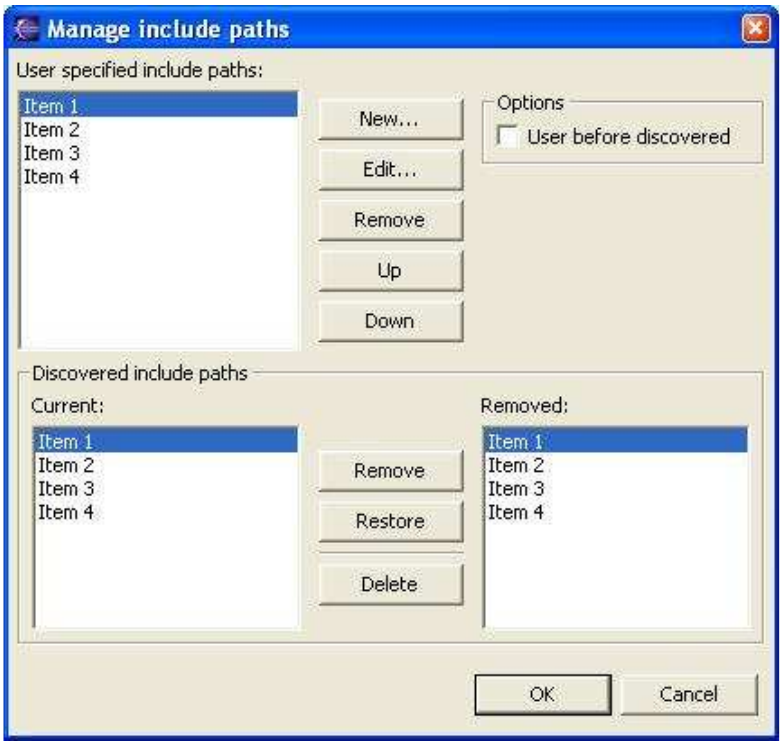
4.2.2 Scanner configuration discovery page

A new preference/property/wizard page to specify scanner configuration discovery options.



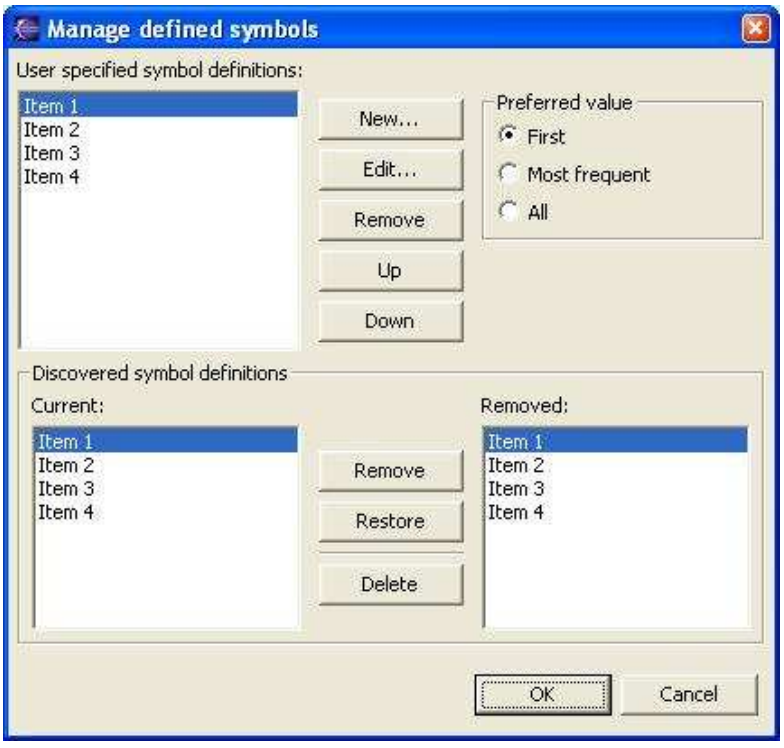
4.2.3 New Manage include paths dialog

A new dialog to manage user specified and discovered include paths.



4.2.4 New Manage defined symbols dialog

A new dialog to manage user specified and discovered symbol definitions.



4.2.5 Variable path definitions dialog

Current intention is to reuse Eclipse's `PathVariableSelectionDialog` dialog. In case it does not provide desirable functionality, a custom one will be implemented. The minimum the dialog needs to provide is functionality to create new path variables, select an existing path variable and replace a part of a discovered include path with a path variable.

4.3 API changes

4.3.1 Persisting scanner configuration

Currently there is an API to add path and symbol information from GUI to build settings (project properties). Some API changes may be required to enable contribution of discovered scanner configuration settings to project properties. The intention is to use new `ICPathEntry` mechanism when it becomes available. The format of `.cdtproject` file is most likely to change. Following additions are anticipated:

1. Versioning information (backward compatibility with conversion to a new format).
2. Scanner configuration information (with a list of associated build targets).

An example project file (additions in red):

```
<?xml version="1.0" encoding="UTF-8"?>
<cdtproject id="org.eclipse.cdt.make.core.make" version="1.0">
    <extension id="org.eclipse.cdt.make.core.MakeScannerProvider"
               point="org.eclipse.cdt.core.ScannerInfoProvider"/>
    <extension id="org.eclipse.cdt.core.Elf"
               point="org.eclipse.cdt.core.BinaryParser"/>
    <data>
        <item id="org.eclipse.cdt.make.core.makeScannerInfo">
            <scannerConfiguration name="Default" active="true">
                <buildTarget target="lib1"/>
                <buildTarget target="app"/>
                <buildTarget target="all"/>

                <includePath path="/usr/st60/common60/include"
                             discovered="true"/>
                <includePath path="/usr/st60/ot60/include"
                             discovered="true"/>
                <definedSymbol symbol="DEBUG"
                               discovered="true"/>
                <definedSymbol symbol="LOG_LEVEL=3"
                               discovered="true"/>
                <definedSymbol symbol="LOG_LEVEL=1"
                               discovered="true"
                               removed="true"/>
            </scannerConfiguration>
        </item>
    </data>
</cdtproject>
```

4.3.2 Processing of discovered scanner configuration after the make build

Scanner configuration is discovered on a per file basis and contributed to the project's scanner configuration. When the build is done (and consequently the "Discovery") the discovered scanner configuration needs to be contributed to the existing scanner contribution. However, currently there is no mechanism in place to generate a notification that the build is done. A new mechanism needs to be added to generate build done notifications so

that “Discovery” can take appropriate actions. Alternatively, these actions will be implemented as a new builder that runs after the make builder.

4.4 Extension points

1. `SpecsBuilder` extension point – to provide a way to generate compiler’s specs file output. The output is parsed by the `ErrorParser`. Default implementation for GNU C/C++ compilers.

4.5 Extensions

1. `GCCScannerInfoParser` extension implements `org.eclipse.cdt.core.ErrorParser` extension point. Parses build output for GNU C/C++ compiler include paths and symbol definitions (see 6.1.1).
2. `GCCCompilerSpecsParser` extension `org.eclipse.cdt.core.ErrorParser` extension point. Parses specs output for GNU C/C++ compiler’s intrinsic include paths and symbol definitions (see 6.1.2).

4.6 User documentation

The following points need documentation:

1. Importing a Standard Make project - how to set up a scanner configuration/build target
2. How the automated scanner configuration discovery works

4.7 Unresolved issues

- Include paths as a part of scanner configuration need to be in one of two forms: absolute or variable extended. Relative parts need to be transformed into one of the two forms. Also, compiler’s intrinsic include paths are relative to the environment (absolute paths in case of Linux, relative to the instalation directory in case of Cygwin)
- What to do in case of indexer error or if file or symbol cannot be found? (see *Indexer Features for CDT 2.0* FDS).

  It is assumed that Indexer will generate Problem markers in case a file cannot be found in the include path or a symbol definition cannot be found. In that case the “Discovery” will provide contributions to the Quick Fix dialog to either:

  1. Ask user to start scanner configuration discovery by rebuilding the project, or
  2. Display Manage include paths or Manage defined symbols dialog so that user can enter missing information.

- How does `ICPathEntry` fit into this story?

5. Additional proposed enhancements

5.1 Multiple indexer configurations

Changing an active scanner configuration of a project will cause reindexing of the project. In case of large projects this can be a time consuming operation. Therefore, changing active scanner configuration can be a costly action. It is worth considering storing indexer files per scanner configuration. This way, changing configurations will only mean changing sets of indexer files. (see *Indexer Features for CDT 2.0* FDS)

6. Appendix

6.1 GNU Compiler Output Examples

6.1.1 Example - compiler output

```
make clean all
rm -rf     ClassA.o m.o Proba.exe
g++ -DDEBUG -DLOG_LEVEL=3 -I/usr/st60/common60/include -I/usr/st60/ot60/include -
O3 -gstabs -Wall -c -o ClassA.o ../ClassA.cpp
g++ -DDEBUG -DLOG_LEVEL=3 -I/usr/st60/common60/include -I/usr/st60/ot60/include -
O3 -gstabs -Wall -c -o m.o ../m.cpp
g++  -o Proba.exe     ClassA.o m.o
Build complete for project Proba
```

6.1.2 Example – compiler's intrinsic include paths and symbol definitions

```
$ gcc -c -v specs.cpp
Reading specs from /usr/lib/gcc-lib/i686-pc-cygwin/3.3.1/specs
Configured with: /GCC/gcc-3.3.1-3/configure --with-gcc --with-gnu-ld --with-gnu-as
--prefix=/usr --exec-prefix=/usr --sysconfdir=/etc --libdir=/usr/lib --
libexecdir=/usr/sbin --mandir=/usr/share/man --infodir=/usr/share/info --enable-
languages=c,ada,c++,f77,pascal,java,objc --enable-libgcj --enable-threads=posix --
with-system-zlib --enable-nls --without-included-gettext --enable-interpreter --
enable-sjlj-exceptions --disable-version-specific-runtime-libs --enable-shared --
disable-win32-registry --enable-java-gc=boehm --disable-hash-synchronization --
verbose --target=i686-pc-cygwin --host=i686-pc-cygwin --build=i686-pc-cygwin
Thread model: posix
gcc version 3.3.1 (cygming special)
 /usr/lib/gcc-lib/i686-pc-cygwin/3.3.1/cc1plus.exe -quiet -v -D__GNUC__=3 -
D__GNUC_MINOR__=3 -D__GNUC_PATCHLEVEL__=1 -D__CYGWIN32__ -D__CYGWIN__ -Dunix -
D__unix__ -D__unix -idirafter /usr/lib/gcc-lib/i686-pc-
cygwin/3.3.1/../../../../include/w32api -idirafter /usr/lib/gcc-lib/i686-pc-
cygwin/3.3.1/../../../../i686-pc-cygwin/lib/../../include/w32api specs.cpp -
D__GNUG__=3 -quiet -dumpbase specs.cpp -auxbase specs -version -o
/cygdrive/c/DOCUME~1/vhirsl/LOCALS~1/Temp/ccRQjaNO.s
GNU C++ version 3.3.1 (cygming special) (i686-pc-cygwin)
        compiled by GNU C version 3.3.1 (cygming special).
GGC heuristics: --param ggc-min-expand=100 --param ggc-min-heapsize=131072
ignoring nonexistent directory "/usr/local/include"
ignoring nonexistent directory "/usr/i686-pc-cygwin/include"
ignoring duplicate directory "/usr/i686-pc-cygwin/lib/../../include/w32api"
#include "..." search starts here:
#include <...> search starts here:
 /usr/include/c++/3.3.1
 /usr/include/c++/3.3.1/i686-pc-cygwin
 /usr/include/c++/3.3.1/backward
 /usr/lib/gcc-lib/i686-pc-cygwin/3.3.1/include
 /usr/include
 /usr/include/w32api
End of search list.
 /usr/lib/gcc-lib/i686-pc-cygwin/3.3.1/../../../../i686-pc-cygwin/bin/as.exe --
traditional-format -o specs.o /cygdrive/c/DOCUME~1/vhirsl/LOCALS~1/Temp/ccRQjaNO.s
```

7. References

CDT Code - Managed Make, Standard Make
CDT Core (ICPathEntry)
Scanner Configuration Usability Enhancements SRS

Last Modified on Tuesday, February 11, 2004