Search FDS for CDT 2.0

Author: Bogdan Gheorghe
Version: 1.0
Date: 26/01/04

1 Introduction

Search first made its debut in the CDT 1.2 release. The feature allows users to conduct semantic searches in their project files, using a dialog to specify search parameters. The search results are displayed in the Search console pane and each result acts as a position marker on a file. Double clicking on a search result will open the corresponding file to the position marker.

For the 2.0 release, the main focus of the Search feature is to improve the overall user experience.

This document will present the main use cases for each of the requirements listed for Search.

2 Requirements

The following table summarizes the Search requirements for CDT 2.0:

ID	Requirement	Priority	Committed
R1	Search will make all attempts to work "out of the box" - i.e. all the include paths that can be extracted are extracted	P1	
R2	Search can be invoked in the editor from the context menu	P1	
R3	Search will be able to search and place markers on external documents	P2	
R5	Move search to background	P1	
R6	Add Project as one of the immediate scopes available	P2	
R7	Search will work for C Projects	P1	
R8	Search will be I18N compliant	P1	
R9	Search will follow accessibility guidelines	P1	

3 UI Changes

Most of the changes to Search for CDT 2.0 will be behind the scenes. The only UI changes are as follows:

- A new Project radio button will be added to the Search dialog to allow the selection of project scope

- Context menu will be altered to accommodate selection search. Specifically, the menu will look like this (where the last menu is the same for all 3 search types) :

Search> Declarations > Workspace
Definitions Project
References Working Set …

4 Use Cases

4.1 Search will work out of the box

4.1.1 Description

The initial version of search did not perform any configuring of the user's environment. The user was expected to enter in all of the required include path/symbol information needed by the Indexer to successfully index files. Based on user feedback, it can be deduced that most users were not made aware of this prerequisite to getting search up and running. To make matters worse, there was no indication available to the user that the parser had run into trouble with a certain file. In fact, most users just gave up on search after seeing that it wasn't working right away.

Thus, the overall most important goal for Search is to provide a positive first experience by automating the setup process as much as possible for the user. There are 3 separate parts to this:

1. Improving the Managed Make project setup experience
2. Improving the Standard Make project setup experience
3. Providing visual feedback when things go wrong

4.1.2 Managed Make Setup

As Managed Make requires the user to enter whatever paths are necessary to build his project, the only Search-related problems that can be encountered in Managed Make projects are with a compiler's internal search paths.

Consult the *Managed Make FDS* to see how Managed Make proposes to handle compiler built-ins.

4.1.3 Standard Make Setup

The Standard Make scenario is tougher to automate in terms of include path/symbol setup as all of the information must somehow be obtained from existing makefile(s).

For details about the Standard Make setup, consult the *Scanner Configuration Usability Enhancement FDS*.

4.1.4 Error Tasks

One of the complaints of users trying to use search in CDT 1.2 was that there was no indication of indexing problems (apart from an obscure log in the .metadata folder that most users were not aware of) – so when search ran into problems, users had no idea how to fix it.

For CDT 2.0, the Indexer will create error markers for all reported parser errors. The user will be able to click on the individual error markers and be presented with a number of resolution possibilities. For details about index errors tasks consult the *Indexer Enhancements for CDT 2.0 FDS*.

4.2 Selection Search

4.2.1 Description

CDT 2.0 will provide a selection search facility. The user will be able to select an element in his editor, select the search type and search scope from a context menu and search will do the rest (namely, it will use the parser to figure out what element is selected and perform a search).

The search types and search scopes are the same that are presented in the search dialog.

Search Types

1. Declarations
2. Definitions
3. References

Search Scopes

1. Workspace
2. Project
3. Working Set

4.2.2 Use Case

1. The user will select an element in the editor that he wishes to perform a search with.
2. The user will select a search type from the context menu.
3. The user will select a search scope from the search type's submenu.
4. Search will populate the result window with matches.

4.2.3 Limitations

- Obviously, the element selected will have to be a valid element in order for search to work. If the search being undertaken works from the dialog, it should work as a selection search.
- The file must be indexed in order for search to work – if the element selected is not indexed for whatever reason (parser error, not seen in current configuration

etc.) then search will not return any results. The user will be informed by the indexer of any correctable problems via error tasks.

- The allowable search elements are the exact same ones as the ones in the dialog:
    - Class
    - Structs
    - Functions
    - Variables (not local)
    - Unions
    - Methods
    - Fields
    - Enumerations
    - Enumerators
    - Namespaces

4.2.4 Time Estimate

- Hooking search up to the new parser SELECTION_PARSE mode + framework for selection search – 2 days

- New UI (Actions + Action groups) – 2 days

4.3 Search on External Files

4.3.1 Description

Currently Search does not allow the user to perform searches on external files that are included by source files inside the workspace.

In CDT 2.0 searching for elements in external files will follow the exact same process as regular searches. The user can initiate a search from the dialog or through the selection search mechanism, and the results will be displayed as markers in the search view. These markers, when clicked on, will open the external file in a new editor.

4.3.2 Use Case

1. The user selects an external element in the editor, and selects a search type and a search scope.
2. Search displays the results in the search results window as markers.
3. Clicking on one of the markers will open the external file in a new editor.
4. The file opened will have a marker on the line of the found element.

4.3.3 Limitations

External files must be referenced by a source file in order to be included in the search. This is going to be a fundamental limitation when working with C files that are not included by any file and are external to the workspace. A possible solution is to index all files that are found in include directories whether they are included or not.

4.3.4 Time Estimate

- Modify BasicSearchResultCollector to handle non-local resource match creation – 1 day
- Modify CSearchResultCollector to create a non-local resource marker – 1 day
- Modify GotoMarkerAction to open an editor on a non-local resource and add a marker to the appropriate line – 2 days

4.4 Search moved to Background

4.4.1 Description

Eclipse 3 has introduced the notion of moving long running tasks to background threads. This allows the user to continue working while conducting time consuming operations. Search is a perfect candidate for background processing.

4.4.2 Use Case

1. User will initiate a search (either through the dialog or through the selection search mechanism).
2. Upon starting the search the user will be allowed to continue working as before in the IDE.
3. The search results will appear as they are found in the results pane.
4. If the indexer is currently in the middle of indexing a project, it will pause and search will execute on whatever is indexed at that point. For more details see *Indexer Enhancements for CDT 2.0 FDS*

4.4.3 Time Estimate

Rewrite search mechanism to make use of new Job Manager - 4 days

4.5 Search Project Scope

4.5.1 Description

Search currently has three types of search scopes:

- Workspace : all of the projects in the workspace are included in the search
- Working Set : the user can create a resource based working set
- Selected Resource : whatever resource is currently selected in the navigator

Search in 2.0 will add the Project scope. The Project scope encloses all the elements in a project **including** any resources included by project resources.

4.5.2 Use Case

1. User will choose Project scope from either the context menu or the dialog.
2. The search results will include all elements in the project plus any elements referenced by files in the project.

4.5.3 Limitations

Project inclusions shall come only from files included by source files in the project being searched. Project References will not be taken into account as they really are a build concept that brings around a whole other slew of problems (do we take into account includes, symbols definitions of the referenced project? how do we handle conflicting symbols?)

4.5.4 Time Estimate

2 days to create a project scope and update the UI

4.6 Search will work for C Projects

4.6.1 Description

Search will allow users to search for C language elements that are not included by a source file. This is actually a requirement on the indexer – but the result is that files that reside in the workspace and are not included specifically by a file will still be indexed and, thus, included in searches.

In this case it is possible that search will return inexact matches. In other words, search might not be able to resolve all references with 100% certainty. For more details see *Indexer Search Enhancements for CDT 2.0*.

4.6.2 Use Case

1. User initiates a search for an element in a file that has not been included by any other source files.
2. Search returns the results as best it can (specifically for reference searches).

4.6.3 Limitations

As mentioned before, only C files in the workspace are going to be searched.

4.7 I18N / Accessibility

4.7.1 Description

Search will be as I18N compliant as it needs to be. Both methods of search input – dialog box and editor – are currently capable of handling Unicode characters. But this is meaningless without parser backup – since the parser populates the index. The I18N requirements for the parser are still being determined and search's I18N requirements will flow directly from those.

The one possible I18N requirement on search is sort order by locale. One possibility to tackling this problem is to get an instance of a Collator object for the current locale. The Collator would then handle locale-sensitive string comparison.

For Accessibility the main influences on search will be:

- All search menu items are accessible through keyboard shortcuts
- All search UI items can be navigated by using the TAB and arrow keys (this includes the search dialog and the search result window)
- Search result icons – these are the same as the Outline view icons. Any changes made to them to address accessibility needs will apply to search.