Software Requirement Specification

C/C++ Parser for CDT 2.0

Table of Contents

Revision History

Revision	*Author*	*Date*	*Description*
0.1	John Camelon	11/28/03	Initial Draft
0.2	John Camelon	12/08/03	Incorporated Michel's comments
0.3	John Camelon	01/05/03	Added traceability requirement.
0.4	John Camelon	01/05/03	Added error handling requirement.
0.5	John Camelon	01/09/03	Revisions from review meeting. Update document template.

Introduction

This document serves as the repository for requirements related to the CDT C and C++ parser for 2004's release of CDT (2.0). These requirements are categorized by the features the end-user-features that they provide support for; where more than one CDT feature benefits from the enhancement, the requirement is considered to be a general requirement upon the parser.

The intent of this document is that design documentation shall be provided in the near future detailing exactly how these requirements have been met. It is not the intent to capture requirements of features beyond the 2.0 time frame. Uncommitted C/C++-oriented features include:

- documentation generation support (doxygen, etc.)
- task tags w/problem reconciliation
- quick fix support
- refactoring

Parser clients in the CDT 2.0 time frame include:

- C-Model
- Structure Comparator
- Indexer
- Search
- Code Assist

The requirements specified in this document are given priorities of P1 through P3, where P1 indicates a “must have”.

General Requirements

The following requirements must be met:

1. We must fully support the C++ language as accepted by the latest version of the GNU compiler (gcc). (At this time, this is v.3.3.2 and the detailed manual can be found @ http://gcc.gnu.org/onlinedocs/gcc-3.3.2/gcc/). The specific details as to how that support is provided (syntactically,semantically,or in the AST) will be described in the FDS for this work. **(P1)**
2. Public interfaces need to be documented and published in accordance with our requirement to enable ISVs. **(P2)**

3. The AST interface needs to be made more robust so that navigation in-and-among different AST nodes can be done with minimal effort by parser clients. All constructs in C/C++ must be properly represented in the AST w/full navigability. e.g. : Currently there is no way to navigate to the class specifier that represents the type of a field. **(P3)**
4. Complete traceability for parse errors is required (via Eclipse debug tracing and a log file), independent of the client that is using the parser, in order to try and facilitate debugging problems as they come. **(P1)**
5. Error handling must be improved to ensure that we can get "as good results as possible". We need to accept that most likely, there will still be problems in the parser post 2.0, and we need to make sure that the features that depend upon the parser get as much mileage as they possibly can, irregardless of which language features we successfully complete. **(P1)**
6. In the user documentation, it shall be necessary to fully express which features of C/C++/GNU C++ we do not support. **(P1)**
7. The scanner must support UTF-8 input, as GCC does. **(P2)**

Search Support

1. The parser needs to semantically understand the context within a file between 2 offsets in a source file in order to provide a context for Search to run upon. This is necessary to provide selection search from the editor. **(P1)**
2. This process should not take longer than ~5 seconds, as there is no way for the user to cancel it. **(P2)**

Content Assist Support

In order to support code assist, the following requirements must be met:

1. Since code assist may call the parser several times a minute, it is imperative that we model the performance of the overall operation and look into short and long term solutions for trying to cache and optimize the results. **(P1)**
2. The parser needs to semantically understand the context within the file at a specified offset in order to provide code/content assist with the starting point by which it can query for particular symbols within the parser. **(P1)**
3. The parser needs to provide a lookup mechanism off of the AST so that content assist can query the entire AST/symbol table in order to provide a result-set of completion proposals. **(P1)**

4. While the parser does not currently support templates, there is mounting pressure to allow for code assist on template classes. We should consider providing some partial functionality in this area. **(P2)**

Globalization Support

1. The parser must support Unicode escape sequences within identifiers, string literals and character literals as specified in the ANSI & ISO language specifications. **(P1)**