dcg_build_system

Software Requirements Specification

Document

Version: 1 **Date: 10/09/2016**

Table of Contents

1. Introduction

1.1 Purpose

The purpose of this document is to describe the purpose and features of the software. It will describe its dependencies and installation instructions. It is intended for other software developers and for users.

1.2 Scope

dcg_build_system is a collection of python scripts that build a c++ project.
It supports multiple “build configurations”, each of which can have multiple “modules”

“Precompiled headers” can be specified for individual source files inside a module.

Source files can be specified individually, and they can be found by specifying directories to search or search recursively. Include directories can be individually specified, and they can be found by specifying directories to search recursively.

learning an entire language like “CMake” should be necessary to build c++ projects. Makefiles are flexible and good for creating a project quickly, but they’re difficult to make cross platform, tedious to modify, and they don't have intuitive syntax.

dcg_build_system is installed for a project by copying a folder into the project, and modifying a file in that folder named “config.json” to configure the project.

1.3 Definitions, Acronyms, and Abbreviations.

precompiled header:
A header file that is compiled and used in the compilation of one or more source files. It can dramatically reduce the time it takes to compile a source file if used properly. Each source file can only have up to one precompiled header.

Configuration file:
For the rest of this document, the “configuration file” will be used to refer to a file named “config.json”, which is used to contain the project setup.

configuration:
Settings used to create an executable. Contains modules.

module:
Each module is a group of source files that share some compilation settings, which is useful for large projects which are broken into multiple libraries. It is also an alternative to recursively building project, see the article named “recursive make considered harmful”.

stdin:
The standard input operating system file.

stdout:
The standard output operating system file.

stderr:
The standard error operating system file.

1.4 References

Recursive Make Considered Harmful
10 March 2008
AUUGN Journal of AUUG Inc
http://aegis.sourceforge.net/auug97.pdf

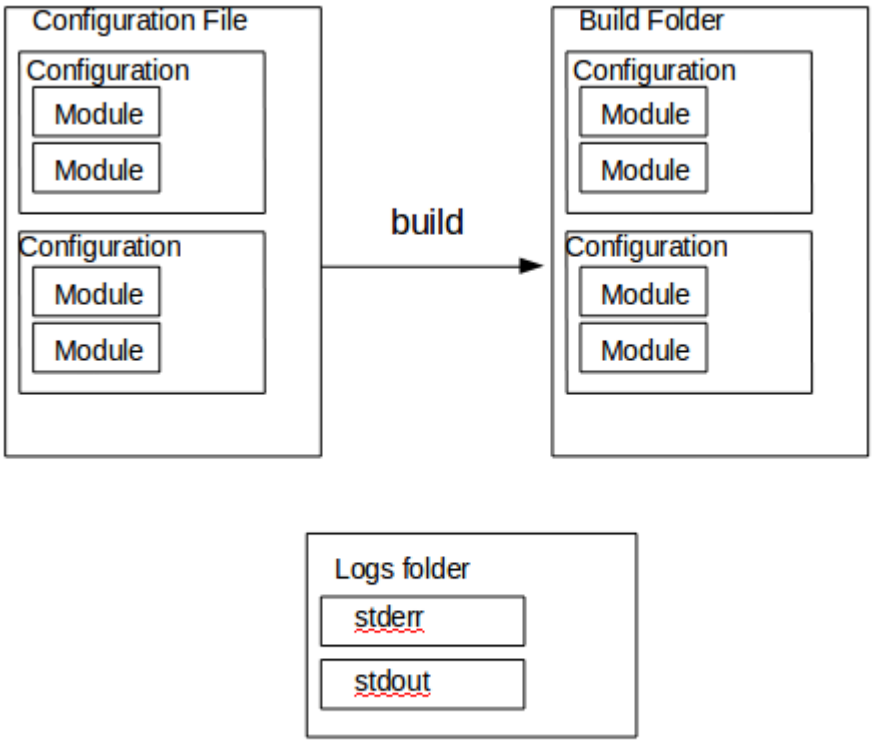
The JavaScript Object Notation (JSON) Data Interchange Format
March 2014
Internet Engineering Task Force (IETF)
https://tools.ietf.org/html/rfc7159

1.5 Overview

Section 2 of the SRS presents the overall description of the software. And section 3 goes into detail about the requirements for the software.

2. The Overall Description

2.1 Product Perspective



2.1.1 System Interfaces

For all path names in the configuration file, use Unix-like path names. The paths must actually exist, otherwise the program will crash and the build will fail. The program will execute commands using spaces as the delimiter between arguments.

2.1.2 Interfaces

The software is used by running "scripts/buildSys.py". Configure the project by modifying the configuration file using any text editor.

2.1.3 Hardware Interfaces

Omitted because the software has no significant hardware requirements. Any modern computer should be able to run the software just fine.

2.1.4 Software Interfaces

GCC (Gnu Compiler Collection):

Used to compile and link. Version 4.8.4 is guaranteed to work. Other versions should work, but have not been tested.

2.1.5 Communications Interfaces

omitted because no communication interfaces are used.

2.1.6 Memory Constraints

Omitted because its use of primary and secondary memory is very small.

2.1.7 Operations

- create or remove configurations or modules
- configure linker settings for a specific configuration
- configure compilation settings for a specific module

2.1.8 Site Adaptation Requirements

A file system is required. And all folders and files specified in the configuration file must exist.

The compiler and linker program are specified in the configuration file. If the program specified is not in the system path then the operating system won't be able to execute the command.

2.2 Product Functions

- clean or build:
  - everything: no other instance of the software should be run while this operation is taking place
  - a specific configuration: no other instance of the software should be building or cleaning the any part of the same configuration at the same time
  - a specific module in a specific configuration: no other instance of the software should be building or cleaning that module in that configuration at the same time

2.3 User Characteristics

Users should understand some JSON. The only JSON types used are string, object, and list. An understanding of compiling and linking may be helpful. Understanding gcc compiler and linker flags is useful, but not necessary. Users should probably know how to make use of the stderr output of the software.

2.4 Constraints

- compiler/linker flags and other values in the configuration file are used directly in the build commands. Therefore they should not contain any command line escape characters, such characters could result in undefined behavior, therefore only trusted users should be allowed to modify the file.
- All paths specified in the configuration file should be Unix-like (using forward slashes).
- If the file system allows the forward-slash '/' character in file and folder names, and some of the folders searched contain items with forward-slashes in their name, there might be problems.

2.5 Assumptions and Dependencies

This section is omitted because there aren't any particular assumptions or dependencies.

2.6 Apportioning of Requirements.

Future development may allow using the “clang” and “cl” to compile and link. So the requirements would change to requiring “clang”, “cl”, or “gcc” compilers instead of only “gdb”.

3. Specific Requirements

3.1 External Interfaces

- Python Interpreter, runs the scripts, input and output are through stdin and stdout, must version 3.
- GCC, can be used to compile and link the project.

3.2 Functions

- build: the system shall print the individual duration in units of seconds that the program spent compiling source files, compiling header files, linking, and generating dependency files.
  - everything: the system shall build all configurations
  - a specific configuration: the system shall build all of its modules and link all the object files from each module into a single executable file. The user will be able to set the target name, linker program, linker flags, library directories, and libraries in the configuration file.
  - a specific module in a specific configuration: the system shall create one compiled header file for each header specified in the configuration file. The system shall create one object file for each source file using a compiled header if one was specified for that source file in the configuration file. The system shall create a text file with a list of include directories used and a text file with a list of source files used. The system shall create dependency files for each source file and for each header that is compiled.

The user will be able to set the compiler program, compiler flags, compiled headers for particular source files, include directories list, recursive include directories, source files, source directories, recursive source directories, and source file-types in the configuration file.

- clean: the system shall print the individual duration in units of seconds that the program spent on the operation.
  - everything
  - a specific configuration
  - a specific module in a specific configuration

3.3 Performance Requirements

Multiple users may run the software simultaneously, so long as they don't use it to build the same module or configuration at the same time, which would cause problems. Also no user should clean any part of the project at the same time that another user is building that part of the project.

3.4 Logical Database Requirements

Omitted because there is no database.

3.5 Design Constraints

Omitted because there aren't any design constraints imposed by other standards.

3.5.1 Standards Compliance

Omitted because the software doesn't follow any standards or regulations.

3.6 Software System Attributes

There must be a file system.

3.6.1 Reliability

The contents of the build folder should not be modified while the software is running. If any changes to the configuration file occur while the software is running, it may or may not use the new version depending on timing.

3.6.2 Availability

The software can be stopped with SIGINT. This won't corrupt the build.

3.6.3 Security

compiler/linker flags and other string values in the configuration file are used directly in the build commands. Therefore they should not contain any command line escape characters.

3.6.4 Maintainability

The code for interfacing with gcc for compiling should be put in its own class. And the code for interfacing with “gcc” for linking should be put in its own class. The reason for these requirements is that they make it easy to add support for other toolsets like “clang”.

3.6.5 Portability

The software doesn't have any host dependent code. It is portable to any machine that has a python version 3 interpreter installed.