Halpern-Wight Software, Inc.
22 Half Moon Hill
Acton, MA 01720-2449

978-263-9095
www.halpernwightsoftware.com

CLI Power Tools

Command Description Language (CDL) High-level Overview

Summary

The heart of CLI Power Tools is the Command Description Language (CDL). A developer using CLI Power Tools uses the CDL to create the commands, modes and data types that the end-user will see in the finished CLI. This document is an overview of the CDL. It is intended to convey the flavor of CDL and give the reader a quick understanding of CDL concepts. This is not a tutorial, user guide, or reference manual.

CDL Source Files and the CDL Compiler and Linker

CLI commands are defined in CDL files whose names end in `.cdl`. Each `.cdl` file defines the syntax for a set of closely related CLI commands, along with help text for those commands and the actions needed to implement those commands. In order to add a new command to your CLI, you create a new CDL file or add a new command to an existing CDL file. CDL files also define CLI modes, CDL macros and CDL token types.

Once you have created the CDL files necessary to specify your CLI, you must process the CDL code with the CDL Compiler and CDL Linker. For each CDL file, the CDL compiler generates three output files: a `.cdo` file, a `.cc` file, and a `.hh` file. All of the generated `.cdo` files are combined by the CDL linker to generate compact parse tables that are used by the runtime library to parse user's input. The

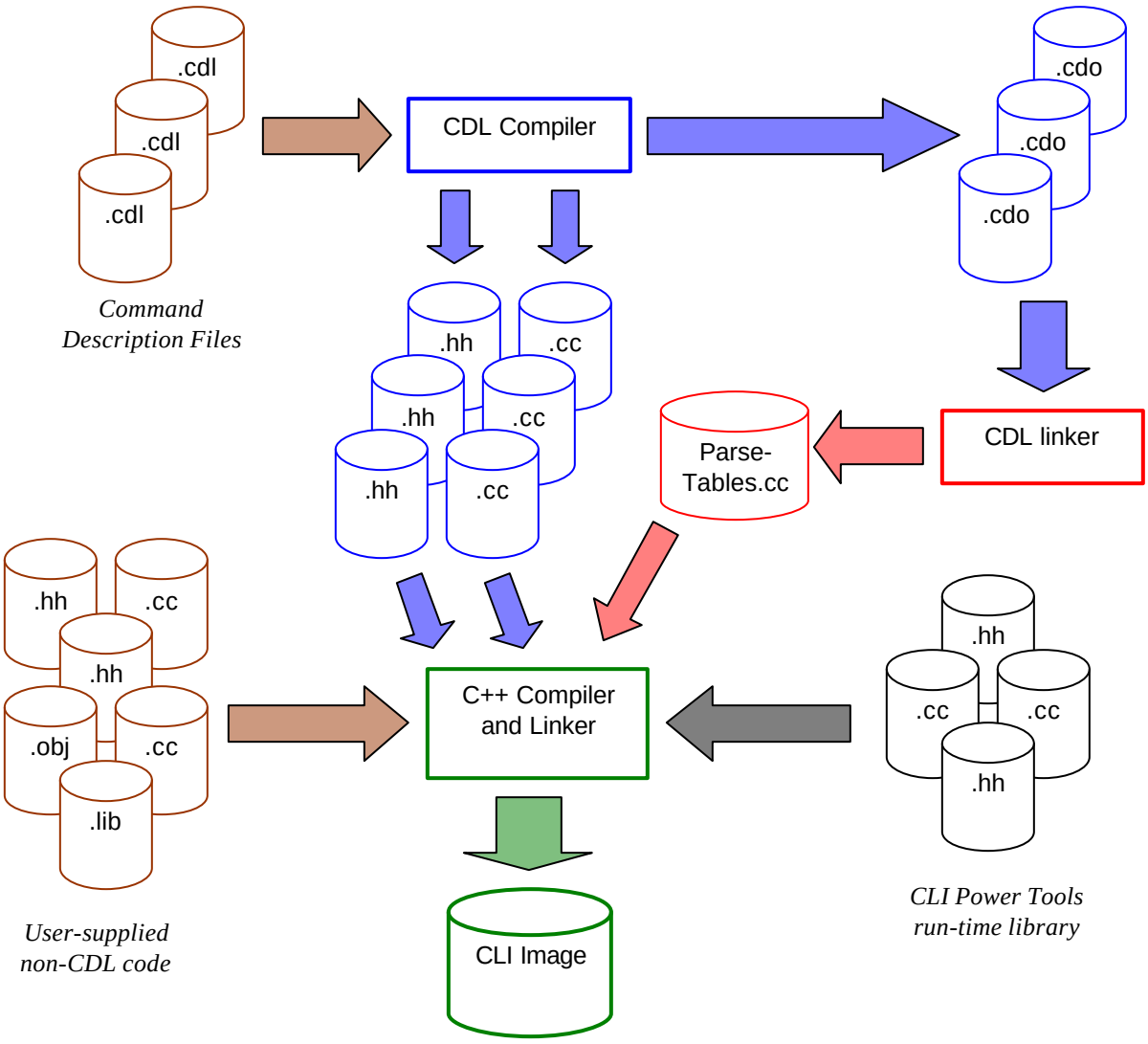

Figure 1: Flow of data through CLI Power Tools

parse tables are formatted as C++ code in a file called `ParseTables.cc`. The generated `.cc` and `.hh` files are standard C++ files containing the code necessary to decode and execute the commands specified in the original `cdl` file. The `.cc` files generated by the CDL compiler and the `ParseTables.cc` file generated by the CDL linker are compiled using a standard C++ compiler. The CLI runtime library, as well as any other user-supplied code, is linked with the generated files to produce the executable CLI image. The overall data flow is shown in Figure 1.

Command Definitions

The syntactic structure developers will use the most is the command definition. A command definition consists of the `command` keyword followed by a command name and a block of statements describing the command:

```
include "modes.cdl";

/* This command does the obvious.
 * No programming document would be complete without it.
 */
command HelloWorld
{
      mode ExecMode;
      privilege PRIV_ANY;

      syntax hello [ <freq> | again ] world;
      keyword hello { "Greet somebody"; }
      Uint(1, 5) freq { "How many times to repeat the greeting."; }
      keyword again { "Repeat the greeting twice"; }
      keyword world { "Greet the whole world"; }

      // We've seen this before, haven't we?
      execute
      {
            unsigned repeat = 1;
            if (HAS_PARAMETER(freq))
                  // Parser guarantees that freq is 1 to 5
                  repeat = freq;
            else if (again)
                  repeat = 2;

            while (repeat–)
                  printf("hello world\n");

            return cliOk;
      }
}
```

The above example illustrates a number of concepts in the CDL. In this overview, we will not go into the details and variations of each syntactic construct, but we will point out some of the key concepts:

- The code is structured like C or C++. The command definition looks like a C `struct` or a C++ `class`. The CDL compiler will generate a structure type for each command definition.

- Comments are enclosed in C-style comment delimiters (/* … */) or C++-style comment delimiters (// … newline).

- The `include` statement imports definitions much like the C `#include` directive.

- Each command can belong to one or more modes, as listed in the `mode` statement.

- Each command has a privilege level, as indicated in the `privilege` statement. Privileges do not have to be hierarchical (i.e., the tools support mutually-exclusive privileges that can be combined on a per-user basis.)

- The syntax for a command is described using a regular-expression grammar including keywords, variable parameters, optional syntax enclosed in square brackets (`[` and `]`), single-choice options

separated by vertical bars (|), etc. Other features not shown include the ability to specify a list of options that can be entered in any order and options that can be repeated more than once.

- Each keyword or variable parameter can be assigned a help message that will be displayed when the user asks for context-sensitive help.

- Each variable parameter is declared with a type and an optional range. Range checking is done automatically by the run-time system and does not need to be explicitly performed in the execute function.

- The execute function contains user-supplied C or C++ code that is copied verbatim into the generated code. This code does the real work of the command and can access anything in the environment, i.e. hardware registers, operating system files, or messaging protocols like SNMP. There is a straight-forward mechanism for accessing command-line parameters. **Note:** You do not need to know C++ in order to use CLI Power Tools.

Other Features

The HelloWorld example above illustrates many of the most important features in the CDL and should give you a sense of how straight-forward it is to translate a command set into actual code. The CLI runtime provided with CLI Power Tools does most of the tedious parsing and error checking, leaving the implementation of each command small and clean.

CDL does not stop there, however. In addition to commands, there are CDL constructs for defining modes, data types, and re-usable syntax fragments called macros. In addition, CDL provides constructs to help you build a Cisco-like `show running-config` command by letting you associate special command-generation code with each command. The generation code would access the state of the system and generate the commands necessary to put the system into that state.

Learn More

If you want to know more about CLI Power Tools, contact us for a demonstration:

By phone: 978-263-9095 (ask for Pablo Halpern)

By email: clipowertools@halpernwightsoftware.com