CLICON Tutorial

Olof Hagsand and Benny Holmgren

July 2014

1 Introduction

CLICON is a software suite for configuration management including CLI generation, YANG specification, NETCONF interface and embedded databases. White-box systems, embedded devices and other systems can with a few steps add a conistent management interface with CLI, netconf access, realtime-database and transactions support.

This tutorial gives an overview of a simple application for configuring NTP, the Network Time Protocol. The complete application contains a CLI, A YANG specification, an embedded database, a configuration engine and a Netconf interface.

1.1 Installation

The application is a part of the CLICON source-code release which can be installed on a variety of platforms using configure, then compiled and installed. Installation installs libraries, binaries and default configuration files in the system. It is also possible to install a development environment using install-include.

A typical CLICON installation is as follows:

```
> configure
> make
> sudo make install
> sudo make install-include
```

The NTP application used in this tutorial is accessed in the directory `examples/ntp`. You install the tutorial example after installing CLICON as follows:

```
> cd examples/ntp
> make
> sudo make install
```

Thereafter, `clicon_backend` and `clicon_cli` are started as follows:

```
> sudo clicon_backend
clicon_config[17651]: clicon_backend: 17651 Started
> clicon_cli
olof@ntp>
```

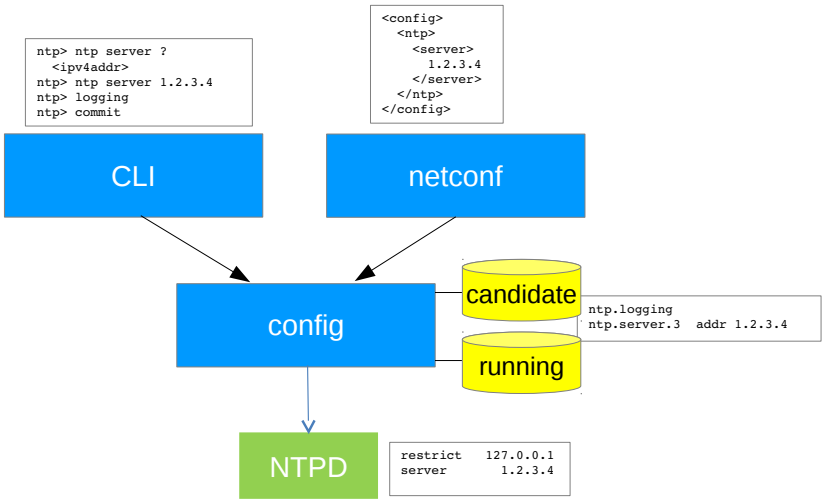

Figure 1: *An example of application flow in CLICON. Users enter commands as CLI or XML statements. A backend daemon manages the candidate and running databases and updates the system (such as the NTP daemon) when database entries change.*

2 Application flow and commit semantics

Figure 1 shows a typical CLICON system where a user interacts with a local CLI to configure and manage a system. In the small example of this tutorial, the user manages an NTP service. The user adds new NTP servers, configures logging and shows the status of the service.

Example:

```
olof@ntp> ntp server ?
  <ipv4addr>              IPv4 address of peer
olof@ntp> ntp server 1.2.3.4
olof@ntp> ntp logging status on
olof@ntp> show configuration
ntp server 1.2.3.4
ntp logging status true
olof@ntp>quit
```

The CLI statements above result in a change in the database which in turn rewrites the NTP configuration file. The new statements take effect when the NTP daemon is notified of the new configuration. The corresponding database and ntpd statements can be seen in Figure 1.

Looking in some more detail, the CLI statements are sent as modification statements to a *backend* daemon which adds them in a *candidate* database. When the user *commits* the change, the config daemon copies the new state into a

running database, changes the NTP configuration file and restarts the NTP daemon.

The backend daemon knows how to update the NTP configuration file since there are associations between database entries and callbacks. When, for example, an `ntp.server` entry is added in the running database, a `server` entry is added to the NTP configuration file, and the daemon is restarted.

If the commit fails, the whole transaction is reverted, the running database is rolled back to a previous state, and the user is notified of the failed commit.

Note that as an alternative, one can define the CLI to use *auto-commit* which means that every configuration statement is executed directly and feedback on failure will be immediate. The ntp example uses autocommit.

The commit semantics as described follows Netconf [5]. One can use a Netconf client to modify or access configured data using XML. The netconf statement in Figure 1 shows a subset of an `edit-config` command, which accomplished an equivalent modification as the CLI command.

The CLI typically has more functionality than handling configurations. For example, it may have commands for examining the configured state, for invoking operations (eg 'ping'), and for showing statistics. However, it is the handling of configurations that is the main objective of the CLICON software, and thus of this tutorial.

3 Software architecture and plugins

CLICON provides a runtime with configuration interfaces for embedded applications. An application developer designs new applications by editing YANG and CLI specification files and programming plugin C-code.

When done, the application is "clicked-on" offering an interactive CLI, a configuration daemon with transaction semantics, an embedded database and a netconf interface for remote machine-machine configuration.

Figure 2 shows a CLICON runtime with a CLI client, a Netconf client and a backend daemon. The figure shows an extended example, not only NTP, but also a hello functionality. The clients communicate with the backend server over a Unix domain socket. A configuration file contains common configure options for the system as a whole.

An important concept in CLICON is the use of application-specific *frontend* and *backend plugins*.

3.1 Frontend plugins: CLI and netconf

Frontend plugins, as shown in the upper part of Figure 2, define how users enter commands in the CLI or process specific messages in Netconf. Netconf plugins are not always necessary, since the standard functionality is often enough.

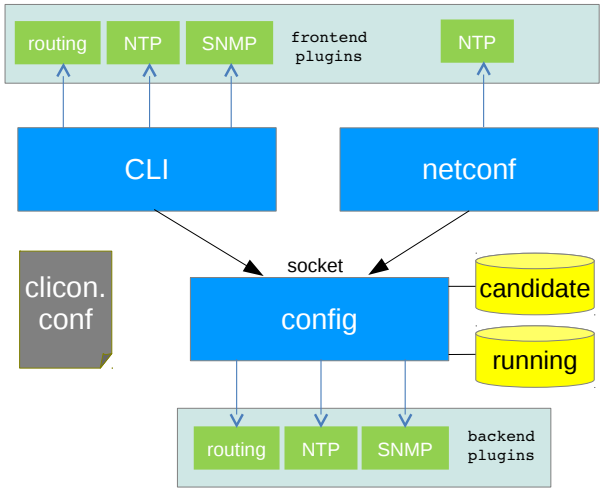

Figure 2: *CLI and Netconf clients communicate with the backend server over a Unix socket. The CLI loads frontend plugins, and the config daemon loads backend plugins.*

CLI plugins contain user callbacks that define how CLI commands are translated into the underlying database. Many of these commands are pre-definied by the system. For a simple system, the CLI plugin can be quite small. The CLI client loads the frontend plugins at runtime into its executable image and loads CLI specifications, defines callbacks, etc. In the figure, there are three CLI and one netconf plugin.

3.2 Backend plugins

Application-specific backend plugins are loaded by the configuration daemon. Each backend-plugin registers *callbacks* with database entries: When the entries change, the callbacks are called. The callbacks performs some action depending on the new settings of the database entries.

Figure 2 shows three backend plugins: routing, NTP and SNMP. The NTP plugin configures the NTP configuration files and restarts the NTPD daemon.

4 Developing applications

CLICON in itself consists of client and daemon binaries, libraries and include files. A developer designs an application by editing the *specification files* and programming the *plugins*.

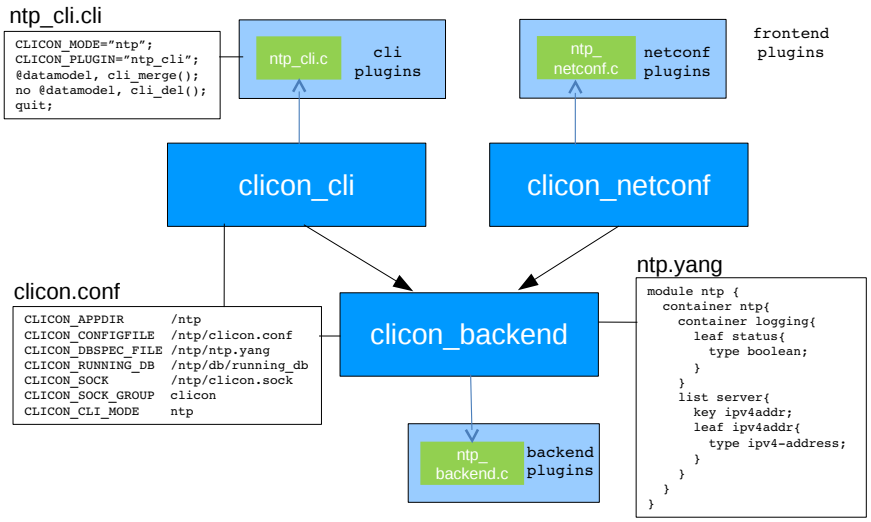

Figure 3: *Example of a CLICON application with plugin source code, CLI syntax and YANG datamodel specifications.*

Figure 3 shows the developing environment for the NTP application. The figure shows the following specification files:

- clicon.conf - Basic CLICON configuration file, see the reference manual [4] for more information. Basic and global configuration options are defined here.
- ntp.yang - The datamodel specification in YANG describes the configuration data, and what state is stored in the database. It is also used to generate CLI syntax. See Section 5 for detailed information.
- ntp_cli.cli - The syntax specification of the CLI commands. Section 6 shows the syntax specification of the NTP application.
- ntp_cli.c - Contains callbacks for the CLI commands. This is described in more detail in Section 7.
- ntp_backend.c - The backend plugin registers functional callbacks, triggered when database entries change. The backend module of the NTP application is described in more detail in Section 8.
- ntp_netconf.c - The netconf plugin defines netconf commands that extend the standard calls. Typically operational commands.

5 The database specification: ntp.yang

The data model specifies a set of keys and values of keys used in the embedded database. The syntax of the data model is a limted YANG syntax [6].

The data model for the NTP example consists of IP addresses to NTP servers and a logging flag:

```
container ntp{
   container logging{
      leaf status{
         type boolean;
      }
   }
   list server{
      key ipv4addr;
      leaf ipv4addr{
         type ipv4-address;
      }
   }
}
```

The data model as shown above contains three kinds of configuration entries: a container which is a sequence of sub-entries, a list of servers each containing a unique IPv4 address, and two leafs containing variable data. The `key` statement indicates that the address is the unique index variable of that list.

The given datamodel is very simple, more elaborate data models include:

- default values for variables not given by the user
- leaf-list, list of varaibles
- mandatory vs optional leafs
- descriptions
- more advanced types, and imports of other modules

The YANG data model is central in CLICON for several reasons:

1. It determines which database entries are valid,
2. CLI configuration commands are generated from the data model,
3. It determines valid NETCONF XML configuration statements.

Based on the data model, the runtime (embedded) databases (eg running and candidate in Figure 1) will be populated by database keys as specified by the model.

Note that the data model describes a tree, while the database consist of keys. This mapping is straightforward and is illustrated as follows [1]

```
ntp.server.3  ipv4addr=1.2.3.4
ntp.server.2  ipv4addr=4.45.5.6
```

[1] The output is a summarized from the `clicon_dbctrl` application

```
ntp.server.0  ipv4addr=2.2.2.2
ntp.logging   status=true
```

The listing shows the content of a runtime database. The three `ntp.server` keys have one variable each, the index variable `ipv4addr` defining the unique index. The single ntp.logging key containing a single variable that will be overwritten if a new value is given.

6 The CLI syntax specification: ntp_cli.cli

The CLI syntax is specified in a syntax definition language as specified in CLIgen [1]. The CLI specification is a tree of commands with associated help texts.

The complete CLI syntax for the NTP application is as follows[2]:

```
CLICON_MODE="ntp";
CLICON_PLUGIN="ntp_cli";
CLICON_PROMPT="%U@%H> ";
show{
    associations, cli_run("ntpq -p");
    configuration, show_conf_as_cli("running ^.*$");
    netconf, show_conf_as_netconf("running ^.*$");
}
@datamodel:ntp, cli_merge();
no @datamodel:ntp, cli_del();
quit, cli_quit();
```

The specification starts with variables defining the CLI behaviour. The reference manual [4] lists such variables with their default behaviour.

Each leaf in the tree may specify a callback function with arguments. For example, "show associations" invokes the `cli_run` command with `ntpq -p` as argument.

All callbacks in this example (such as `cli_run`) are standard callbacks defined in the CLICON library. Many useful callbacks included the ones in the example are listed in [4].

If the developer needs to define application-specific callbacks, this is done in the CLI plugin, in this case `ntp_cli.c`. How to define such callbacks is described in the CLIgen documentation [1] and Section 7.

The CLI syntax also has two examples of *tree references* in the form of the '@datamodel:ntp. In this case, this is a macro expansion of generated CLI syntax from the NTP datamodel YANG module as defined in Section 5.

For example, the line `no @datamodel:ntp, cli_del();` is translated to:

```
no ntp{
   server <ipv4addr:ipv4addr>, cli_del();
   logging status <status:bool>, cli_del();
}
```

[2]Help texts have been removed for clarity

In the exaxmple, comments and variables are substituted. The advantage with generating CLI commands from a YANG specification is that configuration statements need not be entered again in the CLI specification, thus making the specification smaller.

If the application needs better control of the CLI configuration syntax, different from the generated, it is also possible to manually define how CLI commands are translated into database keys using explicit calls to `cli_set, cli_merge and cli_del`.

You need to set some cligen variables to control the frontend:

- CLICON_PROMPT. Prompt to use in this mode. using %H (host) %U (user) and %T (terminal) format.
- CLICON_PLUGIN. Which frontend plugin (.so) to lookup callback functions in. You can use callback functions in the cligen syntax only from one plugin, or from the clicon library. If you do not specify a plugin, the MASTER_PLUGIN will be used.
- CLICON_MODE. name of this mode. Use with -m option and when changing mode.

7 Frontend plugin: ntp_cli.c

The CLI plugin contains user-defined callbacks for CLIgen callbacks, and application-specific C-code. The NTP example needs no such specializations.

For CLI plugins, there are also some predefined callbacks available, the most common are:

- `plugin_init` - called as soon as the plugin has been loaded and is assumed initialize the plugin's internal state.
- `plugin_start` - is called once everything has been initialized, right before the main event loop is entered.
- `plugin_exit` - called in each plugin when the application is about to exit. It gives each plugin the chance to clean up after itself and do general housekeeping.

8 Backend plugin: ntp_backend.c

The backend plugin consists of predefined functions that follow a naming standard allowing them to be called from the configuration daemon. The initiator callbacks also typically sets up validation and commit callbacks.

Once the plugin is setup, the backend daemon reacts on transactions from the candidate to running database when a commit statement has been made. In case of autocommit, every configuration statement results in a transaction.

A transaction is a set of configuration commands that are executed as a whole. If one configuration statement fails the transaction is aborted and rollbacked to previous state.

- `plugin_init` is the first function called and sets up the dependency between database symbols and callbacks. In this case, any time an NTP entry changes, the `ntp_commit` function is called. The latter function actually only sets a variable for later processing. `plugin_start` - is called once everything has been initialized, right before the main event loop is entered.
- `transaction_begin` is called as the first callback in a transaction after a commit or validate command.
- `validation callbacks` as defined in `plugin_init` are called for every database entry that has changed. A validation callback should check the validity of the candidate database.
- `transaction_complete` is called when validation of all entries is complete.
- `commit callbacks` as defined in `plugin_init` are called for every database entry that has changed. The commit callbacks may change the actual system state.
- `transaction_end` is called after all commit functions are called. In this example, the actual communication with the NTP daemon is done here. If this would fail, the commit itself would be cancelled. `transaction_abort` is called if a commit calback fails and after the rollback.
- `plugin_exit` - called in each plugin when the application is about to exit. It gives each plugin the chance to clean up after itself and do general housekeeping.

8.1 Init

A backend plugin registers database keys. When the values of these keys change, the plugin gets notified. so that changes to these can be processed by the plugin. This is strictly only necessary in the incremental node.

Key registration is made using the `dpdep` function. The following example shows an init function registrating the keys "ntp.server[]" and "ntp.logging"[3]:

```
int
plugin_init(config_handle h)
{
   dbdep(h, TRANS_CB_COMMIT, ntp_commit, NULL, 2, "ntp.server[]", "ntp.logging");
}
```

In the NTP example however, the small-granular validation and commit is not used. Instead, the complete configuration update is made in `transaction_end`, and NTPD is restarted.

[3]The actual code uses a somewhat more general approach

Other applications may need to make incremental changes to the state, and then the commit callbacks are useful.

9 Summary

This document has shown a simple CLICON example, where a simple application (ntp) has been extended with a state-of-the-art configuration interface.

The configuration interface provided by CLICON consists of:

- An interactive CLI
- A NETCONF XML configuration interface.
- An embedded runtime database
- Commit/validate transaction semantics
- A YANG datamodel specification

CLICON code is GPLv3 an is available via github `https://github.com/clicon/clicon`.

References

[1] Olof Hagsand, *CLIgen tutorial*, CLIgen technical documentation, April, 2011

[2] Olof Hagsand and Benny Holmgren, "CLICON data model", Technical documentation, September 2013.

[3] Olof Hagsand and Benny Holmgren, "CLICON db2txt formatting", Technical documentation, September 2013.

[4] Olof Hagsand and Benny Holmgren, "CLICON Reference Manual", Technical documentation, September 2013.

[5] R. Enns, Ed, *NETCONF Configuration protocol*, RFC 4741, IETF, Dec, 2006

[6] M. Bjorklund, Ed, *YANG - A Data Modeling language for the Network Configuration Protocol (NETCONF)*, RFC 6020, IETF, Oct, 2010