Installing the OpenLCB Checker Software (Basic Version)

November 8, 2024

1 Introduction

This document describes how to obtain and run a set of basic compatibility checks for OpenLCB node implementations.

The checks are based on the Python 'openlcb' module. More information on that can be obtained from its GitHub project site.

For more information on the checks, see the directory of checking plans.

2 Obtaining the Software

The checker software is distributed as a set of inter-connected Python source files.

2.1 Obtaining and Using via Git

If you're using Git,

```
cd (where you want to put this)
git clone https://github.com/bobjacobsen/OlcbChecker.git
```

will create a OlcbChecker directory containing the most recent version of the software. This also contains git tags for the released versions.

2.2 Obtaining by Downloading a .zip File

You can get a download of the most recent released version by going to the project's Github releases web page tag section [1] and clicking the .zip or .tgz icon on the most recent release.

[1] Linked above or see https://github.com/bobjacobsen/OlcbChecker/tags

To get the very most recent version, [2] go to the project's Github main web page tag section, click the green Code button, and select "Download Zip".

Expand the downloaded file in a suitable place.

2.3 Prerequisites

You need to have Python 3.10 or later installed to run the program. Consult your computer's documentation for how to install that. Many computers already have it installed.

You have to manually install the 'openlcb' module by checking out the python-openlcb repository from GitHub. [3] To do this:

```
cd (where you want to put it)
git clone https://github.com/bobjacobsen/python-openlcb.git
python3.10 -m pip install --editable python-openlcb
```

This will create a python-openlcb directory containing the most recent version of the software and make that available as a 'openlcb' Python module.

If you want to connect to your OpenLCB network via a serial adapter, the 'pyserial' Python module must be installed. To do that, enter[4]

```
python3.10 -m pip install pyserial
```

To run the CDI checks, the 'xmlschema' Python module must be installed. To do that, enter[5]

```
python3.10 -m pip install xmlschema
```

3 Starting the Program

First, do

```
cd (your OlcbChecker directory)
```

to get to the right directory for running the code.

To start the program:

```
python3.10 control_master.py
```

[2] But if you want to stay current with development of the tools, you should probably be using Git.
[3] This will eventually be available via PIP, but not yet.
[4] This is the command for Linux nad MacOS; the Windows command may be different.
[5] This is the command for Linux nad MacOS; the Windows command may be different.

Depending on your Python installation, this simpler form may also work:

```
./control_master.py
```

4 Configuring the Checker Program

When you first start the program, you'll be shown a basic menu:

```
OpenLCB checking program
 s Setup

 0 Frame Transport checking
 1 Message Network checking
 2 SNIP checking
 3 Event Transport checking
 4 Datagram Transport checking
 5 Memory Configuration checking
 6 CDI checking

 q  Quit
>>
```

Type s and hit return to get the setup menu:

```
The current settings are:
  hostname = None
  portnumber = 12021
  devicename = None
  targetnodeid = None
  ownnodeid = 03.00.00.00.00.01
  checkpip = True
  trace = 10

c change setting
h help
r return
>>
```

At a minimum, you should define how to connect to your OpenLCB network, and the Node ID of the device you want to check.

To change the Node ID, select the"change setting" option and work through the prompts:

```
>> c
```

```
enter variable name
>> targetnodeid
enter new value
>> 02.01.57.00.04.9A
The current settings are:
  hostname = None
  portnumber = 12021
  devicename = None
  targetnodeid = 02.01.57.00.04.9A
  ownnodeid = 03.00.00.00.00.01
  checkpip = True
  trace = 10

c change setting
h help
r return

>>
```

Get the proper value from either a label on the device, or from its documentation. [6]

There are currently two ways to connect the program to your OpenLCB network:

1. Via a USB-CAN adapter, or
2. Via a GridConnect-format TCP/IP connection.

Note: Some checks involve resetting the node being checked. If you're using a GridConnect-format TCP/IP connection, this will break the connection and cause those checks to fail. If it's possible to do so, it's better to run the checks using a USB-CAN adapter and a CAN cable.

For a USB-CAN connector, define the devicename to be the address of the device in your computer, e.g. /dev/cu.usbmodemCC570001B1 or COM7.

For a TCP/IP link, define the hostname to be the IP address or host name to be used for connecting, e.g. 192.168.0.200 or localhost.

You must specify one or the other of hostname and device name, but not both. When you enter one, the other will be set to None.

When done with setup, select r for return.

[6] Some checks, but not all, can determine the node ID themselves if you leave the value as None. This is only reliable if there's just one node on your OpenLCB network. Note that some OpenLCB hubs add a node of their own to the node being checked.

You'll be asked if you want to save changes. Select y to save and n to skip saving.

```
>> r

Do you want to save the new settings? (y/n)
 >> y
Stored
Quit and restart the program to put them into effect
```

Quit and restart the program to put your changes into effect.

5 Running Checks

5.1 Required Equipment

It's generally best to have the device being checked (DBC) as the only device on the OpenLCB network.

If a direct CAN connection will be used, a supported USB-CAN adapter is required [7]. Connect the adapter to your computer as indicated in its instructions. Connect the adapter to the DBC using a single UTP cable and attach two CAN terminators.

If a TCP/IP GridConnect connection will be used, configure the DBC to connect to the TCP/IP hub when restarted. Note that if the DBC is providing the hub for the connection and restarting the DBC breaks connections to that hub, several of the checks will indicate problems due to the connection breaking.

Provide power to the DBC using its recommended equipment and connections.

5.2 The Checking Sequence

The checks are categorized by the Standard they primarily check:

```
OpenLCB checking program
 s Setup

 0 Frame Transport checking
 1 Message Network checking
 2 SNIP checking
 3 Event Transport checking
 4 Datagram Transport checking
 5 Memory Configuration checking
```

[7] The checker has been checked with the RR-CirKits LCC buffer-USB and the SPROG DCC Ltd LCC-USB adapter but others with similar operational characteristics will probably work.

```
 6 Configuration Definition Information (CDI) checking
 7 Train Control checking
 8 Train Search checking
 9 Function Definition Information (FDI) checking

 a Run all in sequence
```

It's generally best to run them in order, as issues flagged by earlier checks may prevent later ones from running properly.

Once you select a section, you'll be presented with a sub-menu of the actual checks. These match the sections in the relevant plan document. For example:

```
Frame Transport Standard checking
 a Run all in sequence
 1 Initialization checking
 2 AME checking
 3 Collision checking
 4 reserved bit checking
```

"Run all in sequence" is usually a good starting point.

The checker may prompt you to reset/restart the node or to compare something to the node documentation. For example, the Initialization checking requests that you restart the node so that its start-up sequence can be checked for validity. It waits for about 30 seconds, and if it still hasn't seen anything, the check fails.

6 Logging

Output is defined by a 'logging.conf' file in the main directory.

By default, it logs to two places:

1. To the console, i.e. the terminal screen. The logging level and format for this is defined by the 'consoleHandler' and 'consoleFormatter' clauses. By default, this shows INFO and above messages preceeded by the name of the check module being run.

2. To the 'log.txt' file. This file is restarted every run, so it contains only the most recent content. The logging level and format for this is defined by the 'fileHandler' and 'fileFormatter' clauses. By default, this shows INFO and above messages preceeded by the time, the name of the check module being run, and the logging level of the message.

The -T option can be used to override this. Setting this option to various values sets the minimum logging level:

- 0 Only WARN and ERROR messages
- 10 INFO, WARN and ERROR messages. This is the default set by the logging.conf file.
- 20 Includes also DEBUG level messages.

7 Technical Information

Your selected defaults are stored in the localoverrides.py file. The original values are stored in the defaults.py file.

Should something corrupt the localoverrides.py file, you can delete it, restart the program, and re-enter your configuration.

There are several kinds of Python scripts in the package.

1. Scripts named check_*.py, which hold the checks for each specific step. The naming contention is e.g. check_fr20_ame.py which is the 2nd (20) check of the frame level (fr) and does some AME checking (ame). [8] These can be individually run e.g. to rapidly exercise a feature for debugging.
2. Scripts name control_*.py which hold the simple keyboard-oriented program for running the check scripts. control_master.py is the main script, which invokes a control_*.py script for each level of the checking and setup.
3. The olcbchecker directory and package contains general implementation files.

[8] The numbers are spaced by 10 to allow us to add new scripts in between.