Checking the OpenLCB Message Network Standard

The OpenLCB Group

February 10, 2024

1 Introduction

This note documents the procedure for checking an OpenLCB implementation against the OpenLCB Message Network Standard.

The checks are traceable to specific sections of the Standard.

The checking assumes that the Device Being Checked (DBC) is being exercised by other nodes on the message network, e.g. is responding to enquiries from other parts of the message network.

1.1 Required Equipment

See the separate "Installing the OpenLCB Checker Software" document for initial installation and set up of the checker program.

If a direct CAN connection will be used, a supported USB-CAN adapter [1] is required. Connect the adapter to the DBC using a single UTP cable and connect two CAN terminators.

Provide power to the DBC using its recommended method.

2 Set Up

The following steps need to be done once to configure the checker program:.

1. Start the checker configuration program.
2. Select "Set Up".
3. Get the Node ID from the DBC[2]

[1] See "Installing the OpenLCB Checker Software"

[2] Where do we require this to be marked on a node?

4. Enter that Node ID into the program.
5. Configure the checker program for the USB-CAN adapter's device address or the TCP hostname and port.
6. Return from the setup section and reply "Y" to "Save configuration?" when prompted.
7. Quit from the program.

The following steps need to be done at the start of each checking session.

1. Check that the DBC is ready for operation.
2. Start the checker program.

3 Message Network Procedure

Select "Message Network checking" in the program, then select each section below in turn. Follow the prompts for when to reset/restart the node and when to check outputs against the node documentation.

3.1 Node Initialized checking

This section checks the interaction in Standard section 3.4.1 Node Initialization.

It does this by having the operator reset/restart the DBC and then checking that a Node Initialized message is received and

1. The Node Initialized message is the first message received. [3]
2. The message indicates its source is the DBC.
3. The message uses the appropriate MTI [4] for whether PIP indicates the node uses the Simple Protocol or not.
4. The data part of the message contains the source node ID.

3.2 Verify Node checking

This section checks the interaction in Standard section 3.4.2 Node ID Detection.

It does this by

1. Sending a global Verify Node ID message and checking the reply.
   (a) The reply message indicates its source is the DBC.

[3] See Section 3.2 of the Standard
[4] See Section 3.3.1 of the Standard

(b) The reply message contains the Node ID of the DBC in its data field.

(c) The reply message uses the appropriate MTI [5] for whether PIP indicates the node uses the Simple Protocol or not.

2. Sending a Verify Node ID message addressed to the DBC and checking the reply.

   (a) The reply message indicates its source is the DBC.

   (b) The reply message contains the Node ID of the DBC in its data field.

   (c) The reply messages uses the appropriate MTI for whether PIP indicates the node uses the Simple Protocol or not.

3. Sending a Verify Node ID message addressed to a different address from the DBC and checking for the absence of a reply.

3.3 Protocol Support Inquiry checking

This section checks the interaction in Standard section 3.4.3 Protocol Support Inquiry and Response.

It does this by

1. Sending a Protocol Support Inquiry message to the node, and checking for the resulting Protocol Support Response message.

   (a) The reply message is addressed to the checking node.

   (b) The reply message indicates its source is the DBC.

   The check displays the resulting list of supported protocols for checking against the DBC's documentation.

2. Sending a Protocol Support Inquiry message addressed to a different address from the DBC and checking for the absence of a reply.

3.4 Optional Interaction Rejected checking

This section checks the interaction in Standard section 3.5.1 Reject Addressed Optional Interaction.

It does this by sending an unallocated MTI [6] addressed to the DBC and checking for the Optional Interaction Rejected message in reply.

[5] See Section 3.3.3 fo the Standard

[6] Initially 0x048. Others may be added in the future for an eventual check of all possible unallocated MTIs

1. The reply message is addressed to the checking node.
2. The reply message indicates its source is the DBC.
3. The reply message carries at least four bytes of content, with the third and fourth bytes carrying the original MTI.

3.5 Duplicate Node ID Discovery checking

This section checks the interaction in Standard section 3.5.4 Duplicate Node ID Discovery.

It does this by sending a global Verify Node message with the source ID set to the DBC's ID.

It then checks for the well-known global event. If it finds that, the check passes. If not, it prompts the operator to confirm that indication has been made "using whatever indication technology is available", e.g. via LEDs.

4 Frame Level Procedure

This section is only applicable to implementations what use a CAN-format link layer implementation, e.g. via a USB-CAN adapter or by using Grid Connect coding over a TCP/IP link. See Standard section 7.

All checks in the prior section are assumed to have passed before these checks are run, as they depend on message-layer behaviors.

In general, proper operation of the frame level is checked by exercising it through the message level in the checks above and in subsequent documents. We limit ourselves here to checking atypical behaviors.

4.1 Reserved Field Handling

This section discusses the reserved bit fields described in Standard section 3.1.1.

The CAN implementation does not carry bits 15-14 nor bit 12 of the MTI. The full 12 bit sof the rest of the MTI are checked in other sections of this check plan.

However, in a CAN implementation, the 0x1_0000_0000 bit is reserved, send as 1, don't check on receipt.

To check this, the checker sends a Verify Node ID Global with zero in the 0x1_0000_0000 bit. It then checks:

1. That a Verified Node ID message frame is received from the DBC,

2. That the 1_0000_0000 bit in that frame is a one.