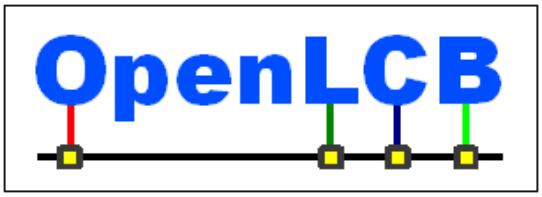

OpenLCB Technical Note	
Simple Node Information Protocol	
Feb 9, 2015	**In Review**

1 Introduction

One of the important innovation of OpenLCB compared to other layout control systems is the ability for end-users to centrally browse, manage and configure nodes connected to the OpenLCB network. This management and configuration is usually achieved by a computer software with a Graphical User Interface, such as JMRI.

These user interfaces need three major features: node discovery, node presentation in a user-friendly way (in other words *read access*), and node configuration (in other words *write access*). The protocols defined for these purposes have to be simple enough for all nodes to implement, and robust and scalable enough to work from the smallest networks to the largest layouts or module meets.

The Simple Node Information Protocol is a light-weight and scalable protocol designed for the middle use-case: to allow a network browse tool to read a small set of node identification information (such as the manufacturer and model, as well as user-specified name and description) efficiently. This information can then be used to present the node in a UI and allow the user to recognize the node in a possibly long list or table of nodes, and to select it as a target for possible interactions.

OpenLCB provides a very versatile protocol stack for communicating between nodes, which allow very generic transfers to take place between nodes, including accessing read-only or read-write memory areas of the target node, for example via the Memory Configuration Protocol. This protocol can be parametrized to transmit the same information as the Simple Node Information Protocol.

The Simple Node Information Protocol on the other hand has been designed with simplicity and efficiency in mind. It is able to provide only a fixed and very small set of information. On the other hand, this makes it easy to supply data to fixed UI elements, or even very limited UI (such as a single-line alphanumeric display as it could be found on a throttle). Also the the implementation state flows are very simple, and allow this protocol to be implemented on the smallest (and cheapest) nodes with very minimal engineering effort. This is an important consideration, because manufacturers are naturally interested in driving the costs down, but the protocol stack should still support the node browsing use-cases. Another important consideration is efficiency: how much network traffic is required to retrieve basic information per node. This becomes important for very large networks, where the mere enumeration of the available nodes could lead to network congestion.

Larger nodes are therefore encouraged to still implement this protocol to allow better handling of the entire network by network management tools.

For a Node to implement the Simple Node Information Protocol (as a target node), the following needs to be done:

- In the Protocol Support Reply message, indicate the availability of the Simple Node Information Protocol (SNIP) protocol.
- Listen for incoming Simple Node Information Requests.
- Reply to each of these requests by sending back one or more addressed messages to the source Node, containing the fixed payload. Ensure the framing bits are appropriately set when using CAN.
- When a write operation (via any Protocol) changes the user-specified information, reset the node or transition to Uninitialized state, and transition back to Initialized state by sending Node Initialization Complete message.

2 Annotations to the Standard

2.1 Introduction

Note that this section of the Standard is informative, not normative.

2.2 Intended Use

Note that this section of the Standard is informative, not normative.

2.3 Reference and Context

The OpenLCB Message Network Standard defines the CAN frame encoding of addressed OpenLCB messages. The reply generated by the Simple Node Information Protocol is an addressed message, and thus is governed by this formatting requirement. While CAN-bus messages can only transmit 8 bytes of data at a time, the response payload to send can be up to 253 bytes long. This requires the use of multiple CAN frames. The OpenLCB Message Network Standard defines two bits in the first data byte which allows sending multiple frames that constitute a single addressed message.

The Protocol Support Inquiry and Protocol Support Reply messages allow a node (for example the network browser node) to check what protocols a particular destination node supports. If a node implements the Simple Node Information Protocol, the network browser will know from the Protocol Support Reply.

The OpenLCB protocol stack provides a number of methods, some very flexible, to exchange data between participating nodes. One of these methods is laid out to represent the same information as provided by the Simple Node Information Protocol, albeit in a more general way. The OpenLCB Datagram Transport Protocol can be used to send medium-sized messages between nodes, the OpenLCB Memory Configuration Protocol can be used on top of this to provide read-only or read-write access to blocks of memory in the target node. The exported blocks are organized as so-called Memory Spaces, and two of these spaces are reserved to provide the same information as the Simple Node Information Protocol. The space 0xFC gives access to the read-only manufacturer-specific data, while the space 0xFB gives access to the read-write user-modifiable data. These two data blocks together give the information returned by the Simple Node Information Protocol. The format is described in the OpenLCB Configuration Description Information Standard.

For a Node to implement the *alternative* retrieval method for this information, the following steps have to be made:

- In the Protocol Support Reply, indicate the availability of the Abbreviated Configuration Definition Information (ACDI) protocol.
- Implement the OpenLCB Datagram Transport Protocol and the OpenLCB Memory Access Configuration Protocol.
- Implement a read-only memory space 0xFC and export the manufacturer-specific identification information in a fixed format.
- Implement a read-write memory space 0xFB and export the user-specific identification information in a fixed format.
- If the Configuration Definition Information (CDI) memory space is implemented, then add the <acdi … /> tag to the XML description of the node configuration.

For details, see the above quoted standards.

2.4 Messages

This section of the Standard is normative, and describes the OpenLCB messages in a generic form (transport-agnostic). For examples of how these messages would look on CAN, see Section 2.7.

2.5 Payload Format

2.5.1 Format

The total size of the reply can be up to 253 bytes. A typical reply will be much shorter, because only the actual length of the strings have to be transported as opposed to the maximum length (like for example in the case of the Memory Configuration Protocol).

2.5.2 Versioning

The first version of this standard used the version identifiers 0x01 and 0x01 in the two sections, respectively. In order to allow future extensions of the data transmitted, we must ensure that nodes parsing the response according to the current version of the standard will be able to cope with a response generated by a Node according to a future version of the Standard.

The solution is that the version identifier byte will double as a count of the null-terminated strings in the particular data segment. Thus a receiving node can count the number of zero bytes in the received message to find the separator between the sections and identify the known fields.

2.6 Interactions

When a node gets a Simple Node Information Request, if possible it shall reply with one or more Simple Node Information Reply messages containing the node's information. If it's not able to process the request, (for example because it is already servicing another request), it shall send an Optional Interaction Rejected with an appropriate error code. It's recommended that the rejection message have the temporary-error bit set, so that the node sending the original request will retry it. The suggested error code is 0x2020, Buffer unavailable. See the OpenLCB Message Network Standard for the Optional Interaction Rejected message and for the error code format.

The first version of this Standard predates the ability to send long addressed messages over CAN-bus. There are Nodes produced which will send the reply in as many messages as the number of CAN frames necessary to cover the length, without using the framing bits. Therefore network browsers using the Simple Node Information Protocol must be prepared to receive the response in multiple messages. If there is no framing bits to signal the end of transmission, the receiving node may count the zero bytes to realize when the response transmission is finished.

While the Simple Node Information Protocol is a lightweight protocol, enumerating a large set of nodes will still put a considerable traffic onto the bus. Therefore it is strongly recommended for network browser nodes to cache the returned information in their memory to avoid unnecessary traffic in repeatedly requesting it. The Standard specifies that this cache must be invalidated when the Node Initialization Complete message is received from a particular node, and conversely, if the Node's information payload changes, it must re-enter Initialized state to notify any other node that may have cached the information to refresh it.

2.7 CAN Adaptation

For reference, we describe the messages defined in this standard as formatter for the CAN transport protocol. This section is not normative; should there be a conflict between this document and the Standards, then the Standards take precedence.

Name	CAN header	Data Content
Simple Node Information Request	0x19DE,8sss	fddd

Where 'sss' is the 12-bit CAN Alias of the source Node (the node sending the request), 'ddd' is the 12-bit CAN Alias of the target Node (being queried), and 'f' is typically 0.

Name	CAN header	Data Content
Simple Node Information Reply	0x19A0,8sss	fddd, up to 6 payload bytes

Where 'sss' is the 12-bit CAN Alias of the target Node (sending the reply), 'ddd' is the 12-bit CAN Alias of the source Node (the one that has sent the query), and 'f' is as follows:

- f = 0x0 for replies that fit into a single CAN frame
- f = 0x1 for the first frame of a reply with more than one frames
- f = 0x2 for the last frame of a reply with more than one frames
- f = 0x3 for each middle frames of a reply with more than two frames

Table of Contents

