OpenLCB Technical Note	
Protocol Identification Protocol	
Feb 21, 2015	~~Preliminary~~ Obsolete

Note: This document is obsolete. Its contents have been moved into the General Message Standard.

1 Introduction

OpenLCB defines a number of specific protocols for interacting with nodes, including event exchange, datagrams, streams, configuration, etc. These protocols are optional, in the sense that not every node will implement every one.

To determine which protocols a node implements, a Protocol Support Inquiry message is sent to the specific node. It will reply with a Protocol Support Reply message that contains six bytes of data. A specific bit position has been reserved for each defined protocol. If the bit is zero or not present, the protocol is not supported and requests to use it will result in a error. If present and 1, the protocol is supported.

It is not necessary to check whether an addressed protocol is supported by a node before attempting to use the protocol. If it's not, the standard error handling mechanism will indicate that.

This protocol provides a way to check, without errors, whether the protocol is supported. Avoiding errors provides a cleaner system. Further, this protocol can check support for protocols that use global (non-addressed messages); nodes are not permitted to return errors for global messages.

2 Annotations to the Standard

2.1 Introduction

(No annotations)

2.2 Intended Use

Although this protocol isn't mandatory, it's a very good idea to provide it. High-function nodes such as PC programs are likely to use it to customize their operation. Programmers are more likely to use this protocol rather than write special error-handling code for each of the many different protocols. Use of this protocol also avoids “intentional errors”, which might confuse or annoy users.

2.3 Reference and Context

We don't explicitly reference all the protocol definitions that are associated with specific bits. Implementors can find them from the protocol names, and we don't want to clog up this section of the Standard with a set of references that we'll have to update continually.

2.4 Messages

The CAN messages were defined to be part of the "simple" subset. Low-end nodes may want to implement this protocol so that higher-function nodes can easily learn their limitations.

Generally, the node designer will just provide a simple fixed value for the reply. On CAN, the entire frame is fixed except for the source and destination aliases. The destination alias can be taken directly from the request packet.

The requirement for messages of "six or more bytes" is to allow for future expansion. Messages of more than 6 bytes on CAN will be sent using the start/end message framing bits.

2.5 Interactions

Other nodes can snoop on these interactions to learn the protocols supported by a node, without having to send their own inquiry. It's not expected that the protocols supported by a specific node will change with time, although the Standard does not require that they be immutable. There's no mechanism for a node to request an update if the list of supported protocols changes.

A node supporting this protocol must promptly reply to the Inquiry message with the Reply message. On the other hand, the general message standard requires that a node not supporting this protocol reply to the Inquiry message with a Optional Interaction Rejected message. Either way, a node sending a Protocol Id Inquiry message can count on getting a rapid reply.

2.6 Protocol Identification Values

~~The length of the field is constrained by the CAN frame length.~~

OpenLCB is big-endian, so these have been assigned from the MSB of the 1st byte.

The MSB for Protocol Identification Protocol is always 1, because it has to be present for the reply to be generated.

~~0x00 00 00 00 00 08 through 0x00 00 00 00 00 01 are reserved for control of future expansion. They'll eventually be needed when there are more protocols defined than can fit in a single CAN frame. One way of doing the expansion is to return more than one frame, which the individual frames identified using these bits. To preserve compatibility with that possibility, nodes built now must ignore frames that have one or more 1 bits in that field, so that they are not confused by future expansion frames.~~

In general, nodes using this protocol don't need to know the meaning of bits defined after the node was created. A node is looking to see whether a particular protocol is present or not so that it can select what actions to take. A newly-defined protocol isn't among the things that the node will try to reason about. This allows us to eventually extend the length of the reply in any one of several ways.

We define the bits here, rather than in the individual protocol definitions, to reduce the risk of duplicate assignments. Duplicate assignments would be obvious here, but not so much when spread across separate documents.

Before the CAN message start/end bits were defined, the 0x00 00 00 00 00 0F bits in the PIP reply message were defined for expansion. Nodes where to ignore any frames with one or more of those bits set. That mechanism is no longer present, but some of those nodes exist, so for backwards compatibility those bits are permanently reserved.

3 Extensions and Alternatives

This protocol is optimal for requesting information from a single node, but not for requesting which nodes on the entire network can provide a service. Event-based methods, whether a capability is announced via an event or the Identify Producer and Identify Consumer mechanism is used to find which nodes can source the event that identifies the capability, may be more effective for that use case.

The information returned is intended to be considered static: A node may request it and never have to request it again, because it won't change. (For development purposes, a node might be reprogrammed, so it might be useful if configuration tools have a way to force rescan of this data)

Table of Contents

