1 Introduction

This explanatory Technical Note contains informative discussion and background for the corresponding "OpenLCB CAN Message Network Specification". This explanation is not normative in any way.

5 The protocol is described via three components: the state machine within the node(s), the messages, and the basic interactions that the nodes must take part in. These are separate described below. The messages are described in terms of a general format, and specific message definitions.

Messages are transported across a specific data-link level implementation, for example using
10 CAN frames or TCP/IP sockets. The messages are described first in general terms, then mapped to specific implementations. The states and interactions are the same across all data-link implementations.

2 Annotations to the Specification

This section provides background information on corresponding sections of the Specification
15 document. It's expected that two documents will be read together.

2.1 Introduction

OpenLCB is based on a global exchange of individual messages. This specification defines the basic messages and how they interact. Higher-level protocols are based on this message network, but are defined elsewhere.

20 This specification separately describes how messages and/or parts of messages are transported across CAN segments within OpenLCB-CAN format frames.

This specification separately describes how messages and/or parts of messages are transported across TCP/IP sockets.

2.2 Intended Use

25 The interactions described here are used by all OpenLCB nodes to connect to the OpenLCB network.

All OpenLCB protocols are built upon the exchange of messages. Higher-level protocols for exchange of data via datagrams, events and for other purposes are constructed using messages as described here.

2.3 References and Context

For more information on format and presentation, see:

- OpenLCB Common Information Technical Note

This specification is in the context of the following OpenLCB Specifications:

- The OpenLCB Node Identifier Specification, which specifies Node Identifiers and how they are defined.

This specification is in the context of the following OpenLCB-CAN Specifications:

- The OpenLCB-CAN Frame Transfer Specification, which specifies transfer of OpenLCB messages over CAN segments.

"CAN" refers to the electrical and protocol specifications as defined in ISO 11898-1:2003 and ISO 11898-2:2003 and their successors.

This specification is in the context of the following OpenLCB-TCP/IP Specifications:

- The OpenLCB-TCP/IP Segment Transfer Specification, which specifies transfer of OpenLCB messages over TCP/IP links.

Conformance with a later version of a standard shall be accepted as conformance with the referenced versions.

2.4 Message Format

OpenLCB messages are sent using the transfer mechanism and format described in the specification for a specific wire protocol.

All messages shall contain a source Node ID and a Message Type Indicator (MTI). The MTI defines both the general format of the message and its specific type. All messages with the same MTI are of the same type.

2.4.1 Message Type Indicators

(Most of this section should be moved to the end as a section 3, past the Standard annotation)

The general Message Type Indicator (MTI) is a 16-bit quantity, although it may be remapped for specific wire protocols.

Many nodes will treat MTIs as magic 16-bit numbers, just comparing them for equality to specific values of interest. That's a perfectly fine node implementation strategy.

This section describes how the numeric values for those MTIs are allocated. The current allocations are documented in a separate spreadsheet[1]. We keep them in just that one place to avoid conflicting updates. Those allocations are normative. The discussion in this section is not normative on OpenLCB users or node developers, but does describe the methods that are to be used for allocating new MTI values for new OpenLCB message and protocol types.

---

[1]See http://openlcb.org/specs/index.html for the current version of the spreadsheet. It provides concrete examples that may help you understand the material in this document.

(This is just informative, not normative; it's the actual MTI values that are normative, not how they were picked)

65   Because the MTI values are specified for each kind of message, the Standard just documents those results. This section of the Technical Note addresses the method for choosing specific values.

We've chosen to allocate MTI bit fields to make decoding simpler; if possible, aligned on nibble boundaries to make it easy to read as hexadecimal numbers. We've also used a mix of bit-fields and individual flag bits to increase compatibility when additional MTI values are defined later.

70   There are two basic approaches to identify classes of message types, such as "addressed" vs "global" messages.

1.   Use a dedicated bit field to distinguish the types, e.g. 1 indicates addressed and 0 means global.

2.   Encode in the type number, e.g. "Type A (addressed) is 1", "Type B (addressed) is 2", "Type C (global) is 3", "Type D (addressed) is 4".

75   The encoded form uses less bits, particularly if there are many classes to distinguish, which would require many dedicated bits. But future expansion is easier with dedicated bit fields, because nodes can do some limited decoding of MTIs even though the node was created before the new MTI values were defined. For example, a gateway can determine whether a message is global or addressed to a particular node, even if the specific MTI was defined after the node was built.

Field Name	Size & Location mask	Description
Reserved	2 bits 0xC000	0 (send and check)
Off-segment flag	1 bit 0x2000	0 means original-segment only, 1 should be forwarded throughout.
Dest ID flag	1 bit 0x1000	1 means Destination ID present, 0 not present
Simple Node flag	1 bit 0x0800	0 means for simple node, 1 means not
Static Priority	2 bits 0x0600	0 to 3 0 goes first, 3 last if priority processing is present
Qualifier	5 bits	See below

	0x01F0	
Event ID flag	1 bit 0x0008	1 means Event ID present, 0 not present
Modifier bits	3 bits 0x0007	Used in some MTIs; by default 0b111 sent and checked

Table 1: Common MTI Layout

80 The "Off-segment flag" bit defines whether a message on a segment, e.g. a TCP/IP backbone link between CAN gateways, should be propagated off that segment to others. A 0 means that the message should not go through gateways; a 1 indicates that it should.

The next bit, the "Dest ID flag" in the least significant bit of the top nibble, indicates whether the message carries a destination address (1) or does not carry a destination address (0). These are also
85 referred to as "addressed" or "global" messages respectively.

Global messages shall be delivered to all nodes.[2]

Addressed messages shall be delivered to the node in their destination node ID. They may, but need not, be delivered to other nodes.

The contents of the top nibble is still under development. Note that there are only four values currently
90 defined as useful: 2 and 3 for unaddressed and addressed general purpose messages that will route to any CAN network; and 0 and 1 for unaddressed and addressed messages that are not propagated to CAN networks. In the future, we may want to change this nibble from bit coding to value coding but those specific values (and very likely the Dest ID flag bit) will be preserved. Values 4-15 or the top 2 bits are not currently used and available for the future.

95 The next eight bits form a specific message type number. This has substructure:

- The top bit "Simple Node flag" is used to indicate messages meant for "simple" or minimal nodes. A 1 in this bit means that these simple nodes can ignore this message. A 0 in this bit means that the simple nodes must process the message. See the section ??? below for more information. This bit is reserved to 0 for all addressed message types, as a message specifically
100 delivered to a node should be processed.

- The next two bits are used to form static priority groups. A 0 bit is considered to have more priority (can be processed first), a 1 bit less priority (can be processed later). The MSB makes a larger statement about priority than the LSB of these two. Priority processing is permitted but not required. This is included to allow protocol designers to ensure that CAN frame reordering
105 (which, although not always present, is a normal part of CAN that must be considered) won't result in problems for communications.

- The next 5 bits, forming the low nibble, are used to indicate the specific message type within a priority group and taking into account the values of all the other bits. This is the unique part

---

5 [2]The "simple node protocol" is an exception to this, which needs to be worked into this Standard.

that's selected at the end of the process for designing a new OpenLCB protocol to ensure a
unique MTI.

The bottom nibble of the MTI is interpreted as flags that define the structure and format of the message type.

- The first bit in the nibble indicates this message carries a P/C Event ID field when set to 1. Setting 0 means that the message does not carry a P/C event ID. If a P/C Event ID is present, it is at a specific location in the message content, right after the destination address (if present) or right after the MTI (if no destination ID present).

- The three least-significant bits can be used as modifiers to the specific MTI, or (later) used to create additional MTI codes. At present, unless used as MTI modifiers, these should be sent and checked as all 1 bits, 0b111. Some MTIs have additional status bits defined as part of this field. For example, there are two status bits associated with "Consumer Identified" which must be kept in the header since there is no room in the CAN data field. These can be consider as MTI modifier bits.

This MTI organization allows nodes to do simple decoding of messages with MTIs that they don't recognize, perhaps because they were defined after the node was created. For example, gateways can use this to control routing of messages that they don't understand, perhaps because they were defined after the gateway was developed.

2.4.1.1 CAN MTI Considerations

MTI information is carried in a different format on CAN links to increase bandwidth efficiency, simplify decoding in small processors, and permit use of hardware filtering.

The standard CAN MTI field is 12 bits in the header (messages without destination address) or one byte in the data segment (addressed messages). Since CAN frames only carry 8 data bytes, a 1-byte MTI short form will be used until future expansion makes more necessary. The possibility of longer MTI values has been reserved, see below.

After mapping into one or more CAN Frames, the standard MTIs are mapped to one of eight frame types:

Frame Type	Meaning
0	Unaddressed MTI
1	(Reserved)
2	Datagram complete in frame
3	Datagram first frame
4	Datagram middle frame
5	Datagram final frame
6	Addressed MTI other than datagram or stream
7	Stream Data

Table 2: MTI Type Values

The 2, 3, 4, 5 (Datagram frames) and 7 (Stream Data) values are special cases chosen for efficient processing of large amounts of data on CAN. Most MTIs will map to 0 or 6.

2.4.1.1.1   Addressed CAN MTIs

Common MTIs with the "destination ID present" bit set are mapped to and from type 6.

In this case, the 12-bit destination alias is placed in the header, and the middle byte from the full MTI (simple node flag, type, qualifier) is the first byte of the CAN frame.

Field	CAN prefix	Frame Type	Destination ID	Source ID
Size & location (within 29-bit CAN Header)	2 bits 0x1800,0000	3 bits 0x0700,0000	12 bits 0x00FF,F000	12 bits 0x0000,0FFF
Value(s)	3	6	12-bit alias for destination node	12-bit alias for source node

Table 3: Addressed MTI CAN Header Format

This format is also used for frames made from Datagram and Stream Data messages, as described in the specifications for those protocols, with values of 2 through 5 and 7 in the Type field respectively.

2.4.1.1.2 Unaddressed CAN MTIs

Common MTIs with the "destination ID present" bit unset are mapped to and from type 0.

150   In this case, the next twelve bits of the CAN header are available for MTI information. The first (most significant) eight bits are used for the Message Number byte. The last (least significant) four bits are used for the Event ID present, the Flag Byte present flag, and any flag bits themselves.

Field	CAN prefix	Frame Type	Simple Node flag	Static Priority	Qualifier	Event ID present	Flag Bits	Source ID
Size & location (within 29-bit CAN Header)	0x1800,0000	0x0700,0000	0x0080,0000	0x0060,0000	0x001F,0000	0x0000,8000	0x0000,7000	12 bits 0x0000,0FFF
Value(s)	3	0	See Table 1 MTI description					12-bit alias for source node

Table 4: Unaddressed MTI CAN Header Format

155

Flag bit handling is used to get an extra 2 bits of data into the CAN frame, e.g. for certain messages that contain an 8-byte Event ID plus some extra information. If a common MTI has the "Flag Byte" bit set, the low order two bits are moved from the data content into the two "flag bits" in the CAN header.

2.4.1.1.3 CAN Notes

160   Note that the priority bit in the CAN frame is separate from the static priority field in the MTI format specification.

Standard-header CAN frames don't carry the bits for the CAN MTI. Both hardware and software generally carry these as 0 bits, and require that the user code check an "extended" flag to know whether it's dealing with a standard or extended frame. By ensuring that CAN header part of the MTI coding
165   can never be zero, we ensure that standard frames don't accidentally get interpreted as OpenLCB frames. (The leading priority bit's default value of 1 can't be assumed to always be present)

2.4.2 Message Content

The message content consists of:

- The source Node ID

170   • The MTI

- If flagged as present, the destination Node ID

- If flagged as present, the Event ID

- Any other content as defined for the specific message type

175 The exact format and order are defined by the specific wire protocols, but in all cases the message must be fully decodable based on the flag-bit information in the MTI.

2.5 States

The message network layer in an OpenLCB node has two states:

- Uninitialized

- Initialized

180 Nodes shall start in the Uninitialized state.

A node in the Uninitialized state may transmit an Initialization Complete message. A node in the Uninitialized state shall not transmit any other message type.

A node in the Initialized state may transmit any message type.

The Uninitialized state is only occupied when the node is first starting up. This makes for a really
185 simple state machine: Just sent Initialization Complete message first thing on coming up.

At present, there's no way to deliberately return to the Uninitialized state. No need for this has been identified.

2.6 Definition of Specific Messages

This section defines the format of common core messages. Although there is a short description of the
190 purpose of the message, this is just for identification & explanatory purposes. The meaning of the messages is defined by the interactions they appear in, which are described in later sections.

Note that Node ID in the data part of several messages is sent in full 48-bit format in all wire protocols, specifically including CAN, even if an alias or alternate form is available elsewhere in the message.

2.6.1 Initialization Complete

195 Indicates that the sending node initialization is complete, and once the message is delivered, reachable on the network.

Name	Dest ID	Event ID	Simple Node	Common MTI	CAN format	Data Content
Initialization Complete	N	N	N	0x2087	0x1808,7sss	Sending Source ID

2.6.2 Verify Node ID

Issued to determine which node(s) are present and can be reached.

Name	Dest ID	Event ID	Simple Node	Common MTI	CAN format	Data Content
Verify Node ID	N	N	Y	0x28A7	0x188A,7sss	(optional) Node ID
	Y	N	Y	0x30A0	0x1Edd,dsss 0A	(optional) Node ID

200

There are multiple forms if the Verify Node ID message.

The global (unaddressed) format may include an optional NodeID. If present, only nodes with a matching Node ID should reply. If absent, all nodes should reply.

The addressed form may include an optional NodeID. The addressed node must always reply, whether
205 or not a Node ID is carried in the data, and whether there's a match when the optional Node ID is present.

(Should that be here, or under interactions?)

2.6.3 Verified Node ID

Reply to the Verify Node ID message.

210

Name	Dest ID	Event ID	Simple Node	Common MTI	CAN format	Data Content
Verified Node ID	N	N	N	0x28B7	0x188B,7sss	Full Node ID of sending node

The node ID in the data is redundant on wire protocols that carry the full source ID, but can be very valuable for wire protocols that abbreviate ("alias") the source ID within the messages, e.g. CAN.

2.6.4 Optional Interaction Rejected

215

Name	Dest ID	Event ID	Simple Node	Common MTI	CAN format	Data Content
Optional Interaction Rejected	Y	N	N	0x30C0	0x1Edd,dsss 0C	MTI, error, optional information

The contents are, in order:

- Two bytes of MTI. If the frame transport only delivered part of the MTI[3], that content is returned with the rest of the MTI bits set to zero.

220
- Two bytes of error code.

- Any extra bytes that the node wishes to include. There can be zero or more of these. These must be described in the node documentation.

Nodes must process this message even if not all of the contents are provided.

Error codes:

225   0x1000 bit: if set, this is known to be a temporary error, and the interaction can be retried.

0x2000 bit: if set, this is known to be a permanent error, and the interaction must not be retried.

2.6.5 Terminate Due to Error

Name	Dest ID	Event ID	Simple Node	Common MTI	CAN format	Data Content
Terminate Due to Error	Y	N	N	0x30D0	0x1Edd,dsss 0D	MTI, error, optional information

The contents are, in order:

230
- Two bytes of MTI. If the frame transport only delivered part of the MTI[4], that content is returned with the rest of the MTI bits set to zero.

- Two bytes of error code.

- Any extra bytes that the node wishes to include. There can be zero or more of these. These must be described in the node documentation.

235 Nodes must process this message even if not all of the contents are provided.

---

[3]For example, CAN delivers 9 or 13 bits of the MTI via each frame.

[4]For example, CAN delivers 9 or 13 bits of the MTI via each frame.

Error codes:

0x1000 bit: if set, this is known to be a temporary error, and the interaction can be retried.

0x2000 bit: if set, this is known to be a permanent error, and the interaction must not be retried.

2.7 Interactions

240 All nodes must be able to take part in all standard interactions.

2.7.1 Node Initialization

Newly functional nodes, once their start-up is complete and they are fully operational, shall send an "Initialization Complete" message and enter Initialized state.

- There is no guarantee that any other node is listening for this. No reply is possible.

245 • Nodes must not emit any other OpenLCB message before the "Initialization Complete" message.

Sending the IC message is required to insure that higher-level tools are notified that they may start to work with the node.

2.7.2 Node ID Detection

250 Upon receipt of a Verify Node ID Number message addressed to it, or an unaddressed Verify Node ID Number message, a node will reply with an unaddressed Verified Node ID Number.

This can be used as check that a specific node is still reachable. When wire protocols compress the originating and/or destination NID, this can be used to obtain the full NID.

The standard Verify Node ID Number interaction can be used to get the full 48-bit NID from a node for
255 translation. At power up each node must obtain a alias that is locally unique. Gateways will also have to obtain unique aliases for remote nodes they are proxying on to the segment.

2.7.2.1 Example: Node obtaining local alias from full node ID

For wire protocols like CAN, which require short node aliases to send messages, Verify Node ID can be used to get the CAN alias from a known node ID by sending the global form with the full node ID in
260 the data. Only the desired node will reply, and it's alias will be in the source ID part of the message.

2.7.2.2 Example: Finding all nodes

Send the global form with no node ID

2.7.2.3 Example: Confirming that a specific node can still be reached

If you have the necessary information (e.g. alia) use the addressed form, as it's less load on the entire
265 system. Otherwise use the global form with the Node ID in the data.

2.7.3 Error Handling

There are multiple mandatory error-handling scenarios defined.

(Need to explain "optional" here)

2.7.3.1 Reject Addressed Optional Interaction

270 • Node A receives an addressed message from Node B that carries Node A's NID.

• The MTI indicates the start of an optional interaction.

• If Node A does not want to take part in the optional interaction, it may send an Optional Interaction Rejected message addressed to Node B with the original MTI in the message content. There is no requirement that OIR be sent; the node may silently ignore the incoming
275 message.

(The message content also contains an optional reason code and an optional data value. (Define use))

(This is written that sending the OIR in return is optional. This greatly increases the complexity of error handling on the originating node, though, as it can't assume that lost messages are a transient error. It would be better to have this be a mandatory response to simplify that. Does it add much
280 complexity cost to the nodes?)

(The phrasing is also in terms of an "optional interaction", which doesn't cover the case of "undefined interaction", e.g. an MTI that's not allocated. Since any given node doesn't actually know whether an MTI has been allocated or not (it might have been built a while ago, with new optional protocols added since then), this should be rephrased. I'll add a to-do item for that too.)

285 #### 2.7.3.2 Reject Unaddressed Optional Interaction

• Node A receives an unaddressed message from Node B.

• The MTI indicates the start of an optional interaction.

If Node A does not want to take part in the optional interaction, it silently drops the message without reply.

290 #### 2.7.3.3 Reject Addressed Standard Interaction Due to Error

• Node A is taking part in an addressed interaction with Node B. Either node may be able to send the next message.

• Some error condition prevents Node A from continuing the interaction.

• To terminate the interaction, Node A sends a Terminate Due to Error message to Node B. It then
295 resets it's state so as to no longer be taking part in the addressed interaction.

The message content contains the most recent MTI received in this interaction, a mandatory reason code and an optional data value. The use of these fields is to be defined, but reserved space is to be transmitted, so we specify the bytes.

300 Note that the specification doesn't say whether Node A or Node B started the interaction. It could have been either. Node A is just the name for the node that can't continue and wants to stop the interaction.

This is a very coarse mechanism that is meant to handle rare events that should not routinely occur. The "most recent MTI received" values is not always sufficient to determine which interaction is being referred to. Higher level protocols should define more focused and reliable mechanisms if they are likely to encounter errors.

305 Nodes must handle messages of this type that arrive without MTIs and error code information.

Needs definition of a permanent vs temporary bit in the error code information (consider choosing bits in a similar way to the datagram definitions)

(add an example e.g. two PIP requests which the node can't handle, please retry) (Make it clear when this option is not available in the protocol docs; assume it is if not otherwise indicated; simplifies little
310 nodes)

2.7.4 Duplicate Node ID Discovery

OpenLCB nodes must have unique node IDs. The Frame Transfer protocol will detect duplicate node IDs on a single CAN OpenLCB segment, e.g. a single CAN bus, but is not intended to detect duplicate node IDs across multiple segments. To detect duplicates across the entire connected OpenLCB, all
315 OpenLCB nodes must indicate an error if they detect an incoming message with a Source Node ID equal to their own. If possible, they should indicate it at the board itself using a light or similar. If possible, they should emit a PCER message with the "Duplicate Source ID detected" global event, which will carry the duplicate node ID in the Source Node ID field. (But how can they "must" if those are "if possible"?)

320 After sending the "Duplicate Source ID detected" global event, the node should not transmit any further "Duplicate Source ID detected" messages until reset because this message will be received at the other duplicate-ID node(s), resulting in additional "Duplicate Source ID detected" global events and causing a possible message loop. (Optionally, could allow to send again after e.g. 5 seconds)

Yes, this is level jumping, but it's the best way to do it within the existing structure.

325 To further improve the reliability of this detection, OpenLCB nodes should, but need not, emit a Verified Node ID message every 30 to 90 seconds. As an implementation detail, it's recommended that CAN-attached nodes use their NIDa to pick that interval so that messages don't bunch up.

2.8 Delays and Timeouts

Nodes shall send messages required by OpenLCB protocols within 750 msec unless otherwise
330 indicated in the documentation for the specific protocol interaction.

Nodes may, but are not required to, use a timeout mechanism to protect against messages lost due to malfunctions. Such a timeout shall not be shorter than 3 seconds.

Needs a discussion of handling failed communications. The protocols are all designed to have error-responses (error reply to datagram; Option Interaction Rejected) instead of silently failing for directed
335 messages. Timeout logic is only necessary to handle transport failures and failures of nodes to execute the protocols, including e.g. when they power off in the middle of something. The Standard should say

that "Nodes shall allow at least nnn msec for a reply to be received to their communications" and "Nodes shall reply to messages within mmm msec". Then this TN can talk about timeouts, how you have to assume that you'll have to wait for a reply when arranging state machines and buffering, and
340   the generic issues around recovering from an expected reply not being received. Any specific issues with timeout recovery can be discussed in the messge-specific parts of the TN.

3 Simple Node Protocol Subset

OpenLCB uses the "Simple Node Protocol" concept to distinguish a subset of global message types that are never needed by certain "simple" nodes. They can then be rapidly ignored by those nodes,
345   gateways can filter them out, etc. This note describes the Simple Node Protocol. It is not normative on node implementors.

Simple nodes are defined as the leaf nodes that do basic layout control (input/output) operations. These nodes need to be able to[5]

   • indicate their presence (by sending Initialization Complete messages and replying to Verify
350   Node messages)

   • send and receive event reports (through PCER messages and associated reports)

   • be configured (through related messages, datagrams and in some cases streams)

and perhaps other things in the future. Those messages form the Simple Node Protocol subset of the full OpenLCB protocols. Note that this is an asymmetric subset: Simple nodes can send some types of
355   messages and receive others.

On the other hand, gateways, configuration tools and other network-aware nodes are not simple nodes. These nodes need access to the full OpenLCB protocol so they can

   • learn about the appearance of other nodes (by receiving Initialization Complete messages)

   • learn about other producers and consumers (by receiving status messages)

360   and similar. To do this, they must be able to send and receive every message type.

For ease of filtering, a specific bit in the MTI identifies the global messages needed by simple nodes. This bit allows OpenLCB to define new MTIs in the future and still include them in the "simple node protocol" subset or outside it without having to modify existing nodes. See the MTI allocation TN for more information on this.

365   Gateways that are serving network segments (e.g. single CAN segments) that contain only simple nodes may suppress unaddressed (global) messages that do not contain simple-node MTIs.

3.1 Protocol Description

Operationally, the simple node protocol is defined by the MTIs that carry a set Simple Node bit, plus all addressed messages. This section summarize received transmitted messages, and describes the
370   reasoning behind those choices in the current MTI definitions.

---

[5]Simple nodes also have to do whatever is needed to function with their specific wire protocol, e.g. send CID/RID frames on CAN.

3.1.1 Messages Transmitted

Simple nodes may transmit any message, which must propagate correctly.

3.1.2 Messages Received

375 Simple nodes must receive any message specifically addressed to them, plus the following unaddressed global messages:

- Verify Node ID – They need to receive this so that they can reply to it.

- Verified Node ID – They need to receive this because it's the reply to their own request, which might be used to e.g. locate a node for delayed sending of status

- Protocol Support Inquiry – They need to receive this so that they can reply to it.

380
- Identify Consumers, Identify Producers, Identify Events – because others will ask this of them

- Learn Event – so they can be programmed

- P/C Event Report – what they do for a living

In the future, additional MTIs will be defined. If simple nodes need to received them, the MTI will indicate that; see the MTI Allocation TN.

385 ### 3.1.3 Messages Not Received

Messages not listed in the section above do not need to be received by simple nodes.

A brief description of why the following message types are not necessary for simple nodes:

- Initialization Complete: Used to indicate that a node is newly available to the network. Simple nodes only care about their specific tasks, and by definition are not interested in the overall
390 structure and availability of the network.

- Consumer Identified, Consumer Identify Range: These are of interest to gateways and configuration tools, but an individual producer does not need to know which (if any) nodes are consuming its produced events.

- Producer Identified, Producer Identify Range: These are of interest to gateways and
395 configuration tools, but an individual consumer does not need to know which (if any) nodes are producing its consumed events.

4 Expansion

4.1 Longer MTIs

On the expansion past 16 bit MTIs both for global and addressed messages. Easy to take up another
400 byte, harder to get optimal use out of it. Expansion needs to be indicated by something that's in the CAN section, so use all zero.

4.2 Longer Messages

Longer than 8-byte global, 7-byte addressed messages don't fit in a single frame. Discuss mechanism. Includes issues of guaranteed place for global to land in a node, e.g. memory management. (Spare frame type; request resend w/o buffering, but how long to hang on to original?)

405

PIP reply uses "break into N frames with MTI at start": One message becomes N smaller ones. But it's a directed reply, so the requestor can ensure that the response will be processed on receipt. Label end with a non-full CAN frame (though Io code sends those in the middle), if needed a zero-length frame.

Table of Contents

410