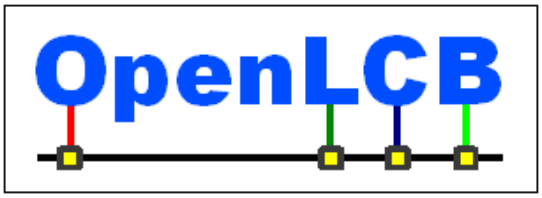

OpenLCB Technical Note	
Abbreviated Configuration Description Information	
Jul 12, 2012	Preliminary

1 Introduction

Consider a "network browser" that can show more specific information on the nodes, e.g. their type information, user-given names, etc. To do that, the browser could

- Read the configuration description information (CDI)
- Parse out the first section(s)
- Do memory-access configuration reads to get the information.

That last step is very fast. The browser could do the necessary configuration reads in a small fraction of a second, at least on an idle CAN segment.

But the first two steps are not-so-fast. The CDI is a very powerful and general method to describe all about the node, but it's a handful to read & parse when you just want to get some specific information.

This Standard provides an optional protocol that allows a node implementor to say "I keep my basic information in a common place", so that e.g. browser tools can pick that up quickly.

There are several parts to this:

1) A standardized set of basic information & its location in configuration space (called Abbreviated Configuration Description Information ASCI; the ACDI acronym is confusingly like CDI, and perhaps a better one is needed). This allows the information to be rapidly retrieved via the Memory Configuration Protocol without first reading the entire CDI.
2) A simple CAN protocol for returning it: Simple Node Information Protocol SNIP, documented elsewhere.

Why the dedicated simple protocol? To make it even easier and faster to be retrieving this from a lot of nodes at once. Memory configuration is powerful and robust for doing complex configuration, particularly for writing to nodes, but it's a little heavy-weight when doing optional reads like for this purpose. The special protocol, which doesn't involve the buffering and acknowledgements of datagrams used for Memory Configuration, simplifies start-up in large networks, and makes it possible for low-capability nodes to gather basic information about the entire network, even from nodes that don't implement the full datagram-based Memory Configuration Protocol.

These can be independently provided. ACDI can exist without SNIP; once PIP has told you ACDI is present, you know the format of (some of) the CDI-described information and can use Memory Config to retrieve the basic node data quickly without reading the full CDI. SNIP can work without ACDI, with the content being stored at some undefined place in memory; SNIP is ideal for nodes that don't have a full implementation of Memory Configuration Protocol, at either end of the connection. But they work better together.

JMRI really likes the SNIP Request/Reply, because it's really fast: One request queues up retrieval of the entire thing, without having to deal with datagrams. JMRI is a big boy and can handle the large reply volume. But the config read mechanism (via individual datagrams) might be better for small nodes making this request that already have the code in place to handle those.

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

Must appear in PIP.

3 XML specification

The <acdi fixed="1" var="1"> element in the CDI XML file indicates that the ACDI is present. The two attributes indicate the version numbers of the fixed and variable information. Only "1" is defines as valid at this time; if the ACDI hasn't been read as part of reading CDI, version "1" must be assumed for both.

Use of the <acdi> element is recommended, but not required. It saves space and transfer time. It's also permissible for the CDI to indicate the equivalent content, which is:

```
<segment origin='0' space='252'>
  <group>
    <name>Manufacturer Information</name>
    <description>Manufacturer-provided fixed node description</description>
    <int size='1'>
      <name>Version</name>
    </int>
    <string size='22'>
      <name>Manufacturer Name</name>
    </string>
    <string size='21'>
```



```
            <name>Node Type</name>
        </string>
        <string size='10'>
            <name>Hardware Version</name>
        </string>
        <string size='10'>
            <name>Software Version</name>
        </string>
    </group>
</segment>

<segment origin='0' space='251'>
    <group>
        <name>User Identification</name>
        <description>Lets the user add his own description</description>
        <int size='1'>
            <name>Version</name>
        </int>
        <string size='21'>
            <name>Node Name</name>
        </string>
        <string size='42'>
            <name>Node Description</name>
        </string>
    </group>
</segment>
```

(the spaces are 0xFC and 0xFB respectively)

The name elements are not required, but the segment and string elements must be arranged so that the memory arrangement is identical.

4 Stuff to be merged into the above

How it works

1) A node denotes that it has this capability using a specific bit in the Protocol Identification Protocol (1 CAN exchange to check that)

2) If it does, the basic information can be found in a specific place in the configuration space (specific, well-known address space, location, and format), so can be directly read via configuration reads (datagram access, with small CAN overhead)

For this to work, there needs to be a common content (so that we don't need another layer of CDI-like stuff to define this content too). Hence this note. What should be in that content?

Some fixed info:

*) A version number, so we can extend this later: Initially 1.
*) A manufacturer name
*) A manufacturer-issued node type name
*) Hardware version
*) Software version

Some variable info, which the user can change:

*) A version number, so we can extend this later: Initially 1.
*) Name
*) Description

These variable items don't have to be provided, but if they are, they can be just the usual identification items that are accessible by the full configuration protocol, configured by the user with the full configuration protocol, but then made available for a quick read this way.

We want to keep this somewhat compact, with the most useful (in the sense of "people will want to see this quickly") info at the front. Reading a full datagram (64 bytes memory-read payload) would let us sample about 50 nodes in a second (idle bus), which is fine. Another factor of 4 larger might start to seem slow to the user, though. The whole idea is to be able to get a view of even large networks faster than a user can browse through the info on an screen. The user can then select a small number of specific nodes that are interesting, pull the full CDI and work with the entire configuration information of those nodes.

Reading an icon (e.g. image of the board) takes longer, and is therefore less vital. That's also not defined in this version of the document. Lives in a 3rd address space (0xFA), which doesn't get any special consideration in the Memory Configuration Protocol.

Content

Spaces chosen to be right below CDI, so they leave as much contiguous as possible.

Up to designer to decide on size (0 bytes, small, large) of icon information.

The memory access in those three spaces doesn't mean that is has to be stored that way.

The use of zero-terminated strings within fixed size slots.

User comment provides a longer field, not computer parsed; node name is a short thing for e.g. headings, etc.

Two spaces provided for simplicity in implementation, as we imagine that some will be from a permanent memory and some will be user configurable.

Memory-access has some special support for these spaces in the “Get Configuration Options” command, so that you don't have to poll the spaces individually. But you don't really need to do that anyway, as (1) they have to be there once the ACDI bit is set in PIP and (2) you'll just get a zero-length reply if they're not.

Chosen to fix in a single 64-byte read operation from each of the spaces.

Strings just concatenate with null termination. A missing value is then just a null byte.

Protocol ID protocol specifies whether this is present, so no more confirmation information is needed.

Later versions with higher ACDI version numbers could contain more fields, result in more than one datagram in reply, etc. But for expansion purposes, they should be proper supersets of this.

All these work together, and have to be consistently done:

- ACDI is marked as present in PIP (requires memory-access, CDI be set too)
- Memory config has the two spaces available
- CDI carries the <acdi> element

4.1 Notes

The definition of the manufacturer-provided information is intended to be vague and hand wavy, based on the discussions in the NMRA DCC group. Manufacturers _don't_ want to be told what to call their nodes. They want a place for a "Manufacturer-issued node type" where they can put whatever "node type" name they want to. No more normalization of that content than that is likely to succeed. Certainly not model number; most aren't numbers but some are. Some DCC decoders don't even have model names.

Basic information does not need to be present if this protocol is not marked; store anything you like.

Table of Contents

1 Introduction..... 1

2 Annotations to the Standard..... 2

2.1 Introduction..... 2
2.2 Intended Use..... 2
2.3 Reference and Context..... 2
3 XML specification..... 2
4 Stuff to be merged into the above..... 3
4.1 Notes..... 5

