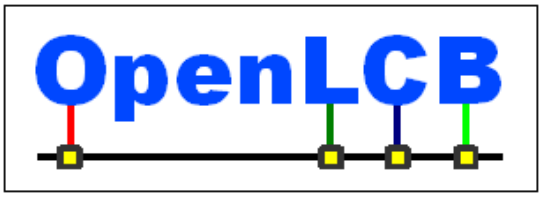

OpenLCB Technical Note	
Function Description Information	
July 22, 2024~~Jul 18, 2024~~	~~Preliminary~~ Adopted

1 Introduction

"~~Configuration~~Function Ddescription Iinformation" (FDI) in this context refers to *fixed* information available from an OpenLCB Train ~~device~~Node, via OpenLCB, so that other devices can properly and correctly ~~configure~~operate it. ~~The information is fixed, so that it can be pre-compressed, stored in the device, and just supplied when needed with minimal work on the part of the device and the device's developers. This means that e.g. the actual current configuration contents are not available as part of the CDIFDI, as that is variable information. Similarly, the CDIFDI cannot contain e.g. a serial number as that would require different CDIFDI contents in each node of a single type.~~

Other information may be available via e.g. manuals or the Internet, ~~and there may be pointers to that information in the CDIFDI,~~ but the format of that information is not under specification here.

A key use for ~~CDI~~FDI is to enable an ~~Configuration Tool (CT)~~ OpenLCB throttle to ~~know how to configure the node.~~determiine the features provided by a Train Node's functions. The ~~configuration tool~~throttle will use the ~~CDI~~FDI information to render some form of suitable Graphical User Interface to allow the user to easily and intuitively operate the functions provided by the Train Node

~~configure all aspects of the node's capabilities.~~ An important design choice was to embed the ~~CDI~~FDI into each node so that the system has all it needs to ~~configure~~operate the Train N~~n~~ode without having to source the information externally to the OpenLCB network from the manufacturer or some other on-line repository via the Internet or a CD/DVD etc. An OpenLCB throttle is not required to have an Internet connection.~~While the CT is likely to be a program running on a PC, it could be a hand-held device~~throttle like mobile phone or PDA or even a custom-built device.

1.1 Requirements

- Train Nodes must carry enough context that a stand-alone throttle can provide a useful human interface without getting any data from an external source, e.g. needing an Internet download or the user entering or uploading a database.

1.2 Preferences

- ~~Small~~Train Nodes shouldn't need a lot of processing power, e.g. to compress or decompress data in real time. Memory usage should also be limited, but is a second priority.
- The necessary OpenLCB operations should be state-less and idempotent to simplify software at both ends.

2 Annotations to the Standard

2.1 Introduction

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

The concept and interaction are using the same pattern and similar format as the Configuration Description Information, purposefully, with the expectation that OpenLCB devices will be able to share the software implementation between these two use-cases.

2.2 Intended Use

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

The most important use-case of operating a model train is controlling speed and functions. The protocol for these are defined in the OpenLCB Train Control protocol Standard. The user typically uses a Throttle for performing these operations, either as a special-purpose hardware device, or as a software function on a generic computing device.

While the semantic meaning of speed control is very well defined, the meaning of functions is left very vague by the OpenLCB Train Control standard, just like other standards of the industry; a function is some form of remote controlled feature of the train (or locomotive). This vague definition is intentional, as manufacturers have come up with new and innovative ideas on what kind of additional features they want to add to their models, and the typical way to operate these features is using the function abstraction.

Typical functions are lighting functions (like headlights, tail lights, class marker lights, etc.), motor control functions (momentum, braking, etc.), sound functions (horn, bell, whistle, engine sounds, compressor, etc.) and a variety of other functions, such as a smoke unit, raising and lowering a pantograph or operating a locomotive-mounted remote uncoupler. There is no definitive list of all possible functions. The TN-218 document by RailCommunity lists over 120 different types.

Not only is the list of possibilities is very large, one specific model alone can carry dozens of different functions. Users of proprietary throttles have to activate these typically by their function number, for example using a numeric keypad. This is easy to operate, provided that the user has memorized which function number does what on that specific locomotive. With dozens of functions per locomotive and up to hundreds of locomotives on a big layout, this is no easy task. As such, it is an important use-case for the OpenLCB Throttle to display a user interface that provides information to the user about what function numbers are available, and which numbers perform what operations on the train.

The Function Description Information (FDI) provides exactly this metadata for the throttle: the list of functions, for each function the number at which it is accessible, and a textual description of what the given function does.

In addition to the description for the user, the throttle user interface needs some additional metadata to determine how the particular function button should behave. There are two common behaviors for on/off functions: *momentary* functions are turned on while the user is holding the button, and turned off upon release. *Toggle* (or binary) functions are turned on with a press of the button, then they stay on after release, until the user presses the button again. A typical momentary function is horn, a typical toggle function is headlight. This is purely a behavior of the user interface, as the underlying protocol for turning the function on and off is the same for both cases. *Analog* functions, such as a horn volume or pitch, take a numerical value to control their operation.

2.3 Reference and Context

~~See~~ The Memory Configuration Protocol~~. That's one use for the CDIFDI, and~~ is one way to retrieve the~~it~~ FDI, but ~~CDI~~FDI is independent of that. Download of this data via a direct BLE transfer offering a similar capability would make this specification equally applicable.

~~CiA 306 "Electronic data sheet specification" describes the CANopen version of a similar capability.~~

2.4 Content

The function description information for a Train Node is invariant while the node has any OpenLCB connections in the Initialized state.

The Train Node's transition back to Initialized state via sending an Initialization Complete global message tells other nodes to flush their caches and pick up any changed content.

There are no default values that e.g. associate "Bell" with a particular location or function. These are thought to be too brittle, and there are just too many possibilities to be useful.

2.5 Format

The structure is somewhat similar to the Configuration Definition Information structure, with named sections that organize and describe individual items.

The use of XML 1.0 format is intended to make the content as easy as possible to parse.

The use of XML Schema 1.0 is intended to allow the use of common XML parsers. Note that the schema definition file includes an XML Schema 1.1 check for extensibility that has been commented out so that the operational schema remains valid Schema 1.0.

2.5.1 XML Elements

Note that nothing in the Standard prevents providing an xml-stylesheet instruction, and it's common to provide one of the form:

```
<?xml-stylesheet type='text/xsl' href='xslt/fdi.xsl'?>
```

2.5.1.1 <fdi> Element

The <fdi> element forms the root of the document. It provides the “xmlns:xsi” and “xsi:noNamespaceSchemaLocation” attributes that define the schema for automated processing.

2.5.1.2 Element

The element is provided for future extensibility. The "space" and "origin" attributes are carried over from the Configuration Description Information format, but at the present state of the standard, there is no way to express a different memory space than 0xF9. Similarly, the layout of offsets in the memory space is not defined today. If a future version of this standard provides such capabilities, then the "space" and "origin" attributes may be used in a backwards-compatible way, with 0xF9 and 0 being their respective defaults. They should be omitted in current implementations.

2.5.1.3 <group> Element

Groups may contain one or more inner groups, with the same representation. This may continue to any desired level.

Note that unlike in the CDI standard, groups can not be specified with a repetition count in the FDI. The reason for this is that individual functions are addressed by the <number> sub-element, and repeating a group is not meaningful, because that sub-element would remain the same.

2.5.1.4 <function> Element

The “kind” attribute is an enumerated type of (currently) "binary", “momentary”, and "analog".

The “number” element indicates which function is being described by this element. This value is to be used in the Train Control Protocol Set Function operation in the "address" field. That field is 24 bits, therefore the maximum value in this element is 16777215.

Note that there is no requirement on the ordering of the <function> elements with respect to their <number>. It is not required for functions to be listed in ascending order by number, and it is not required for every function number in a range to be present (i.e., numbers can be skipped). A function number not listed in the FDI is assumed by throttles to be doing nothing (i.e., not presented on a UI having a list of functions).

The “name” element provides information which can be used if the throttle has e.g. a graphical interface to the user.

The “size” attribute is optionally allowed for compatibility with some early, pre-Standard implementations. It should be omitted in current and future implementations.

~~3~~ ~~Stuff to be merged into the above~~Additional Information

~~4~~

~~[5] Environment of Proposal~~

~~*[5.1] Requirements*~~

- ~~Nodes must carry enough context that a stand-alone configuration tool can provide a useful human interface without getting any data from an external source, e.g. needing an Internet download to handle a new node type.~~
- ~~It must be possible to configure a node entirely over the OpenLCB, without physical interactions, e.g. pushing buttons.~~

~~4.1.1 [5.1.1] Preferences~~

- ~~Small nodes shouldn't need a lot of processing power, e.g. to compress or decompress data in real time. Memory usage should also be limited, but is a second priority.~~
- ~~Configuration operations should be state-less and idempotent to simplify software at both ends.~~
- ~~Multiple independent configuration operations can proceed at the same time. Specifically, multiple devices should be able to retrieve correct configuration description information at the same time.~~

~~Design Points~~

4.2 [5.2] Design points

Basic configuration is done with the configuration protocol defined elsewhere.

~~The "Variables" described here are not exactly the same thing as "Configuration Variables" (CVs) or "Node Variables" (NVs) that are discussed elsewhere. Those are aimed at storage, and so are grouped by address. The "Variables" here are grouped by function. "Long address" might be several CVs, but would be one variable to this proposal. Similarly, CV29 has lots of variables within it, each stored as bits. Perhaps it would be better to use a different name here, such as "Setting" or "Option"?~~

~~*Proposal*~~

~~Definition~~

~~2) The primary design constraints are complexity and size in the OpenLCB device providing CDI, and complexity and size in the device consuming the CDI.~~

~~2A) Size and complexity in the providing device is the more important constraint. There are more of those devices, they are cost sensitive, and they may not be upgradable once delivered.~~

~~2B) Size and complexity in the CDI-consuming device should also be considered. In particular, code complexity is an issue which must be addressed.~~

~~3) Secondary constraints are testability of the provided information, scalability of the format, and the convenience and availability of a suitable toolchain.~~

~~4) There is a physical/logical structure to the configuration which the CDI can and should reflect:~~

~~4A) The basic OpenLCB unit is a "Node". Nodes provide CDI for their needed configuration information. The protocol for that will be defined elsewhere/elsewhen.~~

~~4B) A Node can contain zero or more "Producers". Each Producer is independently configured. There is no ordering between separate Producers, but they can be numbered for ease of reference.~~

~~4C) A Node can contain zero or more "Consumers". Each Consumer is independently configured. There is no ordering between separate Consumers, or between individual Consumers and Producers, but they can be numbered for ease of reference.~~

~~4D) Each Producer or Consumer can be configured with zero or one Events.~~

~~4E) Each Event has an Identifier which uniquely defines it. An event may optionally carry additional data.~~

~~4F) To ensure future growth, there is no required "device", "channel" or other grouping within a node. Those may be present in some node types, and CDI must be able to represent them, but may not require any specific organization.~~Function values are stored- in the 0xF9 memory space. They can be addressed through the Train Control protocol's Set Function and Query Function commands. The Memory Configuration Protocol can also be used to read and write the memory space 0xF9, if that is implemented. This allows reading a large number of functions with a single request. However, while the Train Control Protocol Set Function has 16 bits available to represent a function's value, in the memory space 0xF9 there is only 1 byte per function address; the default value of deprecated attribute 'size' being 1 represents this. There is no resolution for this discrepancy in the current Standard.
~~"Function Definition Information", similar in intent to Configuration Definition Information (CDI) is stored in XML format in address space 0xFA to provide user-oriented context. That includes:~~

- ~~Memory layout of the function values, allowing for multiple data types from binary (one and off for lights) through integer values (for e.g. sound intensities) and strings (sign displays?).~~
- ~~Function naming, so that a throttle can display useful names to the user such as "Bell", "Coupler Clank" and "Master Volume". This includes internationalization of those labels.~~

- What else needs to be conveyed? "Make this prominent on the throttle"? "Have this there, just a little less prominent"? "Seriously, nobody cares about this option, bury it"?
- At present, there are no default values that e.g. associate "Bell" with a particular location or function. These are thought to be too brittle, and there are just too many possibilities to be useful (see the unscientific and incomplete Survey of existing function names).

4.3 Sample Ffunction Ddefinition XML:

[5.3]

```
<?xml version='1.0' encoding='UTF-8'?>
<?xml-stylesheet type='text/xsl' href='xslt/fdi.xsl'?>
<fdi xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
xsi:noNamespaceSchemaLocation='https://openlcb.org/schema/fdi/1/0/fdi.xsdhttp://openlcb.org/trunk/prototypes/xml/schema/fdi.xsd'>

<segment origin='0' space='250'>
  <group>
    <name>Lighting Functions</name>
    <description>Headlights and Marker lights and Running lights, oh my.</description>
    <function offset="5" size ="1" kind="binary">
      <name>Headlights</name>
      <number>0</number>
    </function>
    <function size ="1" >
      <name>Ditch Lights</name>
      <number>4</number>
    </function>
  </group>
  <group>
    <name>Sound Functions</name>
    <description>Toot toot!.</description>
    <function size ="2" kind="analog">
      <name>Horn</name>
      <number>8</number>
      <min>0</min>
      <max>2</max>
    </function>
  </group>
</segment>
</fdi>
```


The structure is similar to CDI, with named sections that give the general location of the storage.
"number" is a hint that can be used to locate the function on a numbered GUI element (e.g. button);
"name" is the same if the GUI element can have a readable label.
"kind" is an enumerated type of (initially) "binary" and "analog". These are done as specific elements in CDI, and perhaps that's a better approach.

4.4 Sample Older Implementation

A sample older implementation of the DCC F0-F28 functions, ~~as described on the Function Control page~~with some optional attributes still in place:

```
<?xml version='1.0' encoding='UTF-8'?>
<?xml-stylesheet type='text/xsl' href='xslt/fdi.xsl'?>
<fdi xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
xsi:noNamespaceSchemaLocation='https://openlcb.org/schema/fdi/1/0/fdi.xsd~~http://openlcb.org/trunk/prototypes/xml/schema/fdi.xsd~~'>
<segment origin='0' space='~~250~~249'>
  <group>
    <name>F0-F28</name>
    <description>Standard DCC functions</description>
    <function size="1" kind="binary">
      <name>Headlight</name>
      <number>0</number>
    </function>
    <function size="1">
      <number>1</number>
    </function>
    ...
    <function size="1">
      <number>28</number>
    </function>
  </group>
</segment>
</fdi>
```

~~4.4.1 [5.3.1] Storage~~

~~**THIS IS COPIED FROM THE CDI CURRENTLY, THIS NEEDS TO BE UPDATED TO BE FDI SPECIFIC.....**~~

~~The configuration definition is stored in a hierarchical manner.~~

~~I) In what follows:~~

~~A "String" must be present; an "Optional String" does not have to be. Strings are in UTF-8.~~

~~An "Integer" may be signed; if no sign, it's taken as positive.~~

~~A "Map" provides a set of named descriptive values. It contains:~~

~~Name: Optional String, if present required to be unique within enclosing group or node~~

Description: Optional String
1 or more "Key", "Value" pairs. Each element of the pair can be of any supported type, depending only on how it is to be used.

Map elements provide a mapping between the pairs they contain. For example, a map can relate numeric values for a variable to description strings. A map can also be used to provide free-form documentation when neither the key nor the value are specified in advance. It may be useful in the future to specify how maps can be defined at the group level to reduce duplication. Having the possibility of a “Name” is meant to ease that future effort.

II) At the top, root level is the information for a "node". This includes:

Manufacturer: String
Descriptive Map: May contain “Model”, “Version”, “URL” and “Description” keys, along with any others desired.

Model: If present, the human-readable model name the manufacturer gives to this node.
Version: If present, the human-readable version string for the current board.
Description: Optional String
URL: If present, a URL for more information. No specific content is expected at the URL; If desired, that can be dealt with in a different specification.

Any other information desired can be added via additional keys.

III) Within the node is zero or more "groups". Each group contains:

Name: String, required to be unique within enclosing group or node
Descriptive Map: Map of documentation information; the “Description” key is the basic item.
Replication count: Integer >= 1 (number of times this group is replicated within the parent item)

A group with a replication count > 1 (called a replicated group) can be used to represent a type of replicated device. For example, a node with 4 identical input devices and 6 identical output devices can be compactly described by two groups, with replication counts of 4 and 6 respectively.

Individual groups within replicated groups are numbered from 1 to the replication count. If more than one replicated group is present, the numbering for each starts again with 1.

Groups may contain one or more inner groups, with the same representation. This may continue to any desired level.

IV) Groups may contain "variable", "producer" and/or "consumer" descriptions.

IV-a) A "variable" description contains:

Name: String, required to be unique within enclosing group or node

Type: Exactly one of "boolean", "digit" (an unsigned binary-coded-decimal value), "signed" (a binary value with a sign), "unsigned" (a binary value without a sign), "string" (a UTF-8 string, not-null terminated), or "blob" (arbitrary byte vector).
Max: Integer - For string and blob variables, the maximum number of bytes that can be stored. For digit, signed and unsigned types, the maximum value allowed.
Min: Integer - For digit, signed and unsigned values, the minimum value allowed.
Description: Optional String
Default: value of this Type, required
Offset: Optional integer offset with in the configuration address space for this item; if not present, data is laid out by length in depth-first order.

A variable may contain zero or one map descriptions. If present, the map represents a mapping between possible values (the "Key" part of the map's pairs) and convenient names for them (the "Value" part of the map's pairs).

Note that the current value of a variable is not considered configuration definition information (see item 1A and 1B in the introduction).

Configuration information must not be packed into variables; each variable must represent one type of information. In particular, the use of individual bits within larger values to pack multiple pieces of information is forbidden; those must be represented as individual variables. (How the information is stored internally is up to the designer of the specific device, and is not restricted; this requirement is about access to the information, not about how it's laid out in physical memory)

IV-b) A "producer" description contains:

Name: String, required to be unique within enclosing group or node
Description: Optional String
Replication count: Integer >= 1 (number of times this producer definition is replicated within the parent item)
Offset: Optional integer offset with in the configuration address space for this item; if not present, data is laid out by length in depth-first order.

A producer description may contain zero or more variable descriptions for any variables that configure details of the producer's function.

IV-c) A "consumer" description contains:

Name: String, required to be unique within enclosing group or node
Description: Optional String
Replication count: Integer >= 1 (number of times this consumer definition is replicated within the parent item)
Offset: Optional integer offset with in the configuration address space for this item; if not present, data is laid out by length in depth-first order.

~~A consumer description may contain zero or more variable descriptions for any variables that configure details of the consumer function. This may include e.g. variables that define how any content in incoming messages will be used.~~

[6] ~~Serialization~~Possible Future Work

This section discusses some areas where future work may be desired. It is meant as a record of the discussions to date, not as defining any specific solution(s).

4.5 [6.1] Data Compression

Some work has gone into XML content compression for the Configuration Description Information (CDI) data. See the CDI Technical Note for various considerations that have been discussed. A solution there, if developed, standardized and deployed, could also be used here.

The function definition information is expected to be significantly smaller than a Train Node's CDI, so there may be less need to compress the FDI. On the other hand, CDI gets retrieved relatively rarely (only when a user intends to modify configuration), whereas FDI gets retrieved regularly as part of normal operations of a model railroad.~~The logical layout in the previous section has to be converted to some serial set of bytes for transfer and storage.~~

~~The primary format is straight-forward XML. (Link to schema)~~

~~Another is compressed binary information. Either an aggressive compression algorithm, or some context-aware compression, can be used. Size and ease of expansion are the key criteria; ease of compression is much less important. XSLT can do some of the reformatting between compact and readable form~~

~~We need to pick one format for a lingua franca.~~

~~*4.6 [6.2] Example*~~

~~The following is not meant to show how configuration definition information would be stored, but what kinds of information would be stored. It's a description of a complex accessory decoder, the Digitrax DS54, modified for use in a Producer-Consumer model.~~

~~Hopefully the syntax will be self-explanatory. In any case, it's just for this example, not a proposal of any kind.~~

~~Manufacturer (String): Digitrax~~
~~Model (String): DS54~~
~~Version (String): 2.33~~
~~Description (Optional String): For more information, see http://digitrax.com/asdf/123~~

~~Group start:~~
~~Name (String): Decoder~~
~~Description (Optional String): These variables describe the entire board~~
~~Replication count (integer): 1~~

~~Variable:~~

Name: Address
Type: Integer
Max: 2044
Min: 0
Description: This is the board address, in DCC space originally

Group start: (Note this is nested in "Decoder")
Name (String): Channel
Description (Optional String): Each Channel is one pair of output wires and contains two inputs
Replication count (integer): 4

Group start: (Note this is nested in "Channel")
Name (String): Input
Description (Optional String): Each Channel has two inputs, called "Switch" and "Aux"
Replication count (integer): 2

Producer start:
Name: Switch Input Active
Description: Driven by the 1st input wire for this channel. The variables control ...

Variable:
Name: Input Type
Type: Integer
Max: 10
Min: 0
Default: 0
Description: Specify the type of signal expected on this input
Map:
Name: Values
"0", "positive edge"
"1", "negative edge"
"2", "either edge"
...
Map End
Variable End
Variable:
Name: Input Task
Type: Integer
Max: 8
Min: 0
Default: 0
Description: Specify the local action when this input is active

Map:
Name: Values
"0", "Output toggle"
"1", "No output change"
"2", "Output thrown"
...
Map End
Variable End
Producer end:
Producer start:
Name: Aux Input Active
Description: Driven by the 2nd input wire for this channel. The variables control ....

Variable:
Name: Input Type
Type: Integer
Max: 10
Min: 0
Default: 0
Description: Specify the type of signal expected on this input
Map:
Name: Values
"0", "positive edge"
"1", "neither edge"
"2", "either edge"
...
Map End
Variable End
Variable:
Name: Input Task
Type: Integer
Max: 8
Min: 0
Default: 0
Description: Specify the local action when this input is active

Map:
Name: Values
"0", "Output toggle"
"1", "No output change"
"2", "Output thrown"
...
Map End
Variable End

~~Producer end:~~

~~Group end: (This is the end of the input group)~~

~~Variable:~~
~~Name: Output Type~~
~~Type: Integer~~
~~Max: 40~~
~~Min: 0~~
~~Default: 0~~
~~Description: Determines what the output leads do in response to events ....~~

~~Consumer start:~~
~~Name: Turnout Active Thrown~~
~~Description: Set the thrown output lead active and closed lead inactive.~~
~~Consumer end:~~

~~Consumer start:~~
~~Name: Turnout Active Closed~~
~~Description: Set the closed output lead active and thrown lead inactive.~~
~~Consumer end:~~

~~Consumer start:~~
~~Name: Turnout Active Both~~
~~Description: Sets both output leads active.~~
~~Consumer end:~~

~~Consumer start:~~
~~Name: Turnout Inactive~~
~~Description: Sets both output leads inactive.~~
~~Consumer end:~~

~~Group end: (This is end of the "Channel" group)~~

~~Group end: (This is end of the "Decoder" group)~~

~~Some thoughts based on putting this example together:~~

~~1) In a real DS54, there are subtle differences between the Switch and Aux configuration choices on the various channels. I blurred those here by documenting them identically via replication. For a real device, they could either be separately specified or (more likely) the differences wouldn't matter in a P/C-based device.~~

~~2) A DS54 can receive messages that put its output into four states: One side on, the other side on, both sides on, and neither side on. These four interacts with the "Output Type" setting in weird and~~

~~wonderful ways. This was represented as four consumers. This seems a much more logical way to configure the device, as it gives more flexibility to the rest of the layout that's originating the requests.~~

~~3) The DS54 inputs also generate messages. They are specified as two producers (for the active and inactive messages).~~

~~Implementation Notes~~

~~This section is non-normative notes and suggestions for implementors.~~

~~Some references for XML compression:~~

~~http://www.ibm.com/developerworks/xml/library/x-datacompression/index.html?emp=dw&cpb=dwxml&ct=dwnew&cr=dwnen&ccy=zz&csr=072111~~

~~http://www.cs.panam.edu/~artem/main/teaching/csci6370spring2011/papers/XML%20compression%20techniques%20A%20survey%20and%20comparison.pdf~~

~~On the other hand, a look-back compression algorithm has the advantage that it's cheap to decompress and might do almost as well:~~

~~http://excamera.com/sphinx/article-compression.html~~

~~XML strings can start with a UTF BOM (either 0xEF, 0xFF or 0xFE in the 1st byte, since there's no need to support UTF-32BE or UTF-32LE), or the UTF-8 text for "<?xml" which starts with 0x3C. (But concern about support for too many character sets!)~~

~~A first byte of 0x80 is defined as the "Compressed" indicator(s), followed by a byte that indicates the compression type. (We don't want to have too many kinds, as receivers need to implement all of them to be able to use the CDIFDI!) 0x80 00 is the code for our choice of default, see http://excamera.com/sphinx/article-compression.html~~

~~Remaining items~~

~~Internationalization of the XML content?~~

~~Talk about the encoding string in the <?xml first line, and UTF-8 coding as default; ASCII as subset.~~

4.7 [6.3] Internationalization of the Content

Not everybody runs their trains in English.

The xml:lang attribute could be used to define versions of the <name> and <description> elements that use an alternate language.[1] This approach can greatly increase the size of the XML content if the internationalization effort is very successful. On the other hand, it's transparent to XML-using nodes that don't implement internationalization.

4.8 [6.4] Machine-readable function classification

Currently the function's type is expressed in a free-text form in the <name> element. This has a wide expressibility, but presents challenges for user interface development:

[1] JMRI DecoderPro does this in its decoder definitions. That experience guides some of the rest of this paragraph.

- Text generally needs to be rendered with a font size that is readable for the given user.
- Long text takes a significant amount of screen real-estate.
- Text display assumes the user can read, which excludes children under a certain age.

Instead of this, a variety of competing model railroad systems define function types which are then represented by appropriate icons. The icons can be standardized in size, and convey significantly more information on a given real estate than text. This limits the user's choice (there is a fixed number of different icons provided by the implementation usually), but a larger number of functions will fit on the user interface than in a text version.

In order to add machine-readable function type, we need two things:

- An enumeration that lists possible function types and assigns numbers to them. There is such a list in TN-218 by RailCommunity.
- An attribute or element for transmitting the function enumeration in the FDI.~~Mini-XML effort; we're using as simple a subset as possible.~~

Table of Contents

1 Introduction....1
1.1 Requirements....1
1.2 Preferences....1
2 Annotations to the Standard....1
2.1 Introduction....1
2.2 Intended Use....2
2.3 Reference and Context....3
2.4 Content....3
2.5 Format....3
2.5.1 XML Elements....3
2.5.1.1 <fdi> Element....3
2.5.1.2 Element....3
2.5.1.3 <group> Element....4
2.5.1.4 <function> Element....4
3 Additional Information....4
3.1 Design points....4
3.2 Sample Function Definition XML:....4
3.3 Sample Older Implementation....6
4 Possible Future Work....6
4.1 Data Compression....6
4.2 Internationalization of the Content....6
4.3 Machine-readable function classification....7

