

Provisional protocol for CBUS (Version 4)

The Physical and Data Link layers (ISO 1 and 2)

The data link uses the industry standard CAN protocol. (ISO 11898/11519) This is a well established network which fulfils all the requirements of a MR layout control bus and removes any additional development effort by manufacturers. All the necessary hardware and software development tools are readily available, together with a wide range of suitable integrated circuits which handle all the CAN bus requirements. The physical connectors and wiring requirements are not specifically defined for CBUS at present. The CAN data may be sent via a pair of twisted wires and the power supply is that which suits the modules. The present modules use terminal blocks to allow maximum flexibility.

CAN frames are defined by the CAN protocol. There is the 'header' section, the 'data' section and a CRC error section. The chosen CAN rate is 125Kbits/ sec.

CAN specific elements are used for the transport only and the message content is kept in the data area. This allows the message to be separated from the transport and be 'transport independent'. A consequence of this is that it allows the use of the standard (11 bit) CAN ID and allocation of this ID is automatic and transparent to the message and nodes. For the SLiM mode nodes only, the CAN ID is set by switches and is the same as the node number (see below). The CAN Header is only used to establish message priority (arbitration) and CAN node uniqueness. There is no hardware based filtering on the CAN ID segment.

However, CAN has a comprehensive error checking mechanism and any frames with errors are rejected and a retry is triggered. As a consequence of this, we felt that no further error checking or checksum element was required in the message itself.

CBUS utilises a special method whereby CAN nodes are able to allocate themselves a unique ID within a 7 bit range. A joining node which does not have a CAN ID sends a zero data length frame with the RTR (Remote Transfer Request) bit set. All other nodes recognise this RTR frame and respond with a zero data length frame containing their own CAN ID. The arbitration process within the CAN nodes causes the node with the lowest ID to send first. Consequently, nodes send ID frames with successively increasing ID values until all nodes have transmitted. The joining node monitors these zero data frames and increments its own ID counter until there is no match and its timer has expired. It then allocates itself the next available ID number. A 7 bit range was chosen as currently available CAN transceivers do not support more than 110 nodes on a bus segment. Also, the time for the allocation process depends on the number of active nodes on a segment. With all 127 nodes, the time is only 60 milliseconds. The standard CAN header has 11 bits. The remaining 4 bits are used in CBUS for setting message priority.

Available CAN transceiver ICs limit the number of nodes per segment to 110 at the full specification. However, CBUS allows for multi-segment operation using bridge nodes between CAN segments with a maximum of 65535 uniquely identifiable nodes for the whole layout. (The node number is a 16 bit value as described below).

CBUS message format.

The general format of a CBUS message when using CAN is:

<CAN ID field : 11 bits>< message field of 0 to 8 bytes>

The general format of the message field is

<command byte>< 0 to 7 data bytes>

The command byte must be present except for the CAN ID enumeration process as described above. The CAN ID field is only present if CAN is used for the transport network.

The CAN ID field.

CAN ID field of 11 bits <aa bb : cccccc>

aa are the two dynamic priority bits with range 00, 01 and 10.

bb are two minor priority bits used to set the static message priority.

ccccc The 7 bit unique CAN ID value.

The dynamic priority bits allow for a message latency scheme where CAN frames that are waiting to be sent can increase their priority to ensure transmission within either a preset time or a preset number of retries. A value of 00 is a 'must go' state. If the minor priority bits can have a value of 11, then the CAN specification disallows 7 consecutive 1 bits in the MSBs of the header so the dynamic priority of 11 is not used.

The minor priority bits are set depending on the message type and its relative urgency. For layout control these would generally be 11 (lowest priority) with higher priority values being used for CAB to Command station transactions.

The data field:

This is from one to 8 bytes and contains the CBUS message.

<nnn:CCCC><data 0><data 1>< data 2><>< data 7>

nnn Used to determine the number of bytes in a message (000 to 111) and assists in parsing the message by high level protocols.

CCCC A 5 bit command or 'event qualifier'. This allows for 32 commands per message length.

The Application Layer.

The communication within CBUS is in the form of 'messages' which are independent of the transport . However the CAN frame format limits the message to a maximum of 8 bytes. The first byte is a 'command' byte where the three MSBs are the message length. <000 to 111> . This allows the remaining 5 bits for 32 commands per message length. There is no separate message length byte. The command byte itself is the first message byte so a command byte of <000CCCC> signifies no further message bytes and a command byte of <111CCCC> signifies 7 more message bytes. The inclusion of the message length in the command byte

allows simpler message parsing especially where CAN is not used. (A CAN frame has its own 'data length' byte).

The term 'command' has been adopted for convenience. It really describes what the message is about and does not imply a 'directive'. There must be at least one byte in the message. Zero byte messages are restricted to the enumeration scheme.

CBUS refers to nodes as an implementation of an interface to the bus. It uses an underlying data model in which nodes are identified by 16-bit node numbers, (NN) and each node can generate events which are identified by an additional 16 bit 'action' number. For layout control using the Producer / Consumer' model, an event comprises the 16 bit node number followed by the 16 bit action number. Thus an event is a 32 bit value. The inclusion of the producer node number in an event allows the event to be traced to its producer and assists in the allocation of node numbers to prevent overlaps and also, where desired, segmentation of node ranges, for example with modular layouts where each module is allocated a node range.

Note: This underlying model may be described by an XML DTD document. The various implementations of nodes in CBUS may be all accompanied by XML files describing them which use that DTD. However, there is no requirement for XML files and a documented description of their functions and mode of operation is all that is necessary to allow user configuration.

The Node/Event model is described in detail in the following sections, and is the underlying data model used in CBUS, for accessory and mobile nodes as well as other node types.

The application layer is subdivided into types of message transactions. These are:

1. Transactions used in layout control.
2. Transactions used for setting up and node programming. (service mode)
3. Transactions used for DCC CAB and Command Station messaging.

1. Layout Control

The approach adopted for layout control is an 'event driven' protocol, sometimes referred to as a 'Producer / Consumer' model. (ref 1). This allows a large degree of flexibility while still being user friendly. The term 'producer' will be used to describe a node that initiates 'events' and the 'consumer' a node that acts on events. In practice, there is no reason why a node cannot combine both functions in the same hardware but logically the functions are different.

A criterion adopted by CBUS is that an 'event' can be identified with its 'producer'. This is useful both for the user who can associate the event easily with the initiating device such as a switch connected to a node and also for subsequent diagnostics where the source of an event needs to be traced. A consequence of this decision is that each 'input device' e.g. a switch or pushbutton produces a unique event and there is no overlap among events produced by different devices.

A consumer node, generally one with outputs that activate layout devices such as turnouts or signals, will recognise an event and act on it appropriately. More than one consumer node

can act on the same event and does not have to act in the same way. How the consumer acts is determined by preset or programmed parameters within the consumer. There is no information contained in the event message other than an event number and an event type (such as ON or OFF). The event type is contained in the 'command' byte. Not only can several or many nodes act on the same event, a single node can act in the same way for different events. This allows, for example, switches on different control panels to activate the same turnouts or routes.

The scheme allows both very simple layout configurations where a control panel with switches activates individual turnouts, signals, aspects or complete routes directly as well as more complex layouts with a PC as the main control logic node. In this latter case, the events from switches or occupancy detectors will be recognised by the PC which will then generate the output events to operate the layout – based on rules and logical constructs set up by the user.

Event format.

Producer nodes will have a node number (NN) so that events can be identified with their 'activator'. The NN is a 16 bit number. Each producer node may have up to 64K events associated with it. The event number (EN) is a combination of the 16 bit NN and the 16 bit event or action from that node giving a 32 bit total., This gives a possible range of 64K events for each producer node, each with a type qualifier (ON, OFF, UNDEFINED etc). Some type qualifiers allow one or more additional bytes to be attached to the event, e.g. analogue values.

All events are five to eight byte messages, depending on the type qualifier.

For producer nodes, the event number (EN) comprises the node number as the high word and the event within that node as the low word. The basic message structure typically is:

```
<100: event ON ><NN==EN hi:16><EN lo:16> or  
<100: event OFF><NN==EN hi:16><EN lo:16>
```

For events carrying a one byte value the format is typically:

```
<101: event type><NN==EN hi:16><EN lo:16><val>
```

The EN high word is the producer NN. With the present command set, event ON is 0x90 and event OFF is 0x91. (There are other commands defined, these are just examples)

These are events triggered by a change in the input state so strictly the ON event means an input has changed from OFF to ON. Similarly an OFF event is the input going from ON to OFF. There is scope for events which are ON to indeterminate and OFF to indeterminate if needed. An on / off switch can be regarded as a single event generator so associating a physical entity with a numeric event.

Event configuration for layout control.

We have implemented two models for layout control nodes.

1,1 The SLiM model (**S**imple **L**ayout interface **M**odel)

This is a particularly user friendly version where a layout can be completely configured without the need of any programming tools or PC. It involves the setting of switches on the nodes and 'teaching' the consumers which event to respond to simply by activation of the required input (switch).

The producer nodes have a set of DIP (or rotary) switches to select the low bits of their node number. The prototype SLiM nodes have a 4 pole DIP switch giving a possible 16 producer nodes, each of which can have up to 64K events. The 8 input node has an additional 3 jumpers giving a NN range of 127. (only 0 to 99 allowed) These limitations on switches are purely for practical reasons and are not inherent in the CBUS protocol. The only user requirement is that no two producer nodes have the same NN setting. The actual numbers do not matter. In addition, SLiM node numbers are limited to the range 1-128. This allows SLiM nodes to operate seamlessly in FLiM mode (full PC supported mode) as described below, and to prevent assignments of duplicate node numbers by a network master (typically a command station or a PC).

The consumer nodes again have switches to select one of the possible outputs. Prototype nodes so far have 4 paired outputs for solenoid driving and 8 individual outputs for lights / relays etc. The former has a two way DIP switch (one of 4) and the latter has a three pole DIP (one of 8). The consumer nodes also have 'learn' and 'unlearn' switches. Again, the switches are limited by purely practical reasons and not the protocol.

Once the nodes are connected and powered up, the user selects the required output on the consumer node and puts it into the 'learn' mode. The event is then sent by activating the appropriate input (switch). The consumer remembers the event and associates it with the output set by the switch. When taken out of learn mode, the consumer always responds to that event by turning the output on or off. For producer nodes that generate events not triggered by the user's action directly (e.g. turnout position sensors) it is recommended to add a mechanism (selector + button, etc.) to allow simulating the events for the selected input lines so that consumer nodes can be set up to learn them.

While in learn mode, it is possible to set the switch to another output and resend the event. This way, the one event will activate more than one output. This is useful for route setting or signal aspects. When in learn mode, it doesn't matter whether the event sent is an ON or an OFF or if it is sent many times. Similarly, a different event can be sent with the same output selected. Thus two different switches, say on two control panels, can operate the same turnout or set the same route. Obviously, different consumers can learn the same event and act on it differently. The prototype solenoid driver and the 8 output signal driver also have a polarity switch so individual outputs can be reversed relative to others for the same event. This also allows output pairs to be toggled with a single event or multiple signal aspects to also correspond to a single event.

An event can also be removed from a consumer node by sending the event with the node in the 'unlearn' state, again set with a switch (or jumper). All events may be cleared by powering up a module with the unlearn switch set.

(Current consumer nodes implement 32 events but that is due to practical considerations.)

The above learning process enables a user to completely configure a layout for turnout and signal operation from a conventional control panel without any programming aids or any knowledge of how the nodes work.

SLiM nodes may also be used with a PC as the controller. Producer events are now learned by the PC and the PC program produces the appropriate output events which are learned by the consumers. Block occupancy detectors produce events in the same way as switches so the PC can operate a full interlocking scheme without any node programming at all.

In case the producer nodes are fitted with 7 way DIP switches (or equivalent) it is possible to have all 127 producer nodes using the SLiM method, each generating up to 65536 possible events. However, SLiM was not originally designed for large or complex layouts.

Note 1: The current SLiM implementation restricts user settable node numbers to a range of 1 to 100. A NN of zero is not allowed and numbers between 100 and 127 are reserved for specific types of node with fixed (predefined) node numbers. This includes the RS-232 and USB communication nodes. If the node switches are set to all zeroes, the firmware translates this to a NN of 1.

Note 2: The above description of the methods used on the prototype nodes is for illustration of the underlying process. It is not a restriction of the CBUS protocol. How the nodes are physically implemented is up to the designer / manufacturer.

1.2 The Full Layout interface Model. (FLiM)

Full specification CBUS nodes are fully programmable over the CBUS. This includes setting the producer node number, configuring the producer variables, setting consumer node numbers (needed for programming consumer nodes) and setting any consumer node variables. The 'learn' process remains basically the same except the consumer is put into learn (or unlearn) mode by a command to its Node Number (NN). The required output(s) are also sent with a command and the event is then sent and remembered. Configuration of a FLiM based layout would be appropriate to a PC with suitable software or an "intelligent" command station or programmer device.

FLiM nodes have the facility for reading back their NNs and other programmed variables via the CBUS using a PC, or "network manager". Allocated Node Numbers are in the range 256 and above so do not overlap with SLiM node numbers. It is possible to mix SLiM and FLiM nodes on the same layout so there is a cost free upgrade path for more complex layouts.

While the SLiM model only allows for 127 producer nodes, the FLiM node number range is 65536 so there is virtually no limit to the number of nodes, any of which may have multiple inputs, outputs or both. FLiM node numbers may be segmented for modular layouts.

As CAN transceivers only allow about 110 nodes on any segment, there will be a requirement for 'bridge nodes' (called CAN-CANs ?) which pass messages between CAN segments. As the message is independent of the CAN header, the CAN ID of the bridge can safely be used in target segments rather than the source CAN ID. The CAN ID is only required to be unique within the CAN segment. The bridge nodes will be able to adjust the dynamic priority of any transferred message to ensure speedy throughput. As node numbers are unique throughout the layout, but not related to the CAN segment, they can be accessed and programmed from anywhere in the system and events and their responses are also system wide.

In addition to 'events', CBUS has commands for polling event producers to read static values such as block occupancy, turnout position or analogue values. This could be used by a central PC for interlocking or monitoring layout current, track voltages etc.

2. Node programming and setting up.

All full specification nodes (FLiM) are fully configurable. The basic requirement is they have a unique node number (NN) which is a 16 bit value. Once this NN is recorded and stored in the node, all subsequent programming of the node is by reference to its Node Number. As node configuration is not a 'Producer / Consumer' process, a set of CBUS commands has been defined which describe the messages required for node configuration and parameter readback. These messages have the general format of:

```
[<MjPri><MinPri=3><CAN ID>]<command byte>< destination NN: 16 bit>  
< 0 to 5 message specific bytes>>
```

The Node Number must be entered with the node in its 'setup' mode. For fixed (accessory type) nodes, it will be put into setup mode with a switch or jumper on the node board. The NN will be entered by the programmer or PC with programming capability. Once entered, the node is taken out of 'setup' mode. Only one such node may be in 'setup' mode at any one time otherwise nodes may have duplicate node numbers. It is recommended that the allocation of the NN be done prior to installing the node under the layout or any other relatively inaccessible place. This NN is the equivalent of the DCC 'short address' which must be entered on the programming track. All subsequent programming is done 'on the main'.

FLiM nodes have a parameter block which is preset at manufacture and may be read back over the CBUS. This identifies the manufacturer, model or node type and all information required to identify the node and any corresponding XML or other specification file.

There is a set of CBUS commands for setting and reading node variables (NVs). The number and nature of these variables will depend greatly on the function and capability of the node itself and is largely manufacturer specific. A description of the variables may be provided in the form of an XML file associated with the specific node manufacturer, model and version as well as written documentation. Node variables will be accessed by reference to the node number and a node variable index. (NVI).

Currently the NVI is an 8 bit value giving 265 variables per node. It will be up to the manufacturer to specify what node variables are supported and their index number.

In addition to NVs, there is a set of up to 256 variables associated with every event (EVs). Those are addressed via the event number (32 bits) and an index byte. They are used to configure the action taken by the node upon occurrence of the specific event. EVs are typically modified during the configuration process. In SLiM mode the equivalent EVs are preset by the user with the switches to relate output actions to events.

Note: As for the SLiM description, the above is for illustrative purposes and relates to existing hardware implementations. It is not a limitation on the CBUS protocol itself.

3. DCC and CAB to Command Station communication. (provisional)

As well as a general purpose layout control bus, CBUS supports all the requirements of a 'CAB' bus.

Cabs attached to the bus send commands to a command station (CS), which is responsible for generating the proper stream of DCC packets on the rails, to maintain speed/direction, function activation and any reporting coming from the track. The CS coordinates ownership of locomotive addresses among the cabs and notifies the cabs of any state changes. Cabs may use the feedback from the CS to display status to the user. Due to the differences in operation, and the requirement to channel all mobile control commands through a CS and allow it to generate the appropriate DCC stream, mobile nodes use the Node/Event model in a different manner. Mobile devices are continuously controlled by a stream of speed/direction and function command updates (in the form of DCC packets). The CS is responsible for maintaining a list of all active mobile decoders and sending the appropriate updates. It is therefore more suitable to establish a "session" between the cab and command station which is linked to the specific task of controlling a locomotive. In CBUS, that "session" is identified by a single byte value assigned to the cab by the CS.

In order to establish such a session, the cab has to request control of a specific DCC address. If control is to be granted, the CS assigns an 8-bit 'handle' which is used for further control-related communication between the cab and the CS. This handle is sent back to the cab, and is used by the cab to request speed/direction and function activation, configure parameters used to control the locomotive (number of speed steps, relative direction etc) by generating events associated with the handle, as explained in the above sections. The cab will also release a locomotive via the handle. The CS is responsible for maintaining the links between handles and the controlled engine information records. As each cab session is identified by a different handle, the CS can address the cabs directly, or alternatively use the DCC address to address all the cabs controlling it and provide engine status reports, etc.

In addition, the cab as an input device serves to request changes which are global to the layout – such as turning power to the rails on or off, changing the system time and controlling power modes. There is no event number sent with these commands.

Mobile device control commands are similar in format to accessory control commands. The first byte contains the number of following data bytes and a command byte. For example, in order to request control of a DCC address the cab sends the following message :

```
[<MjPri><MinPri=2><CAN ID>]<010 0000><AAAAAAAA><AAAAAAAA>
```

Where the 2 address bytes may specify a short (7 bit) or long (14 bit) DCC address. The command byte here is 40 hex followed by the loco address. If the address is a 'long address', the top two bits of the first address byte are set.

The CS checks its refresh stack to find out if a locomotive with that address is already assigned to another throttle. If not it simply adds an entry to the engine list and fills in the locomotive address and attributes. If control is granted to the cab, the CS sends back the data necessary for the cab to control it, including the handle used for further controlling the locomotive :

```
[<MjPri><MinPri=2><NodeID>]<110 00001><handle: 8 bit><AddrH><AddrL><speed / dir><Flags>
```

<Dat1 > is the 8 bit 'handle' or session number assigned to the engine.

<Dat2> is the MS byte of the DCC address. For short addresses it is set to 0.

<Dat3> is the LS byte of the DCC address.

<Dat4> is the DCC format speed and direction byte
<Dat5> is the Function byte for F0 (FL) to F4 (DCC format)
<Dat6> is the Function byte for F5 to F8
<Dat7> is the Function byte for F9 to F12

Note: This same message is sent in response to a CAB request for current loco status. Initially, the speed and functions are zero. Only FNs F0 to F12 are held in the refresh stack. FNs F13 to F28 are sent as required.

There are additional commands to enable such activities as consisting and programming both 'on the main' and in service mode (programming track)

The CBUS command bytes for each message length are defined in a separate document.

Ref 1 The Next Generation. Networking Paradigm: Producer/Consumer Model.
by PA Murphy – 2000. Dedicated Systems Magazine. 2000- Q1, pp 26 – 28.
http://www.dedicated-systems.com/magazine/00q1/2000q1_p026.pdf

Mike Bolton and Gil Fuchs revision 4b 16/07/09 ©