VLCB Versatile Layout Control Bus

Event Teaching Service Specification

Service# 7 Teach

Version 1.1, March 2024, for Service version 1

Compatible with CBUS ® 4.0 Rev 8j

Table of Contents

Document History

Date	Changed by	Summary of changes	Service version
20th January 2023	Ian Hogg M.5144	Initial document split from DPH’s original Event Services specification.	1
14 April 2023	Ian Hogg M.5144	Changed name to VLCB	1

1 Introduction

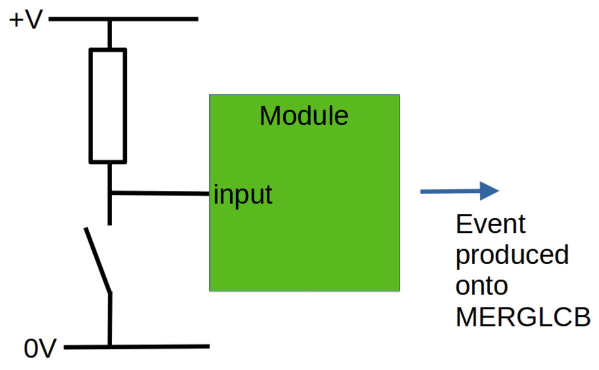
This document describes the service related to Teaching Events following the CBUS mechanism. This is an optional service, which may be added to the MNS.

This service is likely to be used in combination with one or both of the Event Producer Service and the Event Consumer Service.

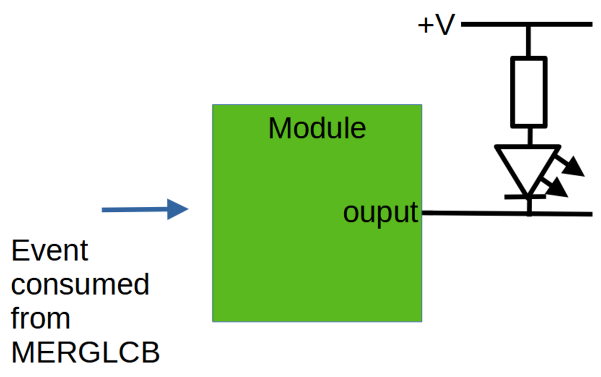
While the above are reported as separate services, practically these can be implemented as a single library, since they have very similar needs, with facilities to characterise the events as producer, consumer, or both. In addition, Learning is intimately related to the definition, storage, and search functions included with Events. While Short and Long events can be considered as separate, again, it is practical to implement them together.

2 Events Introduction

Event messages are a major method of sending information from one node to others. They are commonly used in the Producer-Consumer Network Model. The information is transmitted onto the bus by sending a message containing an ***event specific number*** representing a ***specific change of state*** on the layout. The event conveys the state of the Producer as being either ON or OFF. Most often this is a change in state of a layout accessory, such as an occupancy detector, but is not limited to this. One can think of this event-specific-number as analogous to a number-tag on a wire between two nodes. The nodes must agree on the meaning of the event-number, for example it might represent the change of state of a switch, whether it is open or closed, or the state of occupancy of a piece of track. This change is denoted by a message containing the Event-Number which is sent by the ***Producer Node***.



In addition, a ***Consumer Node*** may use the receipt of the Event message to perform some action, most commonly on an attached accessory. For example the action might be turning a LED on or off, or changing the position of a point (turnout).



One nice feature of Events is that when the Producer sends its Event-Message it does not need to know whether one, many, or no Consumer-Nodes are receiving or consuming its message. It means that the producer- and consumer-nodes are independent, and that additional consumer-nodes can be added at a later date without having to change the producer-node, or any of the other consumer-nodes.

As shown above, Events can be considered to be:

- **Produced** by a node due to some action, such as a button press, track occupancy, a specific time;
- **Consumed** by a node and causes a action, a lamp lighting, a bell ringing, a point moving;
- **Consumed and Produced,** some events can be both produced and consumed, e.g. an event representing a specific time might be produced by a clock-module and consumed by the same node to set the current time to that specific time. This is also referred to as “consume own events”.

How a module uses events will most often be determined by the node’s developer. However, in some cases the user may be able to change an event’s capabilities by writing to its associated event-variables. This must be documented in the manual accompanying the module.

Other Event-related messages include:
- Messages to put a node into, and out of, **Learning mode**, and to teach events to that node;
- **Request messages** for the status of an event from its producer;
- **Administrative messages** to clear, count, and determine event-space available in a node.

2.1 Long and Short Events

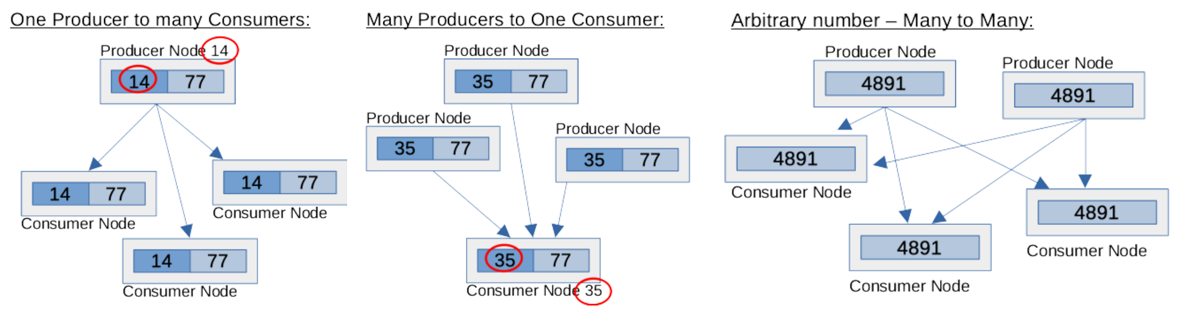
There are two separate Event-message forms:
- Long-events, with a 32 bit (4 byte) specific-event-number, and
- Short-events, with a 16 bit (2 byte) specific-event-number.

These forms are carried by two different sets of opcodes, and hence have two non-overlapping spaces.

The specific event numbers can be considered to be specific arbitrary numbers, but can be interpreted as the user wants to embed information into the events. A user may choose to use specific ranges for specific purposes. For example all points might be labelled 100-199, and events assigned to match.

CBUS classically uses short events in this way, but interprets long-events as having two fields, NN and EN, which represent the producer node's NN and an event number. This is convenient as a producer node can construct its long-events by combining its NN with a set of EN's, which are typically assigned to its inputs. This can be used well with one producer to one consumer, or one producer to many consumers relationships. One advantage of this interpretation is that the NN field can be used to indicate the producer node.

However, the above is only one interpretation, and others can be considered. For example, the NN in the NN:EN might be interpreted as the consumer node NN. This interpretation allows many producers to one consumer, with the NN identifying the target or consumer node. Conversely, one can simply assign an arbitrary number to the whole NN:EN and create a many producer to many consumer relationship. How events are used is up to the modeller. See below.



Of note, using arbitrary numbers lets one divide them up as one wishes, so for example:

- Make all buttons range from 1000-1999, all points 2000-2999, all signals 3000-3999;
- Number buttons, points and signals based on track mileage: a point at Mile 100 might be 10012, a signal 10022, and it controlling button 10052.

Whatever scheme is used, it is very important to keep excellent records.

Only the Opcode, NN:EN is transferred between modules. The configuration at the Producer to specify what happened to produce the event and the configuration at the Consumer what actions to perform when the event is received is specified using Event Variables (EVs) at the Producer and the Consumer. See 3 Event Variables (EV).

Note that the concept of 'ON' and 'OFF' is up to the user and may vary with use. For example, 'ON' may represent a button in the down-position, while 'OFF' represents the button in the up-position, or vice versa. A module may have let this be set with the Event Variables.

A producer module shall report a change in state of inputs or other items (e.g. a timer), by sending ARON/AROF or ARSON/ARSOF as appropriate.

A consumer module should record its status so that upon power up it can return to the previous state before power down. A module may optionally report its current state of outputs as events.

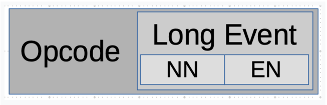
2.1.1 Long Events

Long Events are 32-bits (4 bytes) long. Any long-event can be sent by any node.

In CBUS, they are usually interpreted, and used, as having two 16 bit (2 byte) parts:

- ***eNN - the event's-Node-Number***, two-bytes or 1-65536 (0x0001-0xFFFF)
  NB: an eNN of 0 must not be used.
- ***EN - the event-number (EN)***, two-bytes or 0-65536 (0x0000-0xFFFF)

The combination is often written as ***NN:EN***.



And since a NN cannot be 0, the range of NN:EN is 0x00010000 to 0xFFFFFFFF, or 65536-4294967295(!)

Although Long-Events can be used in many-to-many relationships, VLCB recommends that they should only be used for one producer-node to one, or many, consumer-node relationships. By having the specific-event-number contain the producer-NodeNumber, tracing the origin of the event is made easy. While more than two producer-nodes can send the same Long-Event message, this defeats the tracing capability, and is often referred to as ***spoofing***. Event spoofing has been discouraged but may be useful for some modules and for some situations.

However, users are free to interpret the 32 bits of long-message in any way they wish. For example, one might use the NN field as the consumer's NN, and therefore allow a many producers to one consumer relationship. It is obviously important to record these choices for the user's future reference.

2.1.2 Long-Event Messages

- ACON - Accessory On
  This message represents the change of state to 'On'

  Data field: eNN:EN
- ACOF - Accessory Off

  This message represents the change of state to 'Off'

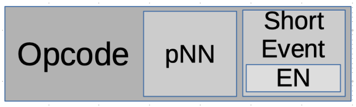
  Data field: eNN:EN

2.1.3 Short Events

Short-events are 16-bits (two bytes), and are represented by a single part:

- ***Event-Number (EN)***, 2-bytes or 0-65535

They have a distinct set of opcodes.



Note that the NN in the message is usually the producer-NodeNumber (pNN), and is included in the messages as a separate field. This is useful for tracing the origin of a specific message, as it can differ for each producer-node sending the same Short-Event message. NOTE, only the EN-field shall be used to match the events stored in the node, and Consumer nodes shall ignore the pNN when processing the event.

Note to implementers, short messages can be recorded as 00:EN, where the top two bytes are zero, and since long-events (NN:EN) cannot have a NN of zero there will not be a conflict. This allows both long and short events to be stored in one table, and share the same search function.

2.1.4 Short-Event Messages

- ASON - Accessory Short-On
    - This message represents the change of state to 'On'
    - Data-fields: producer-NodeNumber, EventNumber
- ASOF - Accessory Short-Off
    - This message represents the change of state to 'Off'
    - Data-fields: producer-NodeNumber, EventNumber

These message forms include the producerNodeNumber (NN) as a separate field.

2.1.5 Events With Data

The reader will have noted that there is not just ACON, ACOF, ASON, and ASOF messages, but also similar ones: ACONn, ACOFn, ASONn, and ASOFn. These allow events to be sent with associated data, and could be used, for example, to send a potentiometer position, which might be used to determine how fast an LED flashes.

Opcode	Number of data bytes
ACON1, ACOF1, ASON1, ASOF1	1
ACON2, ACOF2, ASON2, ASOF2	2
ACON3, ACOF3, ASON3, ASOF3	3

2.1.6 Distinguishing Events With and Without Data

How a module responds to these is up to the designer of the module, ie it is application dependent, but the following are recommended:

a) If a module has been taught an event which requires associated data, and that module receives the event without data then it should send a GRSP error. The GRSP message includes the opcode of the offending messages and an EN_DATA error-code.

b) If a module handles events with no data, it can optionally handle events with data in the same way, but this must be DOCUMENTED in the module's manual.

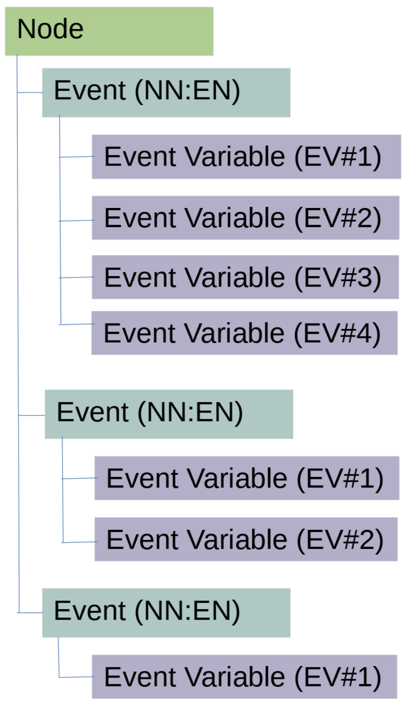
2.2 Dependencies on other services

The Teach service depends upon the mandatory Minimum Node Service.

If a module is designed to handle only default events then the Event Teach service is not required and events may be processed by the module's application code.

3 Event Variables (EV)

Event Variables (EVs) are a set of 1-byte values that allow a user to configure a module's Event-handling behaviour. Different Events may have differing numbers of associated Event Variables, and this will depend on module design.



Representation of a node with Events and Event-Variables

When a Consumer-Event is received by a module, the module uses the EVs associated with that event to determine what actions are to be taken by the module. For example, an event-variable might determine the end-position of a servo, or the flash-rate of a signal.

The EVs associated with an Event may also be used to determine under what conditions a module may produce/transmit that Event. VLCB does not impose any constraint on the usage of EVs however a software library implementing VLCB may impose restrictions on the format and/or use of EVs.

The process of teaching an Event and its associated Event Variable(s) is described in 4 Event Teaching

4 Event Teaching

As stated in the introduction, for events to be useful, two or more nodes have to agree on the meaning of a particular specific-event-number. This is done by 'teaching' the nodes to use the same specific-event-number so that the producer can send it on a specific change of state, and the consumers can use it to perform some resulting action.

4.1 Opcodes for Event Teaching

Refer to the VLCB Opcode Specification document for details of the opcodes

4.1.1 Events

The Event Creation/ Read/ Delete messages are straightforward:

Purpose	**Request**		**Responses**	
	Message	**Num bytes**	**Message**	**Num bytes**
Create event	`ENSET(NN, nn:en)`	6	`GRSP(NN, ENSET, 7, OK)`	4
Read events	`ENRD(NN)`	2	One `ENANS(NN, nn:en)` for each event	6
Delete event	`ENDEL(NN, nn:en)`	6	`GRSP(NN, ENDEL, 7, OK)`	4

4.1.2 Event Variables

A straightforward approach to reading and writing EVs requires more than the available 7 parameters available in a standard message.

To solve this problem a token is used to allow these requests to be split over two messages. The messages are 'joined' by the token. The token is a handle to a module's particular event.

Purpose	Request		Responses	
	Message	**Num bytes**	**Message**	**Num bytes**
Set Token	`ENTKSET(NN, nn:en, tk)`	6	`GRSP(NN, EVDEL, 7, resp) where resp may be one of: OK, INVALID_TOKEN, INVALID_EVENT`	5
Set an EV	`EVSET(NN, tk, EV#, EVval)`	5	`EVANS(NN, tk, EV#, EVval)`	5
Read an EV	`EVRD(NN, tk, EV#)`	4	`EVANS(NN, tk, EV#, EVval)` or multiple EVANS for each EV if EV#=0	5
Delete an EV	`EVDEL(NN, nn:en, EV#)`	7	`GRSP(NN, EVDEL, 7, OK)`	5

With this solution, the token is allocated by the management system and is valid until the module needs to reallocate the token space. Whilst using the token, if the module responds with INVALID_TOKEN then the management system must obtain a new token for further EV management requests.

Whenever an event is deleted any tokens associated with that event shall be invalidated.

The module uses the same token in responses that was passed in the request.

The management system needs to keep hold of the token/NN/nn:en combo so that it is able to associate responses with the correct events.

The module needs to maintain a table of token/nn:en so that it can associate requests containing tokens with the correct event.

Implementation Recommendations

Management systems should watch for ENTKGET from other systems and attempt to avoid using the same tokens as other systems.

Modules must maintain a table of token to event mappings. Although valid tokens are in the range 1..255, a module need only maintain a small (e.g. 8) tokens. Slots within the token table are used cyclically, overwriting any token assignment currently in place.

Any request to use a token which is no longer in the table shall result in an INVALID_TOKEN GRSP error response.

4.1.3 Supplemental operations

Purpose	**Request**		**Responses**	
	Message	**Num bytes**	**Message**	**Num bytes**
Request number of event slots available	`NNEVN(NN)`	2	`EVLNF(NN, number)`	3
Request number of events in use	`RQEVN(NN)`	2	`NUMEV(NN, number)`	3
Clear all events from a node.	`NNCLR(NN)`	2	`GRSP(NN, NNCLR, 7, ok)`	4

4.2 Teaching Long and Short events

Note that there are not specific messages to teach Long- and Short-Events. However, since Long-events (nn:en) must not have a nn of zero, Short-Events can be taught by sending the specific-event-number as (00:en). How this is handled in a module is implementation specific, and not part of this document. However, three common methods would be to have a single table, and maintain the (00:en) form for Short-Events; tag the specific-event-numbers; or maintain two tables, one each for Long- and Short-Events.

4.3 Number of Events

Parameter 4 returns the maximum possible number of events that can be stored by a module.

The EVLNF response from a module when sent an NNEVN request indicates the number of event slots remaining in the module.

The NUMEV response from a RQEVN request is the number of events currently stored by the module.

Note that the value in EVLNF (remaining space) added to the value in NUMEV (used space) must be less than or equal to Parameter 4 (maximum number of spaces).

4.4 Reading Events

A configuration tool may read all the events from a module using the ENRD message. The module responds with an ENANS message for each event stored within the module.

The module shall send the ENANS messages at a rate allowing the configuration tool to receive and handle these messages. The module shall transmit these messages with at least 10 ms between each message.

4.5 Writing Events

An event must be created before EVs can be assigned to it.

4.6 Removing an Event

A configuration tool may remove an event from a module using the ENDEL message. The event to be removed is specified with the module's NN, the event's node number and event number within the ENDEL message.

Node responds with an OK GRSP response or a NO_EVENT GRSP response if the specified event was not found.

All event variables for the event are also removed.x

4.7 Reading Event Variables

A configuration tool may read all the event variables associated with an event from a module using an ETKSET request followed by one or more EVRD messages.

A EVRD request for EV index 0 returns an EVANS message with the count of the number of EVs followed by an EVANS message for each EV.

If an invalid event variable index is specified the module will return an Invalid Event Variable Index GRSP error response.

A request for an EV with a valid index returns an EVANS message with the value of the EV.

If the initial ETKSET request fails then subsequent EVRD requests will also fail with INVALID_TOKEN. In this case the ETKSET and EVRDs should be reattempted.

4.8 Writing Event Variables

A configuration tool may write an EV with a ETKSET message followed by EVSET messages.

The meaning of the event variable values is module dependent and must be documented by the module designer.

If an invalid event token is specified the module will return an INVALID_TOKEN GRSP error response.

If an invalid event variable index is specified the module will return an Invalid Event Variable Index GRSP error response.

If an invalid event variable value is specified the module will return an Invalid Event Variable Value GRSP error response.

After a successful write of EV the module will respond with an EVANS with the actual EV value written.

4.9 Removing an Event Variable

A EVDEL message may be used to delete a single EV from an event.

A module may either implement EVs as a sparse data structure or represent missing EVs with an Unused EV value.

5 Additional Event Messages with Data

There is a set of Event-messages that carry additional data, these are:

- ACON1 / ACON2 / ACON3 Accessory ON-Long-event with 1 / 2 / 3 added bytes, respectively
- ACOF1 / ACOF2 / ACOF3 Accessory OFF-Long-event with 1 / 2 / 3 added bytes, respectively
- ASON1 / ASON2 / ASON3 Accessory ON-Short-event with 1 / 2 / 3 added bytes, respectively
- ASOF1 / ASOF2 / ASOF3 Accessory OFF-Short-event with 1 / 2 / 3 added bytes, respectively

Note that ACDAT, RQDAT and ARDAT are not events and are not included within the Event services.

There is no specific change in the teaching process to teach events with data. Whether a module requires data or sends data with events is application specific.

6 Processing of events

As stated in the introduction, for events to be useful, two or more nodes have to agree on the meaning of a particular specific-event-number. This is done by ‘teaching’ the nodes to use the same specific-event-number so that the producer can send it on a specific change of state, and the consumers can use it to perform some resulting action.

A module’s parameters contain information regarding events:

- Parameter 4 the maximum number of events supported by the module
- Parameter 5 the maximum number of event variables per event

7 Default Events

Modules may implement automatic configuration of events (default events) which can make a module easier to configure and use. Default events would be implemented as automatic configuration of EVs based upon a module's manufacturer's configuration, configuration of NVs or configuration of other events.

Default events behave in the same way as user configured events, it is only the way in which they come into existence which is different.

Default events must be reported by an event enquiry and if the module supports the Teach service they must be able to be deleted or reconfigured.

8 Service specific Modes

None.

9 Service Specific Status Codes

The following additional GRSP status codes are specified by the Event services.

These are really just place holders at the moment.

Code	Short Name	Comment
255	NO_EVENT	No event
254	NO_EV	No Event-Variable
253	INVALID_TOKEN	
252	INVALID_EV_INDEX	
251	INVALID_EV_VALUE	

10 Service Specific Diagnostic Data

10.1 DiagnosticCodes

The following DiagnosticCodes for the Event services are supported:

0x01: return the number of events taught since power on.

10.2 Diagnostic Payload Data Return

The following RDGN diagnostic data numbers are specified by the Event services.

Diagnostic Code	Diagnostic Byte1	Diagnostic Byte2	Comment
0x01	Count Hi	Count Lo	Count of number of events taught since power on.

11 Service Specific Automatic Power-up Tests

No service specific power-up tests are specified by the Event services.

12 Service Documentation

Modules implementing any of the Event services must provide full documentation. In particular the following are required:

Details of the EV usage.

13 Service Data

13.1 Parameters

The following parameters are associated with events and are to be provided.

Param#	Name	Usage	VLCB should set these values
4	No Events	Max number of events available	Max number of events supported.
5	No EV per event	Max number of Evs per event	Max no Evs per event
8.5	Learn	Indicates if the module is in Learn mode.	Set if the module is currently in Learn mode.

13.2 ESD data bytes

Data1 = maximum number of events

Data2 = maximum number of EVs per event

Data3 = unused, set to 0

14 Appendix - Discussion for Nerds

In the classic producer-consumer model, each event-number is unique and independent from other event-numbers. The CBUS developers decided that because many layout items are binary, they would use one event-number for a pair of trigger and actions, namely “On” and “Off”, and differentiate the action by having a pair of opcodes.

The CBUS Long-Events were developed when SLiM-modules were common, and the developers felt that many modules would be producers only. Therefore, it was convenient to pre-assign event numbers (ENs) to a module’s input lines (0-n), and to ensure that the specific-event-number was unique by including the module’s node-number as the high-part of that number (i.e.NN:EN).   Long-Events were promoted as a one-to-many solution, since one producer would connect to many consumers.

It then became clear that a many-to-many solution was required. This was mainly due to the development of throttles, since many throttles might control many accessories and locos. A solution was to invent and implement Short-Events, where the producer-NN is ignored, and only the EN, renamed to device number (DN), would be matched to other Short-Events stored in a consumer module.

These solutions work ok, and since there are 65k Short-Events, there is an ample supply for a layout.

In fact, the Long-Events are really 32-bit arbitrary numbers (albeit with the restriction that the top two bytes cannot be zero, if Short-Events are used). They can be assigned as the user likes, since modules are capable of sending any of them, and therefore they,too, can be used for many-to-many relationships.

The summary of the appendix is that advanced users may choose to push the bounds of the VLCB as they see fit. The rule “It’s your layout” still rules supreme.

15 Glossary

EN	Event Number (0-65536, 0x0000-0xFFFF).
Event	Represented as a 32bit value nn:en and when transmitted onto the VLCB bus indicates a change of state.
EV	Event Variable. Used to define the behaviour of a module associated with an event.
Event Index	A temporary identifier used as a shorthand for NN:EN.
NN	Node Number (1-65536, 0x0001-0xFFFF)
Teach	The process of associating an Event and its EVs with a module.