The websocket server is based on the Socket.IO javascript library, and the following is taken from the library documentation (<https://socket.io/docs/>)

*Socket.IO is****NOT****a WebSocket implementation. Although Socket.IO indeed uses WebSocket as a transport when possible, it adds additional metadata to each packet. That is why a WebSocket client will not be able to successfully connect to a Socket.IO server, and a Socket.IO client will not be able to connect to a plain WebSocket server either*

The method typically used to send a message is emit(**‘eventName’[, …args][, ack])** and the following methods document the eventName and any arguments for that method.

# Received Messages

## ACCESSORY\_LONG\_OFF

### Format

‘ACCESSORY\_LONG\_OFF’, {"nodeId": <value>, “eventId”: <value>}

#### Where

### nodeId:

* + - Node Number
    - Type: integer, 16 bit
    - Range: 0 to 0xFFFF

### eventid:

* + - event Number
    - Type: integer, 16 bit
    - Range: 0 to 0xFFFF

### Behaviour

Transmits the CBUS message ‘ACOF’ with Node Number and Event Number

## ACCESSORY\_LONG\_ON

### Format

‘ACCESSORY\_LONG\_ON’, {"nodeId": <value>, “eventId”: <value>}

#### Where

### nodeId:

* + - Node Number
    - Type: integer, 16 bit
    - Range: 0 to 0xFFFF

### eventid:

* + - Event Number
    - Type: integer, 16 bit
    - Range: 0 to 0xFFFF

### Behaviour

Transmits the CBUS message ‘ACON’ with Node Number and Event Number

## ACCESSORY\_SHORT\_OFF

### Format

‘ACCESSORY\_SHORT\_OFF’, {"nodeId": <value>, “deviceNumber”: <value>}

#### Where

### nodeId:

* + - Node Number
    - Type: integer, 16 bit
    - Range: 0 to 0xFFFF

### deviceNumber:

* + - Device Number – this is an effective ‘address’ of a physical device
    - Type: integer, 16 bit
    - Range: 0 to 0xFFFF

### Behaviour

This command transmits the CBUS short message ‘ASOF’ with Node Number and Device Number

Short (or device addressed) message use an ‘addressed’ scheme where each layout device (e.g switch or relay) has a known address or, as it’s termed, a ’device number’.

The node number of the sending node is still included in the CBUS message for traceability and diagnostics but is ignored by the receiving nodes or any attached PC software. In this case it is the Device Number that is unique to the layout device which either produces or consumes the event

When teaching short events, the Node Number is set to ‘0’ to indicate the evnt is a short event

## ACCESSORY\_SHORT\_OFF

### Format

‘ACCESSORY\_SHORT\_ON’, {"nodeId": <value>, “deviceNumber”: <value>}

#### Where

### nodeId:

* + - Node Number
    - Type: integer, 16 bit
    - Range: 0 to 0xFFFF

### deviceNumber:

* + - Device Number – this is an effective ‘address’ of a physical device
    - Type: integer, 16 bit
    - Range: 0 to 0xFFFF

### Behaviour

This command transmits the CBUS short message ‘ASON’ with Node Number and Device Number

Short (or device addressed) message use an ‘addressed’ scheme where each layout device (e.g switch or relay) has a known address or, as it’s termed, a ’device number’.

The node number of the sending node is still included in the CBUS message for traceability and diagnostics but is ignored by the receiving nodes or any attached PC software. In this case it is the Device Number that is unique to the layout device which either produces or consumes the event

When teaching short events, the Node Number is set to ‘0’ to indicate the evnt is a short event

## CLEAR\_CBUS\_ERRORS

### Format

‘CLEAR\_CBUS\_ERRORS’

### Behaviour

Clears the list of CBUS errors held by the system

## CLEAR\_NODE\_EVENTS

### Format

‘CLEAR\_NODE\_EVENTS’

### Behaviour

Clears the list of Node Events held by the system

## QUERY\_ALL\_NODES

### Format

‘QUERY\_ALL\_NODES’

### Behaviour

Transmits the CBUS message ‘QNN’ with no other arguments

## REFRESH\_EVENTS

### Format

'REFRESH\_EVENTS'

### Behaviour

Requests the list of events held by the system – see ‘events’ for response message

## REMOVE\_EVENT

### Format

‘REMOVE\_EVENT’, {"nodeId": <value>, "eventName": <value>}

#### Where

### nodeId:

* + - Node Number
    - Type: integer, 16 bit
    - Range: 0 to 0xFFFF

### eventName:

* + - event Name is comprised of a Node Number and an Event Number
    - Type: integer, 32 bit
    - Range: 0 to 0xFFFFFFFF

### Behaviour

Transmits the CBUS message ‘EVULN’, which ‘unlearns’ an event in the specified module

## REQUEST\_ALL\_EVENT\_VARIABLES

### Format

'REQUEST\_ALL\_EVENT\_VARIABLES', {"nodeId": <value>, "eventIndex": <value>, “variables”: <value>, “delay”: <value>}

#### Where

### nodeId:

* + - Node Number
    - Type: integer, 16 bit
    - Range: 0 to 0xFFFF

### eventIndex:

* + - Event Index – position of event in module’s even table
    - Type: integer, 8 bit
    - Range: 0 to 0xFF

### variables:

* + - Number of event variables to request
    - Type: integer, 8 bit
    - Range: Range: dependant on target module, but not exceeding 255

### delay:

* + - Time in milliseconds to stagger individual requests to module
    - Optional – defaults to 100 if not present
    - Type: integer
    - Range: ?

### Behaviour

Transmits the CBUS message ‘REVAL’, to request each variable in turn, up to the maximum number of variables requested

## REQUEST\_ALL\_NODE\_EVENTS

Read back all events in a node

### Format

' REQUEST\_ALL\_NODE\_EVENTS ', {"NodeId":< value>}

#### Where

### nodeId:

* + - Node Number
    - Type: integer, 16 bit
    - Range: 0 to 0xFFFF

### Behaviour

Transmits the CBUS message ‘NERD’, to request all stored events for the specified node

## REQUEST\_ALL\_NODE\_PARAMETERS

### Format

'REQUEST\_ALL\_NODE\_PARAMETERS', {"nodeId": <value>, "parameters": <value>, >, “delay”: <value>}

#### Where

### nodeId:

* + - Node Number
    - Type: integer, 16 bit
    - Range: 0 to 0xFFFF

### parameters:

* + - Number of parameters to request
    - Type: integer
    - Range: dependant on target module, but not exceeding 255

### delay:

* + - Time in milliseconds to stagger individual requests to module
    - Optional – defaults to 100 if not present
    - Type: integer
    - Range: ?

Range: 0 to 0xFF

### Behaviour

Transmits the CBUS message ‘RQNPN’, to request each parameter in turn, up to the maximum number of parameters requested

## REQUEST\_ALL\_NODE\_VARIABLES

### Format

' REQUEST\_ALL\_NODE\_VARIABLES ', {"NodeId": <value>, “variables”: <value>, “start”: <value>, “delay”: <value> }

#### Where

### nodeId:

* + - Node Number
    - Type: integer, 16 bit
    - Range: 0 to 0xFFFF

### variables:

* + - Index of last variable to collect
    - Type: integer, 8 bit
    - Range: Range: dependant on target module, but not exceeding 255

### start:

* + - Index of first variable to request, must be less than or equal to “variables”
    - Optional – defaults to 1 if not present (see note below)
    - Type: integer, 8 bit
    - Range: 0 to 0xFF

### delay:

* + - Time in milliseconds to stagger individual requests to module
    - Optional – defaults to 100 if not present
    - Type: integer
    - Range: ?

Note: node variable index 0 typically contains the number of node variables a specific module supports

### Behaviour

Transmits the CBUS message ‘NVRD’, to request each variable in turn, up to the maximum number of variables requested

## REQUEST\_EVENT\_VARIABLE

### Format

'REQUEST\_EVENT\_VARIABLE', {"nodeId": <value>, "eventIndex": <value>, “eventVariableId”: <value>}

#### Where

### nodeId:

* + - Node Number
    - Type: integer, 16 bit
    - Range: 0 to 0xFFFF

### eventIndex:

* + - Event Index – position of event in module’s even table
    - Type: integer, 8 bit
    - Range: 0 to 0xFF

### eventVariableId:

* + - Index of event variable to request
    - Type: integer, 8 bit
    - Range: Range: dependant on target module, but not exceeding 255

### Behaviour

Transmits the CBUS message ‘REVAL’, to request event variable at the specified index

## REQUEST\_NODE\_VARIABLE

### Format

### ' REQUEST\_NODE\_VARIABLE ', {"nodeId": <value>, "variableId": <value>}

#### Where

### nodeId:

* + - Node Number
    - Type: integer, 16 bit
    - Range: 0 to 0xFFFF

### variableId:

* + - Node Variable Index
    - Type: integer, 8 bit
    - Range: Range: dependant on target module, but not exceeding 255

### Behaviour

Transmits the CBUS message ‘NVRD’, to request Node Variable specified by the Node Variable Index

## REQUEST\_VERSION

### Format

### ' REQUEST\_VERSION '

### Behaviour

Requests the version of the system, see the transmitted ‘VERSION’ message for the expected response

## RQNPN

### Format

'RQNPN', {"nodeId": <value>, "parameter": <value>}

#### Where

### nodeId:

* + - Node Number
    - Type: integer, 16 bit
    - Range: 0 to 0xFFFF

### parameter:

* + - Node parameter Index
    - Type: integer, 8 bit
    - Range: dependant on target module, but not exceeding 255

### Behaviour

Transmits the CBUS message ‘RQNPN’, to request read of a node parameter by index

## TEACH\_EVENT

### Format

'TEACH\_EVENT', {"nodeId": <value>, "eventName": <value>, "eventId": <value>, "eventVal": <value>}

#### Where

### nodeId:

* + - Node Number
    - Type: integer, 16 bit
    - Range: 0 to 0xFFFF

### eventName:

* + - event Name is comprised of a Node Number and an Event Number
    - Type: integer, 32 bit
    - Range: 0 to 0xFFFFFFFF

### eventId:

* + - event Variable Index
    - Type: integer, 8 bit
    - Range: dependant on target module, but not exceeding 255

### eventVal:

* + - event Variable Value
    - Type: integer, 8 bit
    - Range: 0 to 0xFF

### Behaviour

Teaches a module an event and/or an associated event variable, and also updates the system records by transmitting the following sequence of CBUS messages

NNLRN(nodeId) put module into learn mode

EVLRN(eventName, eventId, eventVal) send event to be taught

NNULN(nodeId) take module out of learn mode

NNULN(nodeId) take module out of learn mode

NERD(nodeId) read back all stored events

RQEVN(nodeId) read number of stored events

## UPDATE\_EVENT\_VARIABLE

### Format

‘UPDATE\_EVENT\_VARIABLE’, {"nodeId": <value>, “eventIndex”: <value>, “eventName”: <value>, “eventVariableId”: <value>, “eventVariableValue”: <value>}

#### Where

### nodeId:

* + - Node Number
    - Type: integer, 16 bit
    - Range: 0 to 0xFFFF

### eventIndex:

* + - event Index
    - Type: integer, 8 bit
    - Range: 0 to 0xFF

### eventName:

* + - event Name is comprised of a Node Number and an Event Number
    - Type: integer, 32 bit
    - Range: 0 to 0xFFFFFFFF

### eventVariableId:

* + - event variable Index
    - Type: integer, 8 bit
    - Range: dependant on target module, but not exceeding 255

### eventVariableValue:

* + - event Variable Value
    - Type: integer, 8 bit
    - Range: 0 to 0xFF

### Behaviour

Teaches a module an event and/or an associated event variable, and also updates the system records by transmitting the following sequence of CBUS messages

NNLRN(nodeId) put module into learn mode

EVLRN(eventName, eventVariableId, eventVariableValue) send event to be taught

NNULN(nodeId) take module out of learn mode

REVAL(nodeId, eventIndex, eventVariableId) read event variable

NNULN(nodeId) take module out of learn mode

NERD(nodeId) read back all stored events

RQEVN(nodeId) read number of stored events

## UPDATE\_LAYOUT\_DETAILS

### Format

‘UPDATE\_LAYOUT\_DETAILS’

### Behaviour

Writes current layout to file

## UPDATE\_NODE\_VARIABLE

### Format

' UPDATE\_NODE\_VARIABLE', {"nodeId": <value>, "variableId": <value>, "variableValue": <value>}

#### Where

### nodeId:

* + - Node Number
    - Type: integer, 16 bit
    - Range: 0 to 0xFFFF

### variableId:

* + - Node Variable Index
    - Type: integer, 8 bit
    - Range: Range: dependant on target module, but not exceeding 255

### variableValue:

* + - Node Variable Value
    - Type: integer, 8 bit
    - Range: 0 to 0xFF

### Behaviour

Sets Node Variable, specified by the Node Variable Index, with the Variable value in the specified module

NVSET (nodeId, variableId, variableValue)

NVRD (nodeId, variableId)

## UPDATE\_NODE\_VARIABLE\_IN\_LEARN\_MODE

### Format

'UPDATE\_NODE\_VARIABLE\_IN\_LEARN\_MODE', {"nodeId": <value>, "variableId": <value>, "variableValue": <value>}

#### Where

### nodeId:

* + - Node Number
    - Type: integer, 16 bit
    - Range: 0 to 0xFFFF

### variableId:

* + - Node Variable Index
    - Type: integer, 8 bit
    - Range: 0 to 0xFF

### variableValue:

* + - Node Variable Value
    - Type: integer, 8 bit
    - Range: 0 to 0xFF

### Behaviour

Sets Node Variable, specified by the Node Variable Index, with the Variable value in the specified module

NNLRN (nodeId)

NVSET (nodeId, variableId, variableValue)

NNULN (nodeId)

NVRD (nodeId, variableId)

NNULN (nodeId)

# Transmitted Messages

## cbusError

### Format

‘cbusError’ <json data>

#### Where

An example of the json data follows

{

"1-2": {

"id": "1-2",

"type": "CBUS",

"Error": 2,

"Message": "Not in Learn Mode",

"node": 1,

"count": 1

}

}

## cbusNoSupport

### Format

‘cbusNoSupport’ <json data>

#### Where

An example of the json data follows

{

"FC": {

"opCode": "FC",

"msg": {

"message": ":SB780NFC0001"

},

"count": 1

}

}

## dccError

### Format

‘dccError’ <json data>

#### Where

An example of the json data follows

{

"type": "DCC",

"Error": 1,

"Message": "Loco Stack Full",

"data": "0000"

}

## dccSessions

### Format

‘dccSessions’ <json data>

#### Where

An example of the json data follows

{

"0": {

"count": 0,

"F1": 8,

"functions": [

4,

8,

12,

20,

28

],

"F2": 8,

"F3": 8,

"F4": 128,

"F5": 128

},

"1": {

"count": 0,

"F1": 2,

"functions": [

2

]

}

}

## events

### Format

‘events’ <json data>

#### Where

An example of the json data follows

[

{

"id": "00000000",

"nodeId": 0,

"eventId": 0,

"status": "on",

"type": "long",

"count": 2

},

{

"id": "00000001",

"nodeId": 0,

"eventId": 1,

"status": "on",

"type": "long",

"count": 2

}

]

## layoutDetails

### Format

‘layoutDetails’ <json data>

#### Where

An example of the json data follows

{

"layoutDetails": {

"title": "cbusServerAdmin",

"subTitle": "Settings",

"nextNodeId": 816

},

"nodeDetails": {

"200": {

"name": "Test EthNode",

"colour": "pink",

"group": "Board 1"

},

"259": {

"name": "Test 259",

"colour": "black",

"group": "Board 1"

},

"undefined": {

"colour": "black"

}

},

"eventDetails": {

"022D0001": {

"name": "Node 557 : Button 1",

"colour": "black",

"group": "Lever"

},

"022D0002": {

"name": "Button 2",

"colour": "red"

},

}

}

## nodes

### Format

‘nodes’ <json data>

#### Where

An example of the json data follows

{

"0": {

"node": 0,

"manuf": 165,

"module": "CANACC8",

"flags": 7,

"consumer": true,

"producer": true,

"flim": true,

"bootloader": false,

"coe": false,

"parameters": [],

"variables": [],

"actions": {},

"status": true,

"component": "merg-canacc8",

"learn": false,

"EvCount": 3

},

"1": {

"node": 1,

"manuf": 165,

"module": "CANSERVO8C",

"flags": 7,

"consumer": true,

"producer": true,

"flim": true,

"bootloader": false,

"coe": false,

"parameters": [],

"variables": [],

"actions": {},

"status": true,

"component": "merg-canservo8c",

"learn": false,

"EvCount": 3

}

}

## VERSION

### Format

‘VERSION’ <json data>

#### Where

An example of the json data follows

{

"major": "1",

"minor": "0",

"patch": "2"

}