

Module Descriptor

General

The module is described using key-value pairs, the key identifies the data, and the value is the data content, which can be various types of literal (fixed) data, such as string, numeric or an array to 'nest' further key-value pairs

Key-value pairs can be represented in many data formats, with JSON being the preferred format widely supported by many languages

Camel case is the preferred naming format for keys, unless specified otherwise

Camel case is the practice of starting each word with a capital letter, except for the first word.

There is no direct support for comments, but a key-value pair can be used where necessary

E.g. "comment": "comment added for clarity"

Filename

The filename is generated from the module name, the manufacturer/module identification and the firmware version. Leading with the module name makes the files a little easier to sort by eye

Kebab case formatting (- sign separator) is used to separate the 3 items

Note the firmware version is kept in the same format as used in firmware documentation, i.e. number followed by character

AAAAAAA-BBCC-DE

Where

AAAAAAA : variable length module name registered against the manufacturer ID & Module ID

BB : Manufacturer ID in hexadecimal - two hex digits

CC : Module ID in hexadecimal - two hex digits

D : Firmware major version in decimal - 1 to 3 digits

E : Firmware minor version in single ascii character

moduleName

This is the name registered against the manufacturer ID & Module ID for this specific module.

The value returned from the CBUS command NAME is typically a subset of this module name due to data restrictions

nodeVariables & eventVariables sections

These sections are the descriptors for node & event variables. Both of these have types and properties that work in the same way. The major difference between the two is that there is only one instance of the node variables for each module, where there can be multiple instances of the event variables, so the actual types reflect that difference, but share the same properties. A group type element is defined to allow the grouping of elements using an array. The types are described first, followed by the common properties.

Learn

One branch of firmware based on original CANSERVO8 code needs to be put into 'learn' mode before node variables can be programmed. Setting the [learn](#) key to a value of [true](#) in the [nodeVariable](#) section will indicate if this is required for this specific module. This property will default to false if not present, so only required if set to true.

Types

The type property indicates what the variable represents and how should be handled. Note not all types duplicated for both, as created on an 'as needed' basis.

EventVariableBitArray NodeVariableBitArray	Represents an 8 bit node variable where each bit can be selected independently - also known as flags, bitfield or multi-select. Uses bitCollection to define the bits & their labels.
EventVariableBitSingle NodeVariableBitSingle	Represent a single bit in a node variable. Uses the bitPosition property to identify which bit.
NodeVariableDual	Represents a two byte variable as a simple numeric input value.
EventVariableGroup NodeVariableGroup	Allows a collection of types to be logically grouped together. Uses the groupItems property.
EventVariableNumber NodeVariableNumber	Represents a variable as a simple numeric input value. Option to use min & max to limit user input. Option to use startBit & endBit to use a subset of the bits in a variable. Option to use displayOffset to adjust displayed values.
NodeVariableSlider	Represents a variable as a slider control. Option to use min & max to limit user input.
NodeVariableSlider6Bit	Represents a 6 bit variable as a slider control, using bits 0-5. Option to use min & max to limit user input. Option to use displayScale to adjust displayed values. Option to use displayUnits to display units of measure.

NodeVariableSlider7Bit	Represents a 7 bit variable as a slider control, using bits 0-6 Option to use min & max to limit user input Option to use displayScale to adjust displayed values Option to use displayUnits to display units of measure
EventVariableSelect NodeVariableSelect	Represents a control to select a single value from the array of options Option to use bitMask to define a subset of the bits to use Option to use displayScale to adjust displayed values Option to use displayUnits to display units of measure
EventVariableTabs NodeVariableTabs	Uses the tabPanels property to define a set of tabs and the content of the associated tab panels, the content is any of the other types including the 'group' types.

Properties for [nodeVariable](#) & [eventVariable](#)

property (key)	type	requirement	default
type	string	mandatory	Not Applicable
nodeVariableIndex eventVariableIndex	numeric	mandatory	Not Applicable
bitPosition	numeric	Mandatory for some types	Not Applicable
bitCollection	array	Mandatory for some types	Not Applicable
bitMask	numeric	optional	255
min	numeric	optional	0
max	numeric	optional	maximum size of the variable type
startBit	numeric	optional	0
endBit	numeric	optional	8
groupItems	array	Mandatory for some types	Not Applicable
displayTitle	string	mandatory	Not Applicable
displayScale	numeric	optional	1
displayUnits	string	optional	blank

displayOffset	numeric	optional	0
options	array	optional	Not Applicable
tabPanels	array	Mandatory for some types	Not Applicable

[bitCollection](#)

An array of bitPositions and associated labels used to define a collection of a variable number of bits and their labels used in the BitArray types. bitPositions start from 0
Each array entry of the form {"bitPosition": 1, "label": "bit description"}

[bitMask](#)

A bit value of 1 in the bitMask indicates that the corresponding bit position in the variable should be modified, a value of 0 shows the corresponding bit position in the variable should keep its original value. This allows a type to modify just part of a variable, and leave the remainder for another type to modify. See [options](#) description for an example of how it can be used

[min/max](#)

This pair usually relates to the raw value in the node variable, not the display value, unless stated otherwise in the type description above

[startBit/endBit](#)

[displayScale & displayUnits](#)

For numeric values, this pair allows the variable to be displayed in a 'friendly' fashion, e.g. a time delay in 100mS intervals would have a displayScaling of 100 and a displayUnits of 'mS'
These do not affect the underlying 'raw' variable

[displayOffset](#)

Used in special circumstances where the value the variable represents doesn't start at 0. E.g. a time delay may have a minimum of 500mS (i.e. the variable value of 0 represents 500mS), but intervals of 100mS could have a display offset of 500. Can also be used to display negative starting values, but less useful in this application)

groupItems

An array used by the [NodeVariableGroup](#) type to logically group other types together, e.g. to group more than one node variable to a single channel

options

Array of labels with values to be used in the [NodeVariableSelect](#) type.

Each array entry of the form {"label": "Options 1", "value": 0}

The value field maps onto the bits in the variable, for example, if the top 2 bits are used (bits 6 & 7), then the array will take the form

{"label": "event sent at ON end", "value": 0},	– bits 6 & 7 clear
{"label": "event sent when at OFF end", "value": 64},	– bit 6 set, 7 clear
{"label": "event sent at mid travel", "value": 128},	– bit 6 clear, bit 7 set
{"label": "Start of Day (SoD) event", "value": 192}	– bits 6 & 7 set

The [bitMask](#) option can be used to limit modifications to the specific bits, in this case a value of 192 would be used (bits 6 & 7 set to only allow those to be modified)

tabPanels

An array used by the [NodeVariableTabs](#) & [EventVariableTabs](#) types to logically group other types together in tabbed panels

Each entry in the array contains the displayTitle of the tab, and a further array of items that form the content of the tab panel

```
{ "displayTitle": "Tab number one",  
  "items": [  
    {  
      "type": "EventVariableNumber",  
      "eventVariableIndex": 1,  
      "displayTitle": "Output 1-1"  
    }  
  ]  
}
```

Example JSON file - fictitious module:

```
{  
  "moduleName": "CANACC5"  
  "nodeVariables": [  
    { "learn": false },  
    {
```

```

        "type": "group",
        "displayTitle": "Output 1",
        "groupItems": [
            {
                "type": "NodeVariableSlider7Bit",
                "nodeVariableIndex": 1,
                "displayTitle": "Pulse Duration",
                "displayScale": 20,
                "displayUnits": "Milli Seconds"
            },
            {
                "type": "NodeVariableBitSingle",
                "nodeVariableIndex": 1,
                "displayTitle": "Repeat enabled",
                "bit": 7
            }
        ]
    },
    ...
    ...
    {
        "type": "group",
        "displayTitle": "Output 8",
        "groupItems": [
            {
                "type": "NodeVariableSlider7Bit",
                "nodeVariableIndex": 8,
                "displayTitle": "Pulse Duration",
                "displayScale": 20,
                "displayUnits": "Milli Seconds"
            },
            {
                "type": "NodeVariableBitSingle",
                "nodeVariableIndex": 8,
                "displayTitle": "Repeat enabled",
                "bit": 7
            }
        ]
    }
],
"eventVariables": [
    {
        "type": "EventVariableBitArray",
        "eventVariableIndex": "1",

```

```

    "displayTitle": "Output pairs active",
    "bitCollection": [
      {"bitPosition": 0, "label": "pair 1&2"},
      {"bitPosition": 1, "label": "pair 3&4"},
      {"bitPosition": 2, "label": "pair 5&6"},
      {"bitPosition": 3, "label": "pair 7&8"}
    ]
  },
  {
    "type": "EventVariableBitArray",
    "eventVariableIndex": "2",
    "displayTitle": "event variable showing non-contiguous bit positions",
    "bitCollection": [
      {"bitPosition": 1, "label": "pair 1&2"},
      {"bitPosition": 4, "label": "pair 3&4"},
      {"bitPosition": 5, "label": "pair 5&6"},
      {"bitPosition": 7, "label": "pair 7&8"}
    ]
  },
  {
    "type": "EventVariableSelect",
    "eventVariableIndex": 3,
    "displayTitle": "Event position",
    "bitMask": 3,
    "options": [
      {"label": "event sent at ON end", "value": 0},
      {"label": "event sent when at OFF end", "value": 1},
      {"label": "event sent at mid travel", "value": 2},
      {"label": "Start of Day (SoD) event", "value": 3}
    ]
  }
]
}

```