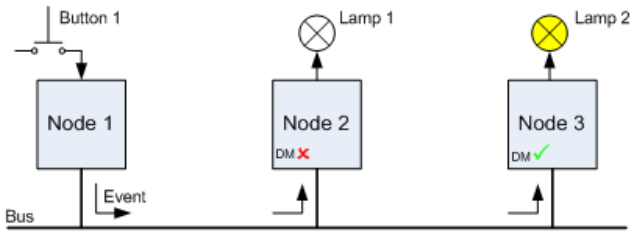
Introduction

VSCP stands for 'very simple control protocol', it is an automation protocol suitable for all sorts of automation jobs among which home-automation. In a VSCP network a common physical layer (wired bus, radio link, ...) connects the individual nodes to form the automation network. This network is a distributed system with all nodes working autonomous. On the bus there can be nodes reading switches, there can be nodes controlling lamps, nodes that read temperature, … . How they communicate & how this results in an automation function is detailed in the next chapters.
VSCP is independent from the physical layer (bus) used. There is a subset of the protocol (level I) that is very much tailored to the CAN bus. There is also a version of the protocol intended for Ethernet, one for TCP and also a wireless version of the protocol. All share the same common message fields & framework.

Event based

VSCP works based on 'events'. Each time an 'event' occurs this event is broadcasted towards all other nodes. Each node on the network will receive the event and will decide if this event needs to be handled or not. The example on the right shows a button being pressed.



Example 1

This will result in node 1 sending an event message onto the bus informing all others the button is pressed. Node 2 receives the message but decides this button should not trigger an event for node 2. Node 3 receives the message and decides this button should trigger Lamp 2 to turn on.
There can be 'events' for all sorts of things happening: it can be a button pressed, presence sensor being triggered, sun setting, … . Events can also be send periodically, for instance a temperature reading every minute. VSCP pre-defines a whole plethora of events that could be happening. These events are defined into classes & types. Whether or not an event received should be handled is determined by the 'decision matrix' or DM in short. Also the DM is explained further.

Event Class & Type

Events are organized into 'Classes'. A class is a collection of events that somehow belong together. There are classes for 'ALARM', 'MEASUREMENTS', 'CONTROL', 'INFORMATION', etc. Currently VSCP specifies about 25 classes, but has room for many more [1].
Each class is on its turn subdivided into 'types'. A type further specifies the event within the class. For instance, events of Class '20' (0x14) are 'INFORMATION' events. In this class there is a subtype '1' (0x01) 'BUTTON' signaling a button being pressed. In this same INFORMATION class there are also types to signal 'ON', 'OFF', 'BELOW LIMIT', etc. Likewise in class 'measurements' there are types to signal temperature, current, voltage, etc.
Having all these classes & types defined makes the nodes speak the same language. Perhaps some definitions make you wonder what you will ever use them for but it's nice they are there. For a full list of pre-defines classes & types check the VSCP wiki.

[1] VSCP Level II has a 16bit CLASS field allowing up to 65536 classes to be defined. Also the TYPE field is 16bit, allowing 65536 types to exist within each class.

VSCP event datagram structure
Events that are broadcasted contain a number of fields together forming one VSCP datagram. Exactly how these fields are mapped onto the physical layer is specified for a number of physical layer protocols such as CAN, Ethernet, TCP, … . For others it is not yet defined but it is in general not difficult to map these fields onto a physical layer protocol.

There are 2 'levels' of the VSCP protocol called 'LEVEL I' & 'LEVEL II'. They are both basically the same protocol but differ in size of the different fields. Level II is intended to be run on nodes that have little resource constraints and can easily cope with larger message sizes. Level I is intended to run on nodes with more constrained resources and fields are defined a bit more conservative.

In the tables below you can see both datagrams compared. Level II has 16bits for CLASS & Type allowing 65536 of each. Level I has a 9 bits CLASS allowing 512 classes and an 8 bit type allowing 256 types within a class. For most applications Level I messages should suffice.

Another difference is the ID being send in the datagram. For a level II datagram this ID used is the full 16 Byte (!) globally unique ID (GUID) being used. A level I message uses a 1 byte NickID. This NickID is an alternate ID for the node to identify it on the network segment. The node still has a GUID embedded but uses this NickID in the datagram. When the node is first powered up it will search for a free NickID and assign it to itself if found. The NickID can also be assigned by a central 'controller' which in most cases would be the gateway to a full LEVEL II network.

Priority	IDHardcoded	CLASS	TYPE	senderGUID	DataSize	DataPayload	CRC
3bits	1bit	16bits	16bits	16Bytes	16bits	0to512bytes	ITU

Table 1: VSCP datagram (level II)

Priority	IDHardcoded	CLASS	TYPE	senderNickID	DataSize	DataPayload	CRC
3bits	1bit	9bits	8bits	1Byte	4bits	0to8bytes	15 bits

Table 2: VSCP datagram (level I)

Decision Matrix
When events are received by a node the node needs to determine if it needs to execute a task based on that event. This is done by evaluating the 'decision matrix' or DM in short. The DM matrix is made of a number of IF … THEN … conditions. Each such IF/THEN condition is called a 'line' and multiple lines make up the decision matrix.
Conditions that can be evaluated by the IF are: SenderGUID (or NickID), Class, Type, zone, subzone. Zone and subzone will be discussed later as these parameters are applicable for some events.

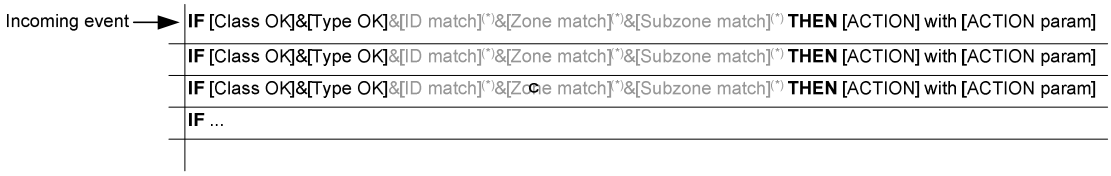


Figure 1: Decision Matrix

The Class & Type of the incoming message is always evaluated by a DM line. Evaluating Class & Type is done by passing the Class/Type thru a mask first & then comparing with a filter.

This method allows multiple class/Types to trigger a valid condition for 1 line of the DM. The other conditions for the DM line (SenderGUID, Zone, Subzone) are optionally evaluated.

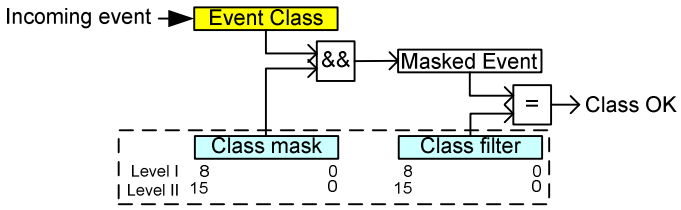


Figure 2: Class/Event mask & filter

When the IF condition is valid the 'THEN' or 'ACTION' is executed. Together with the 'ACTION' there can be 'ACTION parameters'. Exactly which ACTIONs can be executed by the receiving node is determined by the design of the node itself. ACTIONs can be 'turn-on relay x' with x being the 'action parameter' or the action can be 'execute DLL' or … each node developer can determine which actions can be executed by the node.

When multiple lines are present in the DM all lines get evaluated one by one. This means that if wished one certain event can trigger multiple lines to execute.

DataPayload

An event being send can also carry a data payload. The content & organization of this payload is depending on the class & type of the event. For example an event of class '10' (measurement) and type '6' (temperature) is will carry the temperature data (with coding determined by byte 0, degrees or celsius) in it's payload. A 'button' event will carry information about the button & button zone/subzone in it's datapayload. For each class/type the data formatting is determined in the spec, please consult the wiki for details.

Zone / Subzone

Some (quite some) events contain a field 'zone' and a field 'subzone' in their datapayload. This functionality is present to make 'grouping' of nodes possible. For instance we could determine that all buttons controlling a certain lamp are part of the same group. This simplifies the DM for certain scenarios. Instead of having one DM line a the lamp node for each button (1 line per button: IF button x then turn-on lamp) we could have 1 DM line only saying 'IF (zone match) THEN turn-on lamp'. Making multiple node switches part of a group is done by configuring the nodes, the firmware of the node will support this functionality. How this is done is explained in the section about 'register layout' & 'MDF'.

Putting it all together.

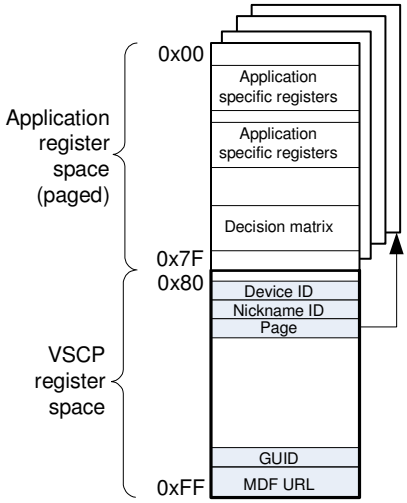
We've now covered the basic components of VSCP to make an automation function work. Taking once more example 1 we now know the node will send an event of class '20' (information) of type '1' (button). Since the button was configured to belong to subzone 2 the datapayload of the event will contain 'subzone = 2' information.

Both node 2 & node 3 will have an identical DM line. Their DM lines will read 'IF (class=20) & (type=1) & (subzone=match) THEN turn-on lamp'. However, node 2 will be configured to belong (for instance) to subzone 1 and node 3 will belong to subzone 2.

The incoming message will not result in a valid DM condition for node 2 since the subzone of the message does not match that of node 2. The incoming message will result in valid DM condition for node 3 since the subzones match. Node 3 will then execute his action: turning on the lamp.

Configuring a VSCP node

Each VSCP node provides some configuration options specifically for it's function. A button node would foresee some possibility to configure the zone/subzones the buttons belong to and a temperature node could have some possibility to set some measurement rates. Also setting the DM is part of configuring a node.



Configuration registers

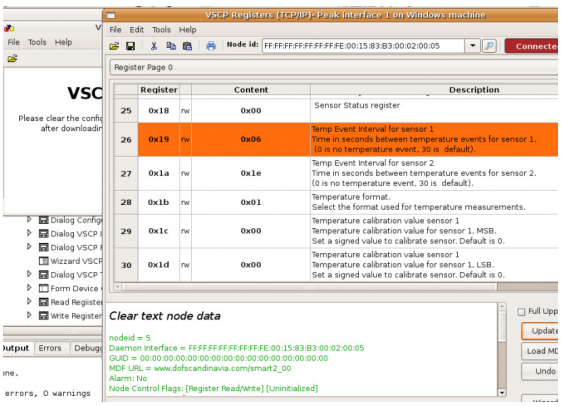
Configuring a node is done by writing to 'registers'. Each (Level I) node provides access to 256 registers. The highest 128 registers are reserved for VSCP core functions. In these 128 registers we find items such as node GUID, Nickname, MDF and a paging register. The lower 128 registers are free for application specific use. If 128 registers are not sufficient then there is a 16bit paging possibility. This allows for 65536 x 128 8bit registers for application use.
Writing/Reading these registers is done using 'CLASS 0' events. Class 0 events are 'VSCP protocol Functionality' messages intended for configuring and managing nodes.

Module Description File

Keeping track of which register serves what purpose can be a challenge, especially for the application specific registers. But this is where the module description file or MDF comes in. The MDF file is a machine-readable XML file describing the function of each register of a module, giving the configuration options for that register, etc. This file is used by configuration software (VSCPWorks) to show configuration options specific for the module addressed. The MDF file can be stored on the node itself and fetched from there by VSCPWorks, but more commonly the MDF file will be an XML file hosted on a webserver somewhere. A node then just needs to inform VSCPWorks where (URL) the XML file can be found. This URL is present in the VSCP reserved registers 0xe0-0xff.

VSCPWorks

VSCPworks is the PC (Linux & Windows) based tool to configure & manage the nodes. VSCPWorks allows reading/setting registers presenting those registers in a human-readable format by parsing the MDF file automatically. VSCPWorks also provides wizzards to set the decision matrix.



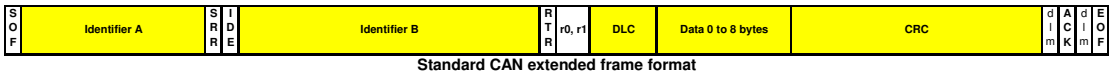
TODO (outside ‘4-pages’)

Explain ‘& friends’

-CANAL

-Deamon

-OHAS



Standard CAN extended frame format

VSCP mapped onto CAN extended frame format