VSCP

Very Simple Control Protocol

Reversion: 1.30 – 2005-05-12

the VSCP Team, http://www.vscp.org

Authors
Mark Marooth, Charles Tewiah, Andreas Nyholm, Ake Hedman

Table of Content

VSCP

The Very Simple Control Protocol

Introduction

There are a lot of protocols and technologies that claims to be the perfect protocol for use in SOHO (Small Office/HOme) control situations. TCP/IP, Bluetooth, different ways of power line communication, to name a few. VSCP is no such thing, period! It is just another solution to a common problem. In this case the need to manufacture low cost nodes that can work reliably and efficiently and be trusted in their day-to-day use.

The VSCP Protocol has been designed to be used in CAN networks. There is one reason for this, CAN is very reliable and today it is also low cost. Two good things. It is also possible to understand the inner workings of the protocol with just minimal effort and it is supported throughout the industry.

Even though the protocol is designed for CAN there is no need for CAN. It can be used equally well in other environments.

VSCP has been designed in two levels. Level 1 is intended for low-end machines while Level 2 is intended for higher level and speedier transport mechanisms such as TCP/IP.

The initial design goals in designing VSCP have been:

- It should be free and open. No usage, patent or other costs for its implementation/usage.
- Keep it simple stupid (K.I.S.S.) is the ruling design model.
- Flexible.
- A node should be constructed for the lowest possible cost.
- Uses standard components and cables.
- All nodes should be identifiable by a globally unique ID. This can also be used as a globally unique serial number for the device.
- No need to configure nodes with addresses. This is totally handled internally by the protocol.

Open?

This protocol is open and free in all aspects that are possible. We want you to contribute work back to the project if you do your own work based on our code but we also like to make as much of this work useful also in commercial projects. The tool we have chosen to do this is the GNU public license and the lesser GPL. For firmware code we use an even more open model so that there is no question that you are allowed to put the code in your own commercial projects if you feel to do that.

The GPL license and the LGPL license is included in the distribution of code in the file COPYING but can also be ordered from by writing to the Free Software Foundation, 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.

Current information about **canal**, **canald** and **VSCP** (***Very Simple Control Protocol)*** can be found at:

http://www.vscp.org
http://can.sourceforge.net

There are two mailing lists available on Sourceforge https://sourceforge.net/mail/?group_id=53560 that is about canal (**can_canal**) and VSCP (**can_vscp**) topics.

To subscribe to the canal list go to http://lists.sourceforge.net/lists/listinfo/can-canal

To subscribe to the VSCP list go to http://lists.sourceforge.net/lists/listinfo/can-vscp

VSCP level I

When the protocol was designed it's usefulness on the CAN bus was the main objective. The identifier length and many other things are therefore very "CAN-ish" in its design and behavior. CAN is however no prerequisite.

VSCP over CAN

- The maximum number of nodes is 254. In practice this is probably around 127 depending on drivers and media used.
- The transmission speed is 500 kilobits per second. Other speeds can be used but all nodes should support 500 kbps,
- The maximum length of the cabling in a segment is 100 meters using AWG24 or similar (CAT5).
- The cable should be terminated at both ends with 120 ohms.
- VSCP requires a 29-bit identifier with a meaning defined below.
- The protocol is compatible with dumb CAN nodes such as the Microchip MCP2502x/5x.

Appendix C shows how the VSCP Level I frame looks like for CAN.

VSCP level II

VSCP level two is designed for TCP/IP and other high-end network transport mechanisms that are able to handle a lot of data. With all the additional bandwidth available it would be ideal to always have a full addresses in packets for this type of media. That means that the address part in the header that is now 8-bits should be 129 bits instead. Information would be packed in the following way

3-bits **Priority**
16-bits **Class**
1-bit **Hardcoded**
16-bits **Type**
128-bits **Originating Address**

The HEAD is defined as
bit 7,6,5 - **Priority**
bit 4 - **Hardcoded**
bit 3,2,1,0 **reserved**

And the packet format is

0 - HEAD
1 - CLASS MSB
2 - CLASS LSB
3 - TYPE MSB
4 - TYPE LSB
5 - ORIGINATING ADDRESS MSB
...
20 - ORIGINATING ADDRESS LSB
21 - DATA SIZE MSB
22 - DATA SIZE LSB
..... data ... limited to max 512-25 = 487 bytes
len -2 - CRC MSB (Calculated on HEAD + CLASS + TYPE + ADDRESS + SIZE + DATA…)
len -1 - CRC LSB

The number above is the offset in the package. Len is the total datagram size.

Note also that the MSB is sent before the LSB (network byte order). So, for little endian machines such as a typical PC the byte order needs to be reversed for multi byte types.

There is a 24-byte overhead to send one byte of data so this is not a protocol which should be used where an efficient protocol for data intensive applications is required and where data is sent in small packets.

The CRC used is the 16-bit CCITT type and it should be calculated across all bytes except for the CRC itself.

The original message format is called Level 1 VSCP and this new format is Level 2 VSCP. The two formats can work side-by-side and all classes and types for Level 1 are also available in Level 2.

As indicated, the Class has resized from 9-bits to 16-bits allowing for 65535 classes. Class 0000000x xxxxxxxx is reserved for the original classes. A low-end device should ignore all packages with a class > 511.

A packet traveling from a Level 1 device out to the Level 2 world should have an address translation done by the master so that a full address will be visible on the level 2 net.

A packet traveling from a Level 2 net to a Level 1 net must have a class with a value that is less then 512 in order for it to be recognised. If it has it is aimed for the Level 1 network. Classes 512-1023 are reserved for packets that should stay in the Level 2 network but in all other aspects (the lower nine bits + type) are defined in the same manner as for the low-end net.

The Level 2 register abstraction level also has more registers (32-bit address is used) and all registers are 32–bits wide. Registers 0-255 are byte wide and are the same as for level 1. If these registers are read at level 2 they still is read as 32-bit but have the unused bits set to zero.

GUID's, Globally Unique Identifiers

To classify as a node in a VSCP net all nodes must be uniquely identified by a globally unique 16-byte (yes that is 16-byte (128 bits) not 16-bit) identifier. This number uniquely identifies all devices around the world and can be used as a means to obtain device descriptions as well as drivers for a specific platform and for a specific device.

The manufacturer of the device can also use the number as a serial number to track manufactured devices.

In many other environments and protocols there is a high cost in getting a globally unique number for your equipment. This is not the case with VSCP. If you own an Ethernet card you also have what is needed to create your own GUID's.

The GUID address is not normally used during communication with a node. Instead an 8-bit address is used. This gives a low protocol overhead. A segment can have a maximum of 127 nodes even if the address gives the possibility for 256 nodes. The 8-bit address is received from a master node called the *segment controller*. The short address is also called the nodes *nickname id* or *nickname address*.

Besides the GUID it is recommended that all nodes should have a node description string in the firmware that points to a URL that can give full information about the node and its family of devices. As well as providing information about the node, this address can point at drivers for various operating systems or segment controller environments.

Reserved GUID's

- Some GUID's are reserved and unavailable for assignment. Appendix A list these and also assigned id's.

The VSCP team controls the rest of the addresses and will allocate addresses to individuals or companies by them sending a request to **guid_request@vscp.org**. You can request a series of 32-bits making it possible for you to manufacture 4294967295 nodes. If you need more (!!!) you can ask for another series. There is no cost for reserving a series. Appendix A in this document contains a list of assigned addresses which will also be available at http://www.vscp.org

Level I Node types

On each segment there can be two kind of nodes. Dynamic and hardcoded.

Dynamic nodes

Are VSCP nodes that confirm fully to the Level I part of this document. This means they have

- a GUID.
- the register model implemented.
- all or most control messages of class zero implemented. As a minimum register read/write should be implemented id addition to the events related to the nickname discovery.
- Have the hard coded bit in the id is set to zero.
- Must react on **PROBE message** on its assigned address with a **probe ACK** and should send out the ACK with the hard coded bit set.

Sample implementations are available at http://www.vscp.org

Hard coded nodes

Are VSCP nodes which have a loose conformity to the Level 1 part of this document.

They can have a nickname address set in hardware or it can have the nickname discovery process implemented. In the former case it is up to the installer to assign a unique address to the node in the segment and remember to have the hard coded bit set to one. **There can not be any hard coded nodes with the hard coded bit set to zero.**

Very simple hard coded nodes can therefore be implemented. A node that send out an event at certain times is typical and a button node that send out an **on-even**t when the button is pressed is another example.

Level II Node types

Level II nodes are intended for media with higher bandwidth then Level I nodes and no nickname procedure is therefore implemented on Level II. As result of this the full GUID is sent in each packet.

The header for a level II event is defined as

```
// This structure is for VSCP Level II
typedef struct _vscpMsg {

      _u16 crc;          // CRC checksum
      _u8  *pdata;       // Ponter to data. Max 487 (512- 25) bytes
                         // Following two are for daemon internal use
      _u32 obid;         // Used by driver for channel info etc.
      _u32 timestamp;    //  Relative  time  stamp  for  package  in
microseconds

// CRC should be calculated from
// here to end + datablock


      _u8 head;          // bit 765 prioriy,
                         //    Priority 0-7 where 0 is highest.
                         // bit 4 = hardcoded, true for a
                         //    hardcoded device.
                         // bit 3 = Dont calculate CRC,
                         //    false for CRC usage.
      _u16 vscp_class;   // VSCP class
      _u16 vscp_type;    // VSCP type
      _u8  GUID[ 16 ];   // Node address MSB -> LSB
      _u16 sizeData;     // Number of valid data bytes
} vscpMsg;
```

The biggest differences is that the GUID is sent in each event and that both class and type has a 16-bit length.

The CRC is calculated using the CCITT polynomial see appendix E.

The max size for a Level II event is 512 bytes and as the header takes up 25 bytes the payload or datapart can take up max 487 bytes. This can seem a bit strange looking at the above structure but in a datagram or aother package not all of the above members are present. The format in a stream is

```
// VSCP LEVEL II UDP datagram offsets
#define VSCP_UDP_POS_HEAD                    0
#define VSCP_UDP_POS_CLASS                   1
#define VSCP_UDP_POS_TYPE                    3
#define VSCP_UDP_POS_GUID                    5
#define VSCP_UDP_POS_SIZE                    21
#define VSCP_UDP_POS_DATA                    23
#define VSCP_UDP_POS_CRC                     len - 2
```

As is noted the obid and timestamp is not present in the actual stream and is only inserted by the driver interface software.

Level I events can travel in the Level II world. This is because all Level I events are repleted in Class=1024. As nicknames are not available on Level II they are replaced in the events by the full GUID. This is an important possibility in larger installations.

A message from a node on a Level I that is transfered out on a Level II can have the nodes own GUID set or the GUID of the interface between the Level I and the Level II segment. Which method to choose is up to the implementor as long as the GUID's are unique.

For interfaces the machine MAC address, if available, is a good base for a GUID which easily can map to real physical nodes that are handled by the interface. By using this method each MAC address gives 65536 unique GUID's.

Other methods to generate GUID's s also available form more information see Appendix A.

Address or “nickname” assignment for Level I nodes

All nodes in a VSCP network segment need a way to get their nicknames ID’s, this is called the **nickname discovery process**.

A VSCP Level 1 segment can have a maximum of 254 nodes + 254 possible hard coded nodes. A segment can have a master that handles address or nickname assignment but this is not a requirement.

After a node has got its nickname id it should save this id in permanent non-volatile storage and use it until it is told to stop using it. Even if a node forgets its nickname a segment controller must have a method to reassign the id to the node. The master needs to store the nodes full address to accomplishe this.

Node segment initialization

Dynamic nodes

In a segment where a new node is added the following scenario is used.

1. The process starts by pressing a button or similar on the node. If the node has a nickname assigned to it, it forgets this nickname and enters the initialization state. Uninitiated nodes use the reserved node id **0xff**.

2. The node sends a **PROBE message** to address 0 (the address reserved for the master of a segment) using **0xff** as its own address. The priority of this event is set to 0x07. The master (if available) now has a chance to assign a nickname to the node.

   If no nickname assignment occurs the node checks the other possible nicknames (1-254) in turn. The node listens for a response event, **probe ACK,** from a node (which may already have the nickname assigned) for five seconds before concluding that the id is free and then uses the id as its own nickname id. On slower medium increase this timeout at will.

   It is recommended that some visual indication is shown to indicate success. A blinking green led that turns steady green after a node has got its nickname is the recommended indication. If there is a response for all addresses a failure condition is set (segment full) and the node goes to sleep.

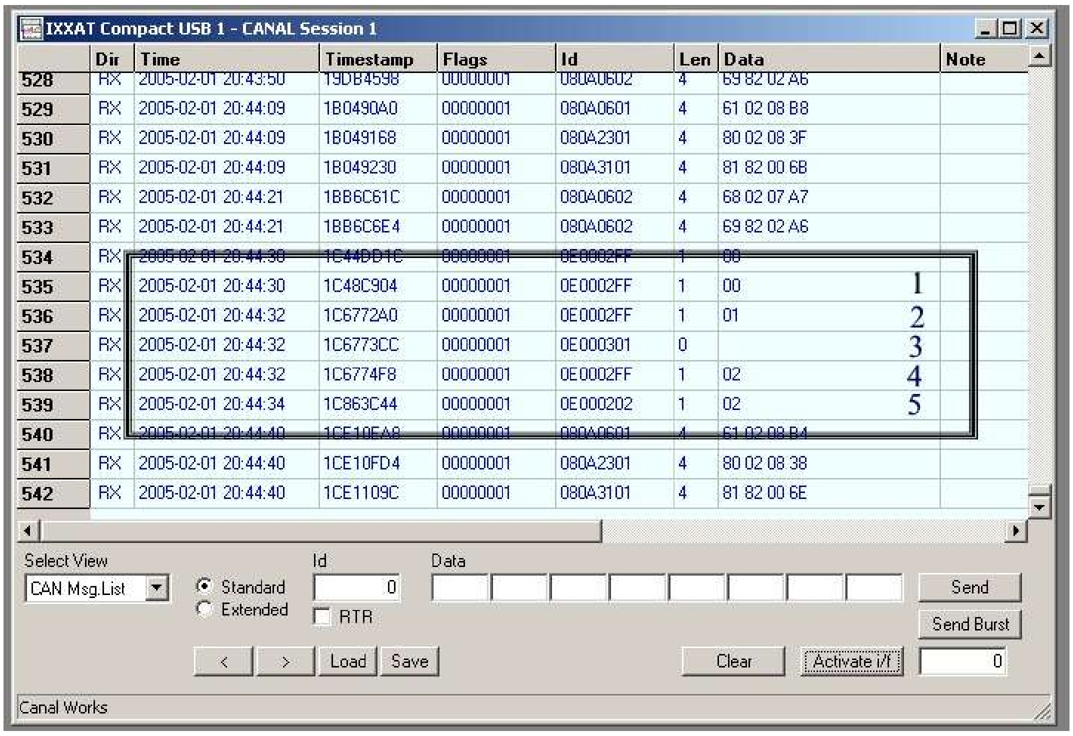
   On an insecure medium such as RF it is recommended that the Probe is sent several time in a row to make sure that the nickname actually is free.

3. After it assigns a nickname to itself the node sends **Nickname id accepted** using its new nickname id to inform the segment of its identity.

4. It’s now possible for other nodes to check the capabilities of this new node using read commands.

Only one node at time can go through the initialization process.

The following picture show the nickname discovery process for a newly added node on a segment


IXXAT Compact USB 1 - CANAL Session 1

	Dir	Time	Timestamp	Flags	Id	Len	Data	Note
528	RX	2005-02-01 20:43:50	19DB4598	00000001	080A0602	4	69 82 02 A6	
529	RX	2005-02-01 20:44:09	1B0490A0	00000001	080A0601	4	61 02 08 B8	
530	RX	2005-02-01 20:44:09	1B049168	00000001	080A2301	4	80 02 08 3F	
531	RX	2005-02-01 20:44:09	1B049230	00000001	080A3101	4	81 82 00 6B	
532	RX	2005-02-01 20:44:21	1BB6C61C	00000001	080A0602	4	68 02 07 A7	
533	RX	2005-02-01 20:44:21	1BB6C6E4	00000001	080A0602	4	69 82 02 A6	
534	RX	2005-02-01 20:44:30	1C44DD1C	00000001	0E0002FF	1	00	
535	RX	2005-02-01 20:44:30	1C48C904	00000001	0E0002FF	1	00	1
536	RX	2005-02-01 20:44:32	1C6772A0	00000001	0E0002FF	1	01	2
537	RX	2005-02-01 20:44:32	1C6773CC	00000001	0E000301	0		3
538	RX	2005-02-01 20:44:32	1C6774F8	00000001	0E0002FF	1	02	4
539	RX	2005-02-01 20:44:34	1C863C44	00000001	0E000202	1	02	5
540	RX	2005-02-01 20:44:40	1CE10EA8	00000001	080A0601	4	61 02 08 B4	
541	RX	2005-02-01 20:44:40	1CE10FD4	00000001	080A2301	4	80 02 08 38	
542	RX	2005-02-01 20:44:40	1CE1109C	00000001	080A3101	4	81 82 00 6E	

1. The node which initially have its nickname set to 0xff probe for a segment controller. Class = 0, Type = 2

2. No segment controller answerers the probe and a new probe is therefore sent to a node with nickname=1. Again Class=0, Type=2.

3. There is anode with nickname= 1 already on the segment and it answerers the probe with “probe ACK” Class=0, Type=3. The initiating node now knows this nickname is already in use.

4. A new probe is sent to a node with nickname=2. Again Class=0, Type=2.

5. No ACK is received and the node conclude that the nickname=2 is free and assign it to itself. It then send a probe again with the new nickname assigned reporting a “new node on line”.

Hard coded nodes

Things are a little different for hard coded nodes

If a hard coded node has its address set in hardware it starts working on the segment immediately.

If the nickname discovery method is implemented it goes through the same steps (1-3) as the dynamic node. In this case all hard coded nodes on the segment must recognize and react on the probe-event.

The hard coded bit should always be set for a hard coded node regardless if the nickname discovery method is implemented or not.

Example

A typical scenario for a master less segment can be a big room where there are several switches to turn a light on or off. During the installation the switches are installed and initialized.

When each switch is initialized it check the segment for a free nickname and grabs it and stores it in local nonvolatile memory. By being conected to the segment the installer makes not of the ids. It is of course also possible to set the nicknames to some desired value instead.

Additionally, VSCP aware relays are installed and also initialized to handle the actual switching of lighting. Again each in tern are initialized and the segment unique nickname noted.

At this stage the switches and the releay nodes have no connection with each other. One can press any switch and an on-event is sent on the segment but the relays don't know how to react on it.

We do this by entering some elements in the decision matrix of the relay nodes.

- If on-event is received from node with nickname n1 set relay on.
- If on-event is received from node with nickname n2 set relay on.
- If on-event s received from node with nickname n2 set relay on.

etc and the same for off-event

- If off-event is received from node with nickname n1 set relay off.
- If off-event is received from node with nickname n2 set relay off.
- If off-event s received from node with nickname n2 set relay off.

As the decsion matrix also is stored in the nodes non volatile storage the system is now coupled together in this way until changed sometime in the future.

To have switches in this way that send on and off events is not so smart when you have a visible indication (lights go on or off) and it would have been much better to let the switches send only an on-event and let the relay-node decide what to do. In this case the decion matrix would loook

- If on-event is received from node with nickname n1 toggle relay state.
- If on-event is received from node with nickname n2 toggle relay state.
- If on-event s received from node with nickname n2 toggle relay state.

But how about a situation when we need a visual indication on the switch for instance? This can be typical when we turn a boiler or something off.

The answer is simple. We just look for a message from the device we control. In the decision matrix of the switches we just enter

- If on-event is received from node with nickname s1 - status light on.
- If off-event is received from node with nickname s1 - status light off.

It s very easy to add aswitch to the scenario above and it can be even easier if the zone concept s used. In this concept each switch on/off message add information on which zone it controls and the same change is done in the decision matrix We get someting like

- If an on-event is received for zone x1 turn on relay

We now get even more flexible when we need to add/change the setup.

What if I want to control the lights from my PC?

No problem just send the same on-event to the zone from the PC. The relay and the switches will behave just as a new switch has been added.

What if whant a repote control that controls the lightning?

Just let the remote control interface have a decisn matrix element that sens out the on-event to the zone when the selected key s pressed.

What if I want the lights to be turned on when the alarm goes off?

Same solution here. Program the alarm control to send the on.event to the zone on alarm.

You imagination is the only limitation....

Level I - Register Abstraction Model

The model used for reads and writes is defined in this section. Register **0x00** – **0x7f** are application specific.

Registers between 0x80-0xff are reserved for VSCP usage.

If the node has implemented EDA the decision matrix is stored in application register space.

With Node control flags (0x83) the node behavior can be controlled. The startup control is two bits that can have two values, binary 10 and binary 01. All other values are invalid and are translated to binary 10. If set to binary 10 it will prevent the initialization routine of the node to start before the initialization button on the node has been pressed.

Bit 5 of the node control flags write protects all user registers if cleared (== 1 is write enabled).

The page select registers (0x92/0x93) can be used if more application specific registers are needed. The page is selected by writing a 16-bit page in these positions. This gives a theoretical user registry space of 256 * 65535 bytes (65535 pages of 256 bytes each). Note that the normal register read/write method can not be used to access this space. The page read/write methods are used instead.

0x98 Buffer size for device is information for control nodes of limitations of the buffer space for a Level II node. Level I nodes have eight in this position. Level II nodes that can accept the normal Level II packet have 0 which indicates 512-25 bytes or can have some number that represent the actual buffer. The Page read/write, boot events etc can handle more then eight data bytes if the buffer is larger then eight even if the event is a Level I event.

The VSCP registers are defined as follows:

Address	Access Mode	Description
0x00 – 0x7f	---	Device specific
0x80	Read Only	Alarm status register content (!= 0 indicates alarm). Condition is reset by a read operation. The bits represent different alarm conditions.
0x81	Read Only	VSCP Major version number this device is constructed for.
0x82	Read Only	VSCP Minor version number this device is constructed for.
0x83	Read/Write	Node control flags **bit 7** –Startup control **bit 6** – Startup control **bit 5** – **r/w** control of registry below 0x80. (1 means write enabled) **bit 4** – Reserved **bit 3** – Reserved **bit 2** – Reserved **bit 1** – Reserved **bit 0** – Reserved
0x84	Read/Write	User ID 0 – Client settable node id byte 0.
0x85	Read/Write	User ID 1 – Client settable node id byte 1.
0x86	Read/Write	User ID 2 – Client settable node id byte 2.
0x87	Read/Write	User ID 3 – Client settable node id byte 3.
0x88	Read/Write	User ID 4 – Client settable node id byte 4.
0x89	Read only	Manufacturer device ID byte 0.
0x8a	Read only	Manufacturer device ID byte 1.
0x8b	Read only	Manufacturer device ID byte 2.
0x8c	Read only	Manufacturer device ID byte 3.
0x8d	Read only	Manufacturer sub device ID byte 0.
0x8e	Read only	Manufacturer sub device ID byte 1.
0x8f	Read only	Manufacturer sub device ID byte 2.
0x90	Read only	Manufacturer sub device ID byte 3.
0x91	Read only	Nickname id for node if assigned or 0xff if no nickname id assigned.
0x92	Read/Write	Page select register **MSB**
0x93	Read/Write	Page Select register **LSB**
0x94	Read Only	Firmware major version number.
0x95	Read Only	Firmware minor version number.
0x96	Read Only	Firmware sub minor version number.
0x97	Read Only	Boot loader algorithm used. 0Xff for no bootloader support.
0x98	Read Only	Buffer size.
0x99-0xcf	---	Reserved for future use.
0xd0-0xdf	Read Only	128-bit (16-byte) globally unique ID (GUID) identifier for the device. This identifier uniquely identifies the device throughout the world and can give additional information on where driver and driver information can be found for the device. MSB for the identifier is stored first (in 0xd0).
0xe0-0xff	Read Only	Module Description File URL. A zero terminates the ASCII string if not exactly 32 bytes long. The URL points to a file that gives further information about where drivers for different environments are located. Can be returned as a zero string for devices with low memory.

It is recommended that unimplemented registers read as oxff

Level II - Register Abstraction Model

The level II abstraction model is the same as Level I but covers a much wider address space. The registers are layed out in an 32-bit address space with the standard Level I register map on the top (0xffffff80 - 0xffffffff).

Level II also have a configuration model where data for a node can be read and written in XML format. Its up to the designed of the node which method to use as long as the required registers are implemented.

Level II – Configuration Model

Preliminary

For the level II part of the protocol a configuration model for nodes has been added. This is a complimentary way to configure a device besides the register model. The manufacturer define what the dataformat for a specifc node looks like.

XML format for configuration messages

```
<config xmlns:xs=http://www.w3.org/2001/XMLSchema>
<name type="xsd:string">Kitchen Light Switch</name>
<idleSeconds type="xsd:integer">10</idleSeconds>
<enabled type="xsd:bool>true</enabled>
</config>
```

No reliance should be placed on the name of the root level element as it's simply used to create a well formed XML message.

The "xmlns:xs" specifies the name space being used and can always be as specified above.

Each parameter for the node is then specified as an element with the "type" attribute specifying the type of value supported for this attribute.

Support will be included for extended type, however, for the moment only simple "xsd" types are supported.

A number of the methods below requires that the GUID of the target node is known beforehand. This can be achieved by sending a "Segment Status Heartbeat" (CLASS.PROTOCOL1, TYPE= 1). Nodes respond to this message by sending a NodeHeartbeat (CLASS.INFORMATION, TYPE=9).

The following is an example of how these messages can be used.

1.**ConfigReadRequest** event is broadcast with the GUID of the target node in the "Data" portion of the message and 0 in the code portion.

2. Node responds with a **ConfigReadResponse** with the code portion indicating the current page / total pages.

3. Requesting node receives all the **ConfigReadResponse** messages and pieces together the message. Note that if a **ConfigReadRequest** is detected whilst compiling the **ConfigReadResponse** message then the process is abandoned.

4. The requesting node can then edit the configuration and then send out a **ConfigChanged** message with the GUID of the target in the data portion. This indicates that the data for another node has been modified and can be sent on request.

5. The target node can now request the new configuration by issuing a **ConfigUpdateRequest**

6.The node that changed the configuration will now notice this message and from the GUID of the sender can determine that it has a new configuration for the node and thus sends out a series of **ConfigUpdateResponse** messages.

7. The target node now waits for a series of **ConfigUpdateResponse** messages with a segment / total segment parameter.

8. The process is ended with a **ConfigChanged** with no GUID in the data portion.

Module Description File (mdf)

Preliminary

The VSCP registers 0xe0-0xff specifies the Module Description File URL (without "http://" which is implied if not specified).

The file is in XML format and defines a modules functionality, registers and events. The intended use is for application software to be able to get information about a node and its functionality in an automated way.

On Level II devices this information can be available in the configuration data and be locally stored on the node.

If <language> tags are missing for a name or a description or in an other place where they are valid English should be assumed.

<vscp> ... </vscp>

This is the top level tag. Data within thees tags are valid. Data outside should be disregarded.

<name>...</name>

Manufacturer Name of devices.

<model>...</model>

Manufacturer model of devices.

<description>...</description>

These tags can be present in several places. At the top level they give a general description of the module and its functionality.

<language>...</language>

Language as an ISO code (same as domain extensions).

<infourl>...</infourl>

Specifies an URL to human readable information pages about the module.

<manufacturer> ... </manufacturer>

This is a tag for a manufacturer information block for the module. One for each region the company may be present in or a single entry (Head Office).

<name> ... </name>

In a manufacturer block this is the name of the manufacturer.

<address> ... </address>

In a manufacturer block this is the postal address of the manufacturer. Several are allowed.

<zipcode> ... </zipcode>
<postalcode> ... </postalcode>

In a manufacturer block this is the zip-code of the manufacturer. Both are valid but only ever one at the time.

<state> ... </state>

In a manufacturer block this is the state of the manufacturer.

<region> ... </region>

In a manufacturer block this is the region of the manufacturer.

<city> ... </city>

In a manufacturer block this is the city of the manufacturer.

<country> ... </country>

In a manufacturer block this is the country of the manufacturer.

<telephone> ... </telephone>

In a manufacturer block this is the telephone number of the manufacturer. Several are allowed.

<description> ... </description>

Description of phone number. “sales”, “info”, “exchange”.

<telefax> ... </telefax>

In a manufacturer block this is the fax number of the manufacturer. Several are allowed.

<email> ... </email>

In a manufacturer block this is the email address of the manufacturer. Several are allowed.

<web> ... </web>

In a manufacturer block this is the web URL of the manufacturer. Several are allowed.

<register> ... </register>

Module register definition.

<pos> ... </pos>

Register position as a decimal or a hexadecimal number. If in hex the number should be preceded with “0x0”. A page can also be defined. In this case the form *reg:page* is used. So 0x10:0x0001 means register 0x10 in page 1.

<name> ... </name>

Register name.

<language>...</language>

Language as an ISO code (same as domain extensions). Name in optional other language

<description> ... </description>

Register description.

<language>...</language>

Language as an ISO code (same as domain extensions). Description for optional other language.

<access> ... </access>

“r” or “w” is valid values. “r” stands for read. “w” stands for write. “rw” is read/write.

<default> ... </default>

Default value for the register as a decimal or a hexadecimal number. If in hex the number should be preceded with “0x0”.

<bit> ... </bit>

Register bit definition.

<pos> ... </pos>

Bit position as a decimal between 0 – 7 .

<name> ... </name>

English bit name.

<language>...</language>

Language as an ISO code (same as domain extensions). Name in optional other language

<description> ... </description>

Bit description

<language>...</language>

Language as an ISO code (same as domain extensions). Description for optional other language

<default> ... </default>

Default value for the bit. 0 or 1.

<access> ... </access>

“r” or “w” is valid values. “r” stands for read. “w” stands for write. “rw” and “w” is read/write.

<event> ... </event>

Specifies an event this device is capable of generating.

<class> ... </class>

VSCP class for event as a decimal or a hexadecimal number. If in hex the number should be preceded with “0x0”.

<type> ... </type>

VSCP type for event as a decimal or a hexadecimal number. If in hex the number should be preceded with “0x0”.

<priority> ... </priority>

VSCP priority for event as a decimal or a hexadecimal number between 0 - 7. If in hex the number should be preceded with “0x0”.

<hardcoded> ... </hardcoded>

VSCP hard coded flag as 0 or 1.

<name> ... </name>

Event name.

<language>...</language>

Language as an ISO code (same as domain extensions). Name in optional other language

<description> ... </description>

Event description.

<language>...</language>

Language as an ISO code (same as domain extensions). Description for optional other language.

<data> .. </data>

Describes one data byte of the event.

<pos> ... </pos>

Data position as a decimal between 0 – 7 .

<name> ... </name>

Name for data byte.

<language>...</language>

Language as an ISO code (same as domain extensions). Name in optional other language

<description> ... </description>

Data byte description.

<language>...</language>

Language as an ISO code (same as domain extensions). Description for optional other language

<bit> ... </bit>

Data bit definition.

<pos> ... </pos>

Bit position as a decimal between 0 – 7 .

<name> ... </name>

Bit name.

<language>...</language>

Language as an ISO code (same as domain extensions). Name in optional other language

<description> ... </description>

Bit description.

<language>...</language>

Language as an ISO code (same as domain extensions). Description for optional other language

<alarm> ... </alarm>

Specifies alarmbits in the alarmstatus register..

<bit> ... </bit>

Status bit definition.

<pos> ... </pos>

Bit position as a decimal between 0 – 7 .

<name> ... </name>

Bit name.

<language>...</language>

Language as an ISO code (same as domain extensions). Name in optional other language

<description> ... </description>

Bit description.

<language>...</language>

Language as an ISO code (same as domain extensions). Description for optional other language

<eda> ... </eda>

Specifies eda capabilities for the node.

<dmrows> ... </dmrows>

The number of rows in the decision matrix as a decimal or a hexadecimal number. If in hex the number should be preceded with "0x0".

<action> ... </action>

Defines an available module specific action.

<code> ... </code>

The code for the action as a decimal or a hexadecimal number between 2 - 63. If in hex the number should be preceded with "0x0".

<name> ... </name>

Action name.

<language>...</language>

Language as an ISO code (same as domain extensions). Name in optional other language

<description> ... </description>

Action description.

<language>...</language>

Language as an ISO code (same as domain extensions). Description for optional other language

If the action can send an event, the event tag can also be defined here.

<bootload> ... </bootload>

Bootloader information

<algorithm> ... </algorithm >

Boot loader algorithm supported. For coding see Class=0, Type=12

<blocksize> ... </blocksize>

Block size for updateable area.

<blockcount> ... </blockcount>

Number of blocks available.

EDA - Message handling and Rules

Every message on a VSCP segment can be seen as an **event**. To be of any use to anyone a **decision** has to be taken on what to do as a result of the event. This is done in a decision-matrix, which is a standardized way to write the rules that take care of events. To get something done when a specific event is detected some sort of **action mechanism** is needed.

The **event-decision-action** model or **eda-model** is the way we look at the functionality of a VSCP-node. First of all a node can be a level I or level II node. A level I node can perform actions from all level I messages. A level II node can handle both level I and level II messages. In fact when we define rules we define them at the highest level.

The eda model

To be able to build a software model that is common for several device/machine/situation types and environments we have make a model to suit the typical control situation. The model is very simple and is also very easy to implement on different target environments.

EVENT: Something happens that triggers some kind of input. This could be the elapse of a timer, a delivered temperature reading, a triggered fire alarm etc. Without this state there is nothing to do. We call this state the EVENT - state.

DECISION: If an event occurs we may react in some way to this happening. This reaction comes after a decision about if and how the event should be handled. We use a decision matrix to control the logic of our decisions. Every element in the decision matrix handles one event and performs one action. A decision matrix element is also capable of generating events and can in this way perform several actions on the behalf of one event.

ACTION: The action is the function, thread, procedure, method or other functionality that should be carried out as a result of an event.

The states represented by **EVENT + DECISION + ACTION** are treated as one transaction. All steps must be handled for the transaction to be marked as completed.

So when you describe the functionality of a VSCP-node you say

- This node reacts on the following events
- This node can perform the following actions.

To make the node a useful piece of equipment it needs a decision matrix. This matrix can be implemented with no possibility for a user to change it or it can be undefined and left for the user to specify.

Decision matrix for Level I nodes.

In resource-limited environments, such as a microprocessor, the previously discussed matrix format takes up to much space and a more resource friendly format is therefore defined. Here the **class + type** bytes formats the event.

This matrix has no trigger for data values. This has to be done internally by the application using user defined codes.

A matrix row consists of 8 bytes and has the following format

byte 0 – byte 7

| oaddr | flags | class-mask | class-filter | type-mask | type-filter | action | action-param |

oaddr

oaddr is the originating address. We are only interested in messages from the node given here. 0x00 is segment controller and 0xff is a node without a nickname. If bit 6 of flags are set oaddr will not be checked and events from all nodes will be accepted.

flag

bit 7	Enabled if set to 1.
bit 6	oaddr should be checked (=1) or not checked (=0)
bit 5	Indicates hardcoded originating address
bit 4	Reserved
bit 3	Reserved
bit 2	Reserved
bit 1	Class-mask bit 8.
bit 0	Class-filter bit 8.

The enable bit can be used to disable a decion matrix row while it is edited.

class-mask / class-filter

A class that should trigger this decision matrix row is defined in class-mask and class-filter with bit eight for both taken from the flags byte.

The following table illustrates how this works

Mask bit n	***Filter bit n***	***Incoming event class bit n***	***Accept or reject bit n***
0	x	x	Accept
1	0	0	Accept
1	0	1	Reject
1	1	0	Reject
1	1	1	Accept

So to only accept one class zset all mask bits to one and enter the class in filter.

type-mask / type-filter

A type that should trigger this decision matrix row is defined in type-mask and type-filter with bit eight for both taken from the flags byte.

A similar truth table as for the class-mask/filter is used.

action

This is the action or operation that should be performed if the filtering is satisfied. Only action code 0x00 is predefined and means No-Operation. All other codes are application specific and typical application defined codes could be do measurement, send predefined event etc

action-parameter

A numeric action can be set and its meaning is application specific.

The number of matrix rows are limited in a micro controller. The control class has an event defined that gets the number of elements in the matrix and the location of the matrix in the register model of the node (Get EDA matrix info, CLASS1.PROTOCOL, Type=33).

The matrix information is read and written with the standard read/write control functions. And is placed in the standard low register range (< 0x80) or in an optional page in this are.

Note that there is no demand that a node implements a decision matrix. If not implemented the Get EDA matrix info just returns a zero size.

A special paged feature is available for the decision matrix to save register space. If the offset for the decision matrix is 0x80 -8 i.e. Starts at 0X78 (120) it is implied that the register position just below 0x77(119) contains a register that is an index to the row that is available in the decision matrix space.

Method CLASS1.PROTOCOL TYPE=32 is used to fetch decision matrix information for a specific node.

General

It is important to note that the decision matrix can contain any number of lines for a specific event element. So one incoming event can trigger many actions.

Events are marked as handled when the action thread is started. This can be bad in some situations where the event chain is dependent on the action to complete its task before any new work is done. In this case it is better to continue the event chain by posting an event from the action thread instead of setting the following event as a next event in the decision matrix.

The decision matrix structure gives us the freedom to implement events that perform

- Exactly one action.
- Several actions.
- Several actions in a sequence.

Decision matrix for Level II nodes.

To understand how the decision matrix is updated one needs to understand how it is set up. Each line of the matrix is built from a table of entries of the following form:

Mask (32-bit)

This is a bit mask, which has ones at the bit positions of the event that is interesting. So 0xffffffff makes all messages interesting. Class is in the high word. Type is in the low word. For a truth table see the class-mask for Level I decision matrices.

Filter (32-bit)

This is a bit mask, which tells the vaule for bits that is of iinterest. Class is in the high word. Type is in the low word. For a truth table see the class-mask for Level I decision matrices.

Control(32-bit)

- Bit 31 - Enabled (if == 1). Indicates if this element in the decision matrix is enabled or disabled. If disabled the decision matrix element is ignored.
- Bit 30 -- Bit 0 Reserved

Action (16-bit)

This is the what should be carried out to make this action happen. This is a 16-bit cod that is application specific. 0X0000 is the only code that is predefined as no operation.

Action Parameters

This is a variable length text-string/binary array that can contain parameters for the action. Format is application dependent. Can be set omitted if no parameters are used.

A decision matrix row is 14 bytes plus the size of the action parameters.

A special paged feature is available for the decision matrix to save register space. If the offset for the decision matrix is 0x80 -dm-row-size it is implied that the register position just below 0x80 -dm-row-size - 1 contains a register that is an index to the row that is available in the decision matrix space.

Method CLASS1.PROTOCOL TYPE=32 is used to fetch decision matrix information for a specific node. Byte six of the response tells the actual size for a decision matrix row.

VSCP boot loader algorithm

This is the VSCP boot loader. Note that several other low level boot loader mechanisms that are more suited for low memory footprint devices are available.

Many devices of today have the capability for in circuit programming and can have their firmware changed whilst still in the system. This is supported by VSCP with boot loader events.

Most flash devices are programmed block by block. The boot loader algorithm must support this. Blocks are usually quite small but to be compatible with future devices 16-bit words are used to describe a block size. 16-bit words are also used to describe the number of available blocks. This means that the total Flash size is described by a 32-bit word and the maximum flash size supported is thus four gigabytes.

The boot loader sequence is as follows:

1.) The master instructs the node to enter boot loader mode by sending an **enter boot loader mode** message to the node. A unit now can use the VSCP boot loader which is described here or another boot loader.
2.) The node confirms that it is ready for code loading by sending the **ACK boot loader mode** message (a **NACK boot loader mode** message is also possible). Block size and number of blocks are sent as arguments in the acknowledge message.
3.) The master sends a **start block data transfer** message to specify which block should be programmed and to initiate the transfer of data for the block.
4.) The master sends one or several **block data** messages until the complete block is transferred.
5.) When all data for a block is received by a node it sends a **confirm block** message to acknowledge the reception of a complete block.
6.) The master now sends a **program block** message to the node to make it write the block buffer into flash memory.
7.) The node confirms the block programming by responding with an **ACK program block** message.
8.) The next block is handled or the node is taken out of the boot loader mode by sending a **drop nickname/reset device** message.
9.) To activate the new program code the boot loader sends an **activate new image** event with the 16 bit CRC for the new block as an argument. The new node should come up after reboot.

The 16-bit CCITT CRC is used.

The bootloader is built to direct control flash if other methods such as intermediate storage is used. Data can be loaded direct and **program block** can just get a dummy ACK.

Data coding

All data is sent in a form that is related to the default format of the data. The number of data bytes in the frame also reflects the size of the variable.

In this definition there is a very important assumption. If two nodes should be able to talk to each other they have to know each others data formats. So our assumption is that if a node is interested in what another node has to say it must learn its data format. Also if a node needs to control another node it has to learn its data format to do so.

As a guideline the format defined bellow for the first data byte of a data frame can be used but if a user likes to use another format it is perfectly fine to do so.

Byte 0 – Tells how data that follows should be interpreted. This is used for CLASS1.MEASUREMENT

Bits	Description
5,6,7	Represent one of several numerical representations in which the data that follows is represented in: **<u>000b Bit Format</u>** The data should be represented as a set of bits. This can be used for picture coding etc. **<u>001b Byte Format(0x20)</u>** The data should be represented as a set of bytes. **<u>010b String Format(0x40)</u>** The data should be represented as an ASCII string (possibly with language extensions (8-bit)). **<u>011b Integer Format(0x60)</u>** Data is coded as an integer. The integer is coded in the bytes that follows and can be 1-7 bytes where the most significant byte always is in byte 1 (big endian). If **DLC=2** the data is a 1 byte integer or 1 byte. If **DLC=3** the data is a 16-bit integer or 2 bytes. If **DLC=4** the data is a 24-bit integer or 3 bytes. If **DLC=5** the data is a 32-bit integer or 4 bytes. If **DLC=6** the data is a 40-bit integer or 5 bytes. If **DLC=7** the data is a 48-bit integer or 6 bytes. If **DLC=8** the data is a 56-bit integer or 7 bytes. **<u>100b Normalized Integer Format(0x80)</u>** Data is coded as a normalized integer. Byte 1 is the normalizer byte which have bit 7 set for a negative value.. The test of the value is coded in the bytes that follows and can be 1-6 bytes where the most significant byte always is in byte 1 (big endian). The normalizer bytes lower six bits tells how many position the decimal point should be shifted left or right. If it has a negative value the decimal point should be shifted in the left direction and if it has a positive value in the right direction. **<u>101b Floating point value(0xA0)</u>** Data is coded as a IEEE-754 1985 floating point value **s eeeeeeee mmmmmmmmmmmmmmmmmmmmmmm** **s** = sign bit(1-bit) **e** = exponent (8-bit) **m** = mantissa (23-bit) That is a total of 32-bits. The most significant byte is stored first. The frame holds a total of five bytes. The full definition is at http://www.psc.edu/general/software/packages/ieee/ieee.html **<u>110b Reserved.(0xC0)</u>** The format is yet to be defined. **<u>111b Reserved(0xE0).</u>** The format is yet to be defined.
3,4	Indicates how the data should be interpreted: **000b** Standard format. All other codes in this field are message class/type specific.
0,1,2	This is the sensor index if there are more then one sensor handled by the node.

Class Definitions Level I

Class = 0 (0x00) VSCP Protocol functionality

CLASS1.PROTOCOL

Description

This class defines some types that must be implemented by every node that implements the VSCP protocol. The types in this class must be handled by all level I and Level II nodes.

Note also that this class is repeated as Level II class=512 whot the only dfifference that GUID's are used instead of nicknames.

Type = 0 (0x00) Undefined.

Undefined protocol function.

Type = 1 (0x01) Segment Status Heartbeat.

A segment controller sends this message once a second on the segment that it controls. The data field contains the 8-bit CRC of the segment controller GUID and the time since the Epoch (00:00:00 UTC, January 1, 1970) as a 32-bit value.

Also other nodes can originate this event on the network. For these nodes the data part should be omitted.

All nodes should save the 8-bit CRC in non-volatile storage and use it on power up. When a node starts up on a segment it should begin to listen for the **Segment controller heartbeat.** When/if it is received the node compares it with the stored value and if equal and the node is assigned a nickname id it continues to its working mode. If different, the node has detected that it has been moved to a new segment and therefore must drop its nickname id and enters the configuration mode to obtain a new nickname id from the segment controller.

If the node is in working mode and it's nickname id changes, the node should do a complete restart after first setting all controls to their default state.

As a segment can be without a segment controller this message is not available on all segments.

Byte 0	8-bit CRC.
Byte 1	MSB of time since Epoch (optional).
Byte 2	Time since Epoch (optional).
Byte 3	Time since Epoch (optional).
Byte 4	LSB of time since Epoch (optional).

Uninitiated nodes have the CRC of the segment controller set to **0xff**.

A node that is initialized on a segment and does not receive a Heartbeat can take the role of segment controller if it wishes to do so. Only one node one a segment are allowed to do this fully by setting its nickname=0 and therefore a standard node should not have this feature built in. Any node can however behave like a segment controler but use a nickname other then zero.

Type = 2 (0x02) New node on line / Probe.

This is intended for nodes that have been initiated, is part of the segment and is powered up. All nodes that have a nickname id that is not set to **0xff** should send this message before they go on line to do their “day to day” work.

Normally all nodes should save their assigned nickname id in nonvolatile memory and use this assigned id when powered up. A segment controller can however keep track of nodes that it controls and reassign the id to a node that it did not get a **new node online** message from. This is the method a segment controller uses to detect nodes that have been removed from the segment.

For the nickname discovery procedure this event is used as the probe. The difference between a probe and a new node on line is that the later has the same originating nickname as value in byte 0.

It is recommended that also level II nodes send this message when they come alive.

Byte 0 - Target address - If specified this is a probe message that the new node is using to test if this is a valid target node. If there is a node with this nickname address it should answer with probe ack

Type = 3 (0x03) Probe ACK.

This message is sent from a node as a response to a probe. There are no arguments.

Type = 4 (0x04) Nickname Failure.

Sent to inform other nodes that a node failed to find a free nickname and will go off-line.

Type = 5 (0x05) Reserved for future use.

Type = 6 (0x06) Set Nickname id for node.

The message contains the new nickname id for a node that is in the initializing state.

Byte 0 Old nickname for node. This is normally 0xff.
Byte 1 The nickname for the node.

Type = 7 (0x07) Nickname id accepted.

A node sends this message to confirm that it accepts its assigned nickname id. When sending this message the node uses its newly assigned nickname address.

Type = 8 (0x08) Drop nickname id / Reset Device.

Request a node to drop its nickname. The node should drop its nickname and then behave in the same manner as when it was first powered up on the segment.

Byte 0 The current nickname for the node.

Type = 9 (0x09) Read register.

Read a register from a node.

Byte 0 Node address.
Byte 1 Register to read.

A read/write response message is returned on success.

Note for event on a Level II network

Byte 0 – byte 15	GUID
Byte 16	Register to read

Type=10 (0x0A) Read/Write response.

Response for a read/write message. Note that the origin is given from the CAN header information. Note that the data is returned for both a read and a write and can be checked for validity.

Byte 0 Register read/written.
Byte 1 Content of register.

Type = 11 (0x0B) Write Register.

Write register content to a node.

Byte 0	Node address.
Byte 1	Register to write.
Byte 2	Content for register.

A read/write response message is returned on success.

Note for event on a Level II network

Byte 0 – byte 15	GUID
Byte 16	Register to write.
Byte 17	Content of register.

Type = 12 (0x0C) Enter boot loader mode.

This is the first package in the boot loader sequence. The node should stop all other activities when in boot loader mode.

Byte 0 The nickname for the node.
Byte 1 Code that select boot loader algorithm to use

0	VSCP algorithm.
1	Microchip PIC algorithm 0 (VSCP PIC Boot loader)
2	Microchip PIC algorithm 1
3	Microchip PIC algorithm 2
4	Microchip PIC algorithm 3
5	Microchip PIC algorithm 4
6	Microchip PIC algorithm 5
7	Microchip PIC algorithm 6
8	Microchip PIC algorithm 7
9	Atmel AVR algorithm 0
10	Atmel AVR algorithm 1
11	Atmel AVR algorithm 2
12	Atmel AVR algorithm 3
13	Atmel AVR algorithm 4
14	Atmel AVR algorithm 5
15	Atmel AVR algorithm 6
16	Atmel AVR algorithm 7
17-255	Reserved.

Byte 2 – GUID byte 0
Byte 3 – GUID byte 3
Byte 4 – GUID byte 5
Byte 5 – GUID byte 7
Byte 6 – Content of register 0x92, Page select MSB
Byte 7 – Content of register 0x93, Page select LSB

Note for event on a Level II network

Byte 0 – byte 15	GUID
Byte 16	Bootloader algorithm code.

Type = 13 (0x0D) ACK boot loader mode.

This message has no meaning for any node that is not in boot mode and should be disregarded.

The node confirms that it has entered boot loader mode. This is only sent for the VSCP boot loader algorithm.

Byte 0	MSB of flash block size.
Byte 1	Flash block size.
Byte 2	Flash block size.
Byte 3	LSB of flash block size.
Byte 4	MSB of number of block s available.
Byte 5	Number of block s available.
Byte 6	Number of block s available.
Byte 7	LSB of number of blocks available.

Type = 14 (0x0E) NACK boot loader mode.

This message has no meaning for any node that is not in boot mode and should be disregarded.

The node was unable to enter boot loader mode. The reason is given by a user specified error code byte.

Byte 0	User defined error code.

Type = 15 (0x0F) Start Block Data Transfer.

This message has no meaning for any node that is not in boot mode and should be disregarded.

Begin transfer of data for a block of memory.

Byte 0	MSB of block number.
Byte 1	Block number.
Byte 2	Block number.
Byte 3	LSB of block number.

Type = 16 (0x10) Block Data.

This message has no meaning for any node that is not in boot mode and should be disregarded.

Data for a block of memory.

Byte 0	Data
Byte 1	Data
Byte 2	Data
Byte 3	Data
Byte 4	Data
Byte 5	Data
Byte 6	Data
Byte 7	Data

Type = 17 (0x11) ACK Data Block.

This message has no meaning for any node that is not in boot mode and should be disregarded.

Confirm the reception of a complete data block.

Byte 0	MSB of 16-bit CRC for block.
Byte 1	LSB for 16-bit CRC for block.
Byte 2	MSB of write pointer.
Byte 3	write pointer.
Byte 4	write pointer.
Byte 5	LSB of write pointer.

The write pointer is the actual pointer after the last data has been written i,e the next position on which data will be written.

Type = 18 (0x12) NACK Block Data.

This message has no meaning for any node that is not in boot mode and should be disregarded.

NACK the reception of data block.

Byte 0	User defined error code.
Byte 1	MSB of write pointer.
Byte 2	write pointer.
Byte 3	write pointer.
Byte 4	LSB of write pointer.

The write pointer is the actual pointer after the last data has been written i,e the next position on which data will be written.

Type = 19 (0x13) Program Data Block

This message has no meaning for any node that is not in boot mode and should be disregarded.

Request from a node to program a data block that has been uploaded and confirmed.

Byte 0 MSB of block number.
Byte 1 Block number.
Byte 2 Block number.
Byte 3 LSB of block number.

Type = 20 (0x14) ACK Program Data Block

This message has no meaning for any node that is not in boot mode and should be disregarded.

A node confirms the successful programming of a block.

Byte 0 MSB of block number.
Byte 1 Block number.
Byte 2 Block number.
Byte 3 LSB of block number.

Type = 21 (0x15) NACK Program Data Block

This message has no meaning for any node that is not in boot mode and should be disregarded.

A node failed to program a data block.

Byte 0 User defined error code.
Byte 1 MSB of block number.
Byte 2 Block number.
Byte 3 Block number.
Byte 4 LSB of block number.

Type = 22 (0x16) Activate new image

This message has no meaning for any node that is not in boot mode and should be disregarded.

This command is sent as the last command during the bootloader sequence. It resets the device and starts it up using the newly loaded code. The 32-bit CRC for the entire program block is sent as an argument. This must be correct for the reset/activation to be performed. **NACK boot loader mode** will be sent if the CRC is not correct and the node will leave boot loader mode.

Byte 0	**16** bit CRC of full flash data block, **MSB**
Byte 1	**16** bit CRC of full flash data block **LSB**

To leave boot mode just send this event and a dummy crc.

Other methods could have been used to load the code but it can still be activated with this event as long as the crc is correct.

Type = 23 (0x17) Reserved for future use

reserve

Type = 24 (0x18) Page Read

The page read is implemented to make it possible to read/write larger blocks of data on a node that supports this. Two register positions are reserved to select a base into this storage. This is a 16-bit number pointing to a 256-byte page. This means that a total of 65535 * 256 bytes are accessible with this method (page 0 is the standard registers).

To read a block of data from the storage, first write the base registers then issue this event and n events will be sent out from the node containing the data from the specified area. If the count pass the border it of the page (> 0xff) the transfer will end there.

Note that the page select registers only selects a virtual page that can be accessed with page read/write and not with the ordinary read/write.

Byte 0	Index into page.
Byte 1	Number of bytes to read.

Note for event on a Level II network

Byte 0 – byte 15	GUID
Byte 16	Index into page.
Byte 17	Number of bytes to read.

Type = 25 (0x19) Page Write

The write page is implemented to make it possible to write larger blocks of data on a node that supports this. Two register positions are reserved to select a base into this storage. See Page read for a full description.

It is only possible to write one 7-byte page at a time in contrast to reading several. This is because VSCP at Level I is aimed at low end devices with limited resources meaning little room for buffers.

Byte 0	Base index.
Byte 1 – 7	Data.

Note for event on a Level II network

Byte 0 – byte 15	GUID
Byte 16	Base index.
Byte 17 -	Data.

Data count can be as many as the buffer of the Level II node accepts.

Type = 26 (0x1A) Read Page Response

This is a response frame for the read page command.

Byte 0	Sequency number.
Byte 1 – 7	Data.

Note for event on a Level II network

Byte 0 – byte 15	GUID
Byte 16	Sequency number.
Byte 17 –	Data.

Data count can be as many as the buffer of the Level II node accepts.

Type = 27 (0x1B) Reserved for future use

reserved

Type = 28 (0x1C) Reserved for future use

reserved

Type = 29 (0x1D) Reserved for future use

reserved

Type = 30 (0x1E) Reserved for future use

reserved

Type = 31 (0x1F) Reserved for future use

reserved

Type = 32 (0x20) Get EDA matrix info

Request a node to report size and offset for its EDA matrix.

Byte 0 Node address.

Note for event on a Level II network

Byte 0 – byte 15 **GUID**

Type = 33 (0x21) EDA matrix info response

Report the size for the EDA matrix and the offset to its storage. The reported size is the number of decision matrix lines. The offset is the offset in the register address counter from 0x00 (See the register model in this document). If the size returned is zero the node does not have a decision matrix.

Byte 0 Matrix size (number of rows).
Byte 1 Offset in register space.
Byte 2 Optional page start MSB (Interpret as zero if not sent)
Byte 3 Optional page start LSB (Interpret as zero if not sent)
Byte 4 Optional page end MSB (Interpret as zero if not sent)
Byte 5 Optional page end LSB (Interpret as zero if not sent)
Byte 6 For a Level II node this is the size of a decision matrix row.

The decision matris can as noted be stored in paged registers and if so it must be accessd with the paged read/write.

Level I

If the offset given is 0x80-8 = 0x78 the matrix is paged and it is implied that rows are selected in a register located at 0x80-8 – 1 = 0x77. 0 means row 1 is in the register space, 1 = row 2 is in register space and so on.

Note for event on a Level II network

The same mapping can be used for Level II but the location of the page register is dependent on the row size. Thus a register at 0x80 -dm-row-size - 1 is an index to the row that is available in the decision matrix space.

Class = 1(0x01) Alarm

CLASS1.ALARM

Description

Alarm events that indicate that someting that s not ordinary has occured. Note that the priority bits can be used as a mean to level alarm for severety.

Type = 0 (0x00) Undefined

Undefined alarm.

Type = 1 (0x01) Warning .

Indicates a warning condition. The data is user defined.

Type = 2 (0x02) Alarm occurred.

Indicates an alarm condition. The data is user defined.

Type = 3 (0x03) Alarm sound on/off.

Alarm sound should be turned on or off.

Byte 0 If equal to zero turn off else turn on.

Type = 4 (0x04) Alarm light on/off.

Alarm light should be turned on or off.

Byte 0 If equal to zero turn off else turn on.

Type = 5 (0x05) Power on/off

Power has been lost or is available again.

Byte 0 If equal to zero power is unavailable else available.

Type = 6 (0x06) Emergency Stop

Emergency stop has been hit/activated. All systems on the segment should go to their inactive/safe state.

Type = 7 (0x07) Emergency Pause

Emergency pause has been hit/activated. All systems on the segment should go to their inactive/safe state but preserve there settings.

Type = 8 (0x08) Emergency Reset

Issued after an emergency stop or pause in order for nodes to reset and start operating .

Type = 9 (0x09) Emergency Resume

Issued after an emergency pause in order for nodes to start operating from where they left of without resetting their registers .

Class = 2 (0x02) Security

CLASS1.SECURITY

Description

Security related events for alarms and simular devices.

Type = 0 (0x00) undefined

Undefined security issue.

Class = 10 (0x0A) Measurement

CLASS1.MEASUREMENT

Description

Byte 0 is the data coding byte for all measurement packages. The default unit has bits 0,1,2,3 set to 0000 and the first optional unit 0001 and so on. It also have a field for the item (if more than one sensor is controlled by the node) that the value belongs to.

Type = 0 (0x00) Undefined

Undefined measurement value.

Type = 1 (0x01) Count

This is a discrete value typical for a count. There is no unit for this measurement just a discrete value.

Byte 0 data coding.

Type = 2 (0x02) Length/Distance

Default unit: Meter.
This is a measurement of a length or a distance.

Byte 0 data coding.

Type = 3 (0x03) Mass

Default unit: Kilogram.
This is a measurement of a mass.

Byte 0 data coding.

Type = 4 (0x04) Time

Default unit: Millisecond.
Opt. unit: Second(1), Absolute: y-y-m-d-h-m-s (binary)(2),String HHMMSS (3), Time since Epoch (00:00:00 UTC, January 1, 1970).

Byte 0 data coding.

Type = 5 (0x05) Electric Current

Default unit: Ampere.
This is a measurement of an electric current.

Byte 0 data coding.

Type = 6 (0x06) Temperature

Default unit: Kelvin.
Opt. unit: Degree Celsius (1), Fahrenheit (2)
This is a measurement of a temperature.

Byte 0 data coding.

Type = 7 (0x07) Amount of substance

Default unit: Mole.
This is a measurement of an amount of a substance.

Byte 0 data coding.

Type = 8 (0x08) Intensity of light

Default unit: Candela.
This is a measurement of luminous intensity.

Byte 0 data coding.

Type = 9 (0x09) Frequency

Default unit: Hertz.
This is a measurement of regular events during a second.

Byte 0 data coding.

Type = 10 (0x0A) Radioactivity and other random events

Default unit: Becquerel.
This is a measurement of rates of things, which happen randomly, or are unpredictable.

Byte 0 data coding.

Type = 11 (0x0B) Force

Default unit: Newton.
This is a measurement of force.

Byte 0 data coding.

Type = 12 (0x0C) Pressure

Default unit: Pascal.
This is a measurement of pressure.

Byte 0 data coding.

Type = 13 (0x0D) Energy

Default unit: Joule.
This is a measurement of energy.

Byte 0 data coding.

Type = 14 (0x0E) Power

Default unit: Watt.
This is a measurement of power.

Byte 0 data coding.

Type = 15 (0x0F) Electrical Charge

Default unit: Coulomb.
This is a measurement electrical charge.

Byte 0 data coding.

Type = 16 (0x10) Electrical Potential

Default unit: Volt.
This is a measurement of electrical potential.

Byte 0 data coding.

Type = 17 (0x11) Electrical Capacitance

Default unit: Farad.
This is a measurement of electric capacitance.

Byte 0 data coding.

Type = 18 (0x012) Electrical Resistance

Default unit: Ohm.
This is a measurement of resistance.

Byte 0 data coding.

Type = 19 (0x13) Electrical Conductance

Default unit: Siemens.
This is a measurement of electrical conductance.

Byte 0 data coding.

Type = 20 (0x14) Magnetic Field Strength

Default unit: Ampere meters.
This is a measurement of magnetic field strength.

Byte 0 data coding.

Type = 21 (0x15) Magnetic Flux

Default unit: Weber.
This is a measurement of magnetic flux.

Byte 0 data coding.

Type = 22 (0x16) Magnetic Flux Density

Default unit: Tesla.
This is a measurement of flux density or field strength for magnetic fields (also called the magnetic induction).

Byte 0 data coding.

Type = 23 (0x17) Inductance

Default unit: Hendry.
This is a measurement of inductance.

Byte 0 data coding.

Type = 24 (0x18) Flux of light

Default unit: Lumen.
This is a measurement of flux of light.

Byte 0 data coding.

Type = 25 (0x19) Iluminance

Default unit: Lux.
This is a measurement of luminance (illumination)

Byte 0 data coding.

Type = 26 (0x1A) Radiation dose

Default unit: Gray.
Opt unit: Sievert.
This is a measurement of a radiation dose.

Byte 0 data coding.

Type = 27 (0x1B) Catalytic activity

Default unit: Katal.
This is a measurement of catalytic activity used in biochemistry.

Byte 0 data coding.

Type = 28 (0x1C) Volume

Default unit: stere, square meter.
Opt. Unit: Liter (dm3).
This is a measurement of volume.

Byte 0 data coding.

Type = 29 (0x1D) Sound intensity

Default unit: Bel.
Opt Unit: Neper.
This is a measurement of sound intensity.

Byte 0 data coding.

Type = 30 (0x1E) Angle

Default unit: Radian (Plane angles).
Opt Unit: Steradian (Solid angles)
Opt Unit: Degree
Opt Unit: Arcminute
Opt Unit: Arcseconds
This is a measurement of an angle.

Byte 0 data coding.

Type = 31 (0x1F) Position

Default unit: Longitude/Latitude.
This is a measurement of a position.

Byte 0 data coding.

Type = 32 (0x20) Speed

Default unit: Meters per second.
This is a measurement of a speed.

Byte 0 data coding.

Type = 33 (0x21) Acceleration

Default unit: Meters per second/second.
This is a measurement of acceleration.

Byte 0 data coding.

Type = 34 (0x22) Tension

Default unit: N/m.
This is a measurement of tension.

Byte 0 data coding.

Type = 35 (0x23) Damp/moist (Hygrometer reading)

Default unit: Relative percentage 0-100%.
This is a measurement of relative moistness.

Byte 0 data coding.

Type = 36 (0x24) Flow

Default unit: Square meters/second.
Opt Unit: Liter/Second.
This is a measurement of flow.

Byte 0 data coding.

Type = 37 (0x25) Thermal resistance

Default unit: Thermal ohm K/W.
This is a measurement of thermal resistance.

Byte 0 data coding.

Type = 38 (0x26) Rafractive power

Default unit: Diopter (dpt) m^{-1}.
This is a measurement of refractive power.

Byte 0 data coding.

Type = 39 (0x27) Dynamic viscosity

Default unit: Poiseuille (Pl) Pa . s.
This is a measurement of dynamic viscosity.

Byte 0 data coding.

Type = 40 (0x28) Sound impedance

Default unit: Rayal Pa . s/m.
This is a measurement of sound impedance.

Byte 0 data coding.

Type = 41 (0x29) Sound resistance

Default unit: Acoustic ohm Pa . s/ m^3.
This is a measurement of refractive power.

Byte 0 data coding.

Type = 42 (0x2A) Electric elastance

Default unit: Darag F^{-1}.
This is a measurement of electric elasticity.

Byte 0 data coding.

Type = 43 (0x2B) Luminous energy

Default unit: Talbot lm . s.
This is a measurement of luminous energy.

Byte 0 data coding.

Type = 44 (0x2C) Luminance

Default unit: Nit (nt) cd/m^2.
This is a measurement of luminance.

Byte 0 data coding.

Type = 45 (0x2D) Chemical concentration

Default unit: Molal mol/kg.
This is a measurement of chemical concentration.

Byte 0 data coding.

Type = 46 (0x2E) Absorbed dose of ionizing radiation

Default unit: Gray J/Kg.
This is a measurement of absorbed dose of ionizing radiation.

Byte 0 data coding.

Type = 47 (0x2F) Dose equivalent

Default unit: Sievert J/Kg.
This is a measurement of dose equivalent.

Byte 0 data coding.

Type = 48 (0x30) Light flux

Default unit: Lumen cd . sr.
This is a measurement of light flux.

Byte 0 data coding.

Type = 49 (0x31) Dew Point

Default unit: Kelvin.
Opt. unit: Degree Celsius (1), Fahrenheit (2)
This is a measurement of the Dew Point.

Byte 0 data coding.

Class = 15 (0x0F) Data

Description

Representation for different general data types. Byte 0 is the data coding byte for all data packages. The default unit has bits 0,1,2,3 set to 0000 and the first optional unit 0001 and so on.

Type = 0 (0x00) Undefined

General Message.

Type = 1 (0x01) I/O – value

General I/O value. First data byte defines format.

Byte 0 data coding.

Type = 2 (0x02) A/D value

General A/D value. First data byte defines format.

Byte 0 data coding.

Type = 3 (0x03) D/A value

General D/A value. First data byte defines format.

Byte 0 data coding.

Type = 4 (0x04) Relative strength

Relative strength is a value that has its maximum at 255 and minimum at 0 if the data part is one byte. If the data part is two bytes the minimum strength is still at zero but the maximum strength is at 65535.

Byte 0 data coding.

Class = 20 (0x14) Information

CLASS1.INFORMATION

Description

Most of the messages below have an index parameter that can be used to indicate which of several SECO (sensor/control) units on a node originated the event. Set to zero if the node only control one item.

Type = 0 (0x00) Undefined

This is a general event of no special type.

Type = 1 (0x01) Button

A button has been pressed/released.

Byte 0	index.
Byte 1	Bits 0,1,2 If 0 the button has been released. If 1 the button is pressed. If equal to 2 this is a key value (press followed by release). Bits 3-7 is repeats 0-32.
Byte 2	MSB of code for button.
Byte 3	LSB of code for button.
Byte 4	LSB of optional code-page.
Byte 5	MSB of optional code-page.

Type = 2 (0x02) Mouse

A mouse movement has occurred.

Byte 0	index.
Byte 1	If zero absolute coordinates follow. If one relative coordinates follow.
Byte 2	MSB of normalized X-coordinate.
Byte 3	LSB of normalized X-coordinate.
Byte 4	MSB of normalized Y-coordinate.
Byte 5	LSB of normalized Y-coordinate.

Type = 3 (0x03) On

A node indicates that a condition is in its on state. Heater on, lights on are two examples.

Byte 0 index.

If more data is supplied it is user defined.

Type = 4 (0x04) Off

A node indicates that a condition is in its off state. Heater off, lights off are two examples.

Byte 0 index.
If data is supplied it is user defined.

Type = 5 (0x05) Alive

A node tells the world that it is alive.

Byte 0 index.
If data is supplied it is user defined.

Type = 6 (0x06) Terminating

A node tells the world that it is terminating.

Byte 0 index.
If data is supplied it is user defined.

Type = 7 (0x07) Opened

A node indicates that am open has occurred. This can be a door/window open, a modem line open etc.

Byte 0 – index.
If data is supplied it is user defined.

Type = 8 (0x08) Closed

A node indicates that a close has occurred. This can be a door/window close, a modem line closure etc.

Byte 0 index.
If data is supplied it is user defined.

Type = 9 (0x09) Node Heartbeat

Heartbeats can be used to indicate that a unit is alive or to send periodic data.

This can be sent out at predefined intervals to indicate that the node is alive, however, it does not necessarily mean the node is functioning as it should. It simply states that the node is connected to the network. To check if a node is functioning, other properties such as a measurement event or registry should be used.

This event should be sent as a response to a "Segment Status Heartbeat" (class=0, Type=1) in order to provide a method of finding out what is connected to the network.

The data bytes can be used to send a descriptive/user friendly name if desired

Not mandatory.

Byte 0 index.
If data is supplied it is user defined.

Type = 10 (0x0A) Below limit

This indicates that the node has a condition that is below a configurable limit.

Byte 0 index.
If data is supplied it is user defined.

Type = 11 (0x0B) Above limit

This indicates that the node has a condition that is above a configurable limit.

Byte 0 index.
If data is supplied it is user defined.

Type = 12 (0x0C) Pulse

This can be used for slow pulse counts. This can be an S0-pulse interface or something similar.

Byte 0 index.
If data is supplied it is user defined.

Type = 13 (0x0D) Error

A node indicates that an error occurred.

Byte 0 index.
If data is supplied it is user defined.

Type = 14 (0x0E) Resumed

A node indicates that it has resumed operation.

Byte 0 index.
If data is supplied it is user defined.

Type = 15 (0x0F) Paused

A node indicates that it has paused.

Byte 0 index.
If data is supplied it is user defined.

Type = 16 (0x10) Reserved

Reserved for future use.

Type = 17 (0x11) Good morning

The system should enter its morning state. This can be a user pressing a button to set his/her house to it's morning state.

Byte 0 index.
If data is supplied it is user defined.

Type = 18 (0x12) Good day

The system should enter its day state. This can be a user pressing a button to set his/her house to it's day state.

Byte 0 index.
If data is supplied it is user defined.

Type = 19 (0x13) Good afternoon

The system should enter its afternoon state. This can be a user pressing a button to set his/her house to it's afternoon state.

Byte 0 index.
If data is supplied it is user defined.

Type = 20 (0x14) Good evening

The system should enter its evening state. This can be a user pressing a button to set his/her house to it's evening state.

Byte 0 index.
If data is supplied it is user defined.

Type = 21 (0x15) Good night

The system should enter its night state. This can be a user pressing a button to set his/her house to it's night state.

Byte 0 index.
If data is supplied it is user defined.

Type = 22 (0x16) See you soon

The system should be on a temporary alert. This can be a user locking the door to go out to the waste bin or similar situation. An alarm system should not be activated in this situation.

Byte 0 index.
If data is supplied it is user defined.

Type = 23 (0x17) Goodbye

The system should be on a goodbye alert. This can be a user locking the door to go out for a day's work or similar situation. All alarm systems should be activated in this situation.

Byte 0 index.
If data is supplied it is user defined.

Type = 24 (0x18) Stopped

A node indicates that a stop event occurred. This can for example be a motor stopping.

Byte 0 index.
If data is supplied it is user defined.

Type = 25 (0x19) Started

A node indicates that a start event occurred. This can be a motor starting.

Byte 0 index.
If data is supplied it is user defined.

Type = 26 (0x1A) ResetCompleted

A node indicates that a reset occurred. This can be a node doing a warm start.

Byte 0 index.
If data is supplied it is user defined.

Type = 27 (0x1B) Interrupted

A node indicates that a reset occurred. This can also be a node doing a warm start.

Byte 0 index.
If data is supplied it is user defined.

Type = 28 (0x1C) PreparingToSleep

A node indicates that a sleep event occurred. This can be a node going to its inactive state.

Byte 0 index.
If data is supplied it is user defined.

Type = 29 (0x1D) WokenUp

A node indicates that a wakeup event occurred. This can be a node going to it awake state.

Byte 0 index.
If data is supplied it is user defined.

Type = 30 (0x1E) Dusk

A node indicates that the system should enter its dusk state.

Byte 0 index.
If data is supplied it is user defined.

Type = 31 (0x1F) Dawn

A node indicates that the system should enter its dawn state.

Byte 0 – index.
If data is supplied it is user defined.

Type = 32 (0x20) Active

A node indicates that its active.

Byte 0 index.
If data is supplied it is user defined.

Type = 33 (0x21) Inactive

A node indicates that its inactive.

Byte 0 index.
If data is supplied it is user defined.

Type = 34 (0x22) Busy

A node indicates that its busy.

Byte 0 index.
If data is supplied it is user defined.

Type = 35 (0x23) Idle

A node indicates that its idle.

Byte 0 index.
If data is supplied it is user defined.

Type = 36 (0x24) Stream Data.

A steam of information from a node can be reported with mathiss event. Mathiss can be a serial RS-232 channel or som other sequential stream.

- **Byte 0** Sequence number which is increased by one for every new event with stream data.
- **Byte 1 – byte 7** Stream data.

Type = 37 (0x25) Reserved.

Type = 38 (0x26) Reserved.

Type = 39 (0x27) Reserved.

Type = 40 (0x28) Level Changed.

Response/confirmation from ex. a dimmer control after a dimmer command or some other unit that change a level.

Byte 0 Level
Byte 1 Zone
Byte 2 Zone Page

Class = 30 (0x1E) Control

CLASS1.CONTROL

Description

Control functionality.

One of the main concepts of VSCP is that it is an event driven protocol. Commands are sent out as events to the network not as messages to specific devices. A device can belong to a zone which select limit messages of interest for the particular node..

If there is a need to control a specific device the registry model should be used. This is the only way to directly control a device.

Type = 0 (0x00) Undefined

General control.

Type = 1 (0x01) Mute on/off

Mute/Un-mute all sound generating nodes in a zone

Byte 0 If equal to zero no mute else mute.
Byte1 Optional zone to mute (0 – 255)
Byte2 Optional zone page (0 – 255) If not preset interpret as zone-page=0.

zone = oxff or not present means all zones in that page and if no pages i s given all nodes in zones on all pages.

Type = 2 (0x02) All lamps on/off

Turn on/off lamps on nodes in zone.

Byte0 - If equal to zero off else on.
Byte1 - Optional zone.(0 – 255)
Byte2 - Optional zone page (0 – 255) If not preset interpret as zone-page=0.

zone = oxff or not present means all zones in that page and if no pages i s given all nodes in zones on all pages.

Type = 3 (0x03) Open

Perform open on all nodes in zone.

Byte 0 Optional byte that have a meaning given by the issuer of the event.
Byte 1 Optional zone. (0 – 255)
Byte 2 Optional zone page (0 – 255) If not preset interpret as zone-page=0.

zone = oxff or not present means all zones in that page and if no pages i s given all nodes in zones on all pages.

Type = 4 (0x04) Close

Perform close on all nodes in zone.

Byte 0 Optional byte that have a meaning given by the issuer of the event.
Byte 1 Optional zone. (0 – 255)
Byte 2 Optional zone page (0 – 255) If not preset interpret as zone-page=0.

zone = oxff or not present means all zones in that page and if no pages i s given all nodes in zones on all pages.

Type = 5 (0x05) TurnOn

Turn On all nodes in a zone.

Byte 0 Optional byte that have a meaning given by the issuer of the event.
Byte 1 Optional zone. (0 – 255)
Byte 2 Optional zone page (0 – 255) If not preset interpret as zone-page=0.

zone = oxff or not present means all zones in that page and if no pages i s given all nodes in zones on all pages.

Type = 6 (0x06) TurnOff

Turn Off all nodes in a zone.

Byte 0 Optional byte that have a meaning given by the issuer of the event.
Byte 1 Optional zone. (0 – 255)
Byte 2 Optional zone page (0 – 255) If not preset interpret as zone-page=0.

zone = oxff or not present means all zones in that page and if no pages i s given all nodes in zones on all pages.

Type = 7 (0x07) Start

Start all nodes in a zone.

Byte 0 Optional byte that have a meaning given by the issuer of the event.
Byte 1 Optional zone. (0 – 255)
Byte 2 Optional zone page (0 – 255). If not preset interpret as zone-page=0.

zone = oxff or not present means all zones in that page and if no pages i s given all nodes in zones on all pages.

Type = 8 (0x08) Stop

Stop all nodes in zone.

Byte 0 Optional byte that have a meaning given by the issuer of the event.
Byte 1 Optional zone. (0 – 255)
Byte 2 Optional zone page (0 – 255) If not preset interpret as zone-page=0.

zone = oxff or not present means all zones in that page and if no pages i s given all nodes in zones on all pages.

Type = 9 (0x09) Reset

Perform Reset on all nodes in zone.

Byte 0 Optional byte that have a meaning given by the issuer of the event.
Byte 1 Optional zone. (0 – 255)
Byte 2 Optional zone page (0 – 255) If not preset interpret as zone-page=0.

zone = oxff or not present means all zones in that page and if no pages i s given all nodes in zones on all pages.

Type = 10 (0x0a) Interrupt

Perform Interrupt on all nodes in zone.

Byte 0 Interrupt level. (0 – 255 , zero is lowest interrupt level.)
Byte 1 Optional zone. (0 – 255)
Byte 2 Optional zone page (0 – 255) If not preset interpret as zone-page=0.

zone = oxff or not present means all zones in that page and if no pages i s given all nodes in zones on all pages.

Type = 11 (0x0b) Sleep

Perform Sleep on all nodes in zone.

Byte 0 Optional byte that have a meaning given by the issuer of the event.
Byte 1 Optional zone. (0 – 255)
Byte 2 Optional zone page (0 – 255) If not preset interpret as zone-page=0.

zone = oxff or not present means all zones in that page and if no pages i s given all nodes in zones on all pages.

Type = 12 (0x0c) Wakeup

Wakeup all nodes in zone.

Byte 0 Optional byte that have a meaning given by the issuer of the event.
Byte 1 Optional zone. (0 – 255)
Byte 2 Optional zone page (0 – 255) If not preset interpret as zone-page=0.

zone = oxff or not present means all zones in that page and if no pages i s given all nodes in zones on all pages.

Type = 13 (0x0d) Resume

Resume all nodes in zone.

Byte 0 Optional byte that have a meaning given by the issuer of the event.
Byte 1 Optional zone. (0 – 255)
Byte 2 Optional zone page (0 – 255) If not preset interpret as zone-page=0.

zone = oxff or not present means all zones in that page and if no pages i s given all nodes in zones on all pages.

Type = 14 (0x0e) Pause

Pause all nodes in zone.

Byte 0 Optional byte that have a meaning given by the issuer of the event.
Byte 1 Optional zone. (0 – 255)
Byte 2 Optional zone page (0 – 255) If not preset interpret as zone-page=0.

zone = oxff or not present means all zones in that page and if no pages i s given all nodes in zones on all pages.

Type = 15 (0x0f) Activate

Activate all nodes in zone.

Byte 0 Optional byte that have a meaning given by the issuer of the event.
Byte 1 Optional zone. (0 – 255)
Byte 2 Optional zone page (0 – 255) If not preset interpret as zone-page=0.

zone = oxff or not present means all zones in that page and if no pages i s given all nodes in zones on all pages.

Type = 16 (0x10) Deactivate

Deactivate all nodes in zone.

Byte 0 Optional byte that have a meaning given by the issuer of the event.
Byte 1 Optional zone. (0 – 255)
Byte 2 Optional zone page (0 – 255) If not preset interpret as zone-page=0.

zone = oxff or not present means all zones in that page and if no pages i s given all nodes in zones on all pages.

Type = 20 (0x14) Dim lamp(s)

Dim all dimmer devices on a segment to a specified dim value.

Byte 0 Value (0 – 100) . 0 = off, 100 = full on.
Byte 1 Zone to mute (0 – 255). 255 is all zones.
Byte 2 Optional zone page (0 – 255) If not preset interpret as zone-page=0.

Value=254 dim down one step.
Value=255 dim up one step.

Type = 21 (0x15) Change Channel

This is typical for changing TV channels or for changing AV amp input source etc.

Byte 0

- A value between 0 and 127 indicates the channel number.
- A value between 128 to 157 is change **down** by the specified number of channels.
- A value between 160 to 191 is change **up** by the specified number of channels.
- A value of 255 means that this is an extended change channel event and that the channel number is sent in byte 3 and after if needed.

Byte 1 Zone to mute (0 – 255). 255 is all zones.
Byte 2 Optional zone page (0 – 255) If not preset interpret as zone-page=0.

Type = 22 (0x16) Change Level

Change an absolute level.

Byte 0 Absolute level.
Byte 1 Zone for which event applies to (0-255). 255 is all zones
Byte 2 Sub Zone for which event applies to (0-255). 255 is all sub zones

Type = 23 (0x17) Relative Change Level

Byte 0 Relative level.
Byte 1 Zone for which event applies to (0-255). 255 is all zones
Byte 2 Sub Zone for which event applies to (0-255). 255 is all sub zones

Type = 24 (0x18) Measurement Request

Byte 0 Zero indicates all measurements supported by node should be sent (as separate events) Non-Zero indicates a Node specific index specifying which measurement to send
Byte 1 Zone for which event applies to (0-255). 255 is all zones
Byte 2 Sub Zone for which event applies to (0-255). 255 is all sub zones

Type = 25 (0x19) Stream

Byte 0 Sequence number which is increase by one for each stream data event sent.
Byte 1 – Byte 7 Stream data.

Class = 40 (0x28) Multimedia

CLASS1.MULTIMEDIA

Description

Dedicated class for multimedia functionality. This class was introduced to supplement the control class and to offer multimedia specific control events.

Type=1 (0x1) Playback

This is for controlling playback functionality

Byte 0

- 0 = Stop
- 1 = Pause
- 2 = Play
- 3 = Forward
- 4 = Rewind
- 5 = Fast Forward
- 6 = Fast Rewind
- 7 = Next Track
- 8 = Previous Track

Byte 1 Zone for which event applies to (0-255). 255 is all zones
Byte 2 Sub Zone for which event applies to (0-255). 255 is all sub zones

Type=2 (0x2) NavigatorKey

This is typically for navigation functions or DVD controls

Byte 0

- 0..9 = 0..9 keys
- 10 = 10+ key
- 20 = OK
- 21 = Left
- 22 = Right
- 23 = Up
- 24 = Down
- 25 = Menu
- 26 = Selecting
- 65—90 = A..Z Keys
- 97..122 = a-z keys (can't use ASCII hex as numbers are too large so this is the next best thing)

Byte 1 Zone for which event applies to (0-255). 255 is all zones
Byte 2 Sub Zone for which event applies to (0-255). 255 is all sub zones

Type=3 (0x3) Adjust Contrast

This is typically for adjusting the contrast level of a display device

Byte 0

- A value between 0 and 127 indicates the specific contrast level to set
- A value between 128 and 159 is change down by the specified number of contrast levels
- A value between 160 and 191 is change up by the specified number of contrast levels
- A value of 255 means that this is and extended event and that the specific contrast level is sent in byte 3 and after

Byte 1 Zone for which event applies to (0-255). 255 is all zones
Byte 2 Sub Zone for which event applies to (0-255). 255 is all sub zones

Type=4 (0x4) Adjust Focus

This is typically for adjusting the focus settings of a display device

Byte 0

- A value between 0 and 127 indicates the specific focus level to set
- A value between 128 and 159 is change down by the specified number of focus levels
- A value between 160 and 191 is change up by the specified number of focus levels
- A value of 255 means that this is and extended event and that the specific focus level is sent in byte 3 and after

Byte 1 Zone for which event applies to (0-255). 255 is all zones
Byte 2 Sub Zone for which event applies to (0-255). 255 is all sub zones

Type=5 (0x5) Adjust Tint

This is typically for adjusting the tint settings of a display device

Byte 0

- A value between 0 and 127 indicates the specific tint level to set
- A value between 128 and 159 is change down by the specified number of tint levels
- A value between 160 and 191 is change up by the specified number of tint levels
- A value of 255 means that this is and extended event and that the specific tint level is sent in byte 3 and after

Byte 1 Zone for which event applies to (0-255). 255 is all zones
Byte 2 Sub Zone for which event applies to (0-255). 255 is all sub zones

Type=6 (0x6) Adjust Colour Balance

This is typically for adjusting the colour balance settings of a display device

Byte 0

- Yet to be decided

Byte 1 Zone for which event applies to (0-255). 255 is all zones
Byte 2 Sub Zone for which event applies to (0-255). 255 is all sub zones

Type=7 (0x7) Adjust Brightness

This is typically for adjusting the tint settings of a display device

Byte 0

- A value between 0 and 127 indicates the specific brightness level to set
- A value between 128 and 159 is change down by the specified number of brightness levels
- A value between 160 and 191 is change up by the specified number of brightness levels
- A value of 255 means that this is and extended event and that the specific brightness level is sent in byte 3 and after

Byte 1 Zone for which event applies to (0-255). 255 is all zones
Byte 2 Sub Zone for which event applies to (0-255). 255 is all sub zones

Type=8 (0x8) Adjust Hue

This is typically for adjusting the hue settings of a display device

Byte 0
Yet to be decided

Byte 1 Zone for which event applies to (0-255). 255 is all zones
Byte 2 Sub Zone for which event applies to (0-255). 255 is all sub zones

Type=9 (0x9) Adjust Bass

This is typically for adjusting the bass level settings of a sound device. Depending on the implementation, this could **automatically** adjust the treble level. To adjust left and right bass levels, a node would have to use separate zones or sub zones for left and right.

Byte 0

- A value between 0 and 127 indicates the specific bass level to set
- A value between 128 and 159 is change down by the specified number of bass levels
- A value between 160 and 191 is change up by the specified number of bass levels
- A value of 255 means that this is and extended event and that the specific bass level is sent in byte 3 and after

Byte 1 Zone for which event applies to (0-255). 255 is all zones
Byte 2 Sub Zone for which event applies to (0-255). 255 is all sub zones

Type=10 (0xA) Adjust Treble

This is typically for adjusting the treble level settings of a sound device. Depending on the implementation, this could **automatically** adjust the bass level. To adjust left and right treble levels, a node would have to use separate zones or sub zones for left and right.

Byte 0

- A value between 0 and 127 indicates the specific treble level to set
- A value between 128 and 159 is change down by the specified number of treble levels
- A value between 160 and 191 is change up by the specified number of treble levels
- A value of 255 means that this is and extended event and that the specific treble level is sent in byte 3 and after

Byte 1 Zone for which event applies to (0-255). 255 is all zones
Byte 2 Sub Zone for which event applies to (0-255). 255 is all sub zones

Type=11 (0xB) Adjust Master Volume

This is typically for adjusting the master volume level. This could be used for adjusting the level for **all** speakers.

Byte 0

- A value between 0 and 127 indicates the specific volume level to set
- A value between 128 and 159 is change down by the specified number of volume levels
- A value between 160 and 191 is change up by the specified number of volume levels
- A value of 255 means that this is and extended event and that the specific volume level is sent in byte 3 and after

Byte 1 Zone for which event applies to (0-255). 255 is all zones
Byte 2 Sub Zone for which event applies to (0-255). 255 is all sub zones

Type=12 (0xC) Adjust Front Volume

This is typically for adjusting the front speaker volume level. This usually means the two front speakers as opposed to the single centre speaker.

Byte 0

- A value between 0 and 127 indicates the specific volume level to set
- A value between 128 and 159 is change down by the specified number of volume levels
- A value between 160 and 191 is change up by the specified number of volume levels
- A value of 255 means that this is and extended event and that the specific volume level is sent in byte 3 and after

Byte 1 Zone for which event applies to (0-255). 255 is all zones
Byte 2 Sub Zone for which event applies to (0-255). 255 is all sub zones

Type=13 (0xD) Adjust Centre Volume

This is typically for adjusting the front speaker volume level. This usually means the single centre speaker as opposed to the two front speakers.

Byte 0

- A value between 0 and 127 indicates the specific volume level to set
- A value between 128 and 159 is change down by the specified number of volume levels
- A value between 160 and 191 is change up by the specified number of volume levels
- A value of 255 means that this is and extended event and that the specific volume level is sent in byte 3 and after

Byte 1 Zone for which event applies to (0-255). 255 is all zones
Byte 2 Sub Zone for which event applies to (0-255). 255 is all sub zones

Type=14 (0xE) Adjust Rear Volume

This is typically for adjusting the rear speaker volume level.

Byte 0

- A value between 0 and 127 indicates the specific volume level to set
- A value between 128 and 159 is change down by the specified number of volume levels
- A value between 160 and 191 is change up by the specified number of volume levels
- A value of 255 means that this is and extended event and that the specific volume level is sent in byte 3 and after

Byte 1 Zone for which event applies to (0-255). 255 is all zones
Byte 2 Sub Zone for which event applies to (0-255). 255 is all sub zones

Type=15 (0xF) Adjust Side Volume

This is typically for adjusting the side speaker volume level.

Byte 0

- A value between 0 and 127 indicates the specific volume level to set
- A value between 128 and 159 is change down by the specified number of volume levels
- A value between 160 and 191 is change up by the specified number of volume levels
- A value of 255 means that this is and extended event and that the specific volume level is sent in byte 3 and after

Byte 1 Zone for which event applies to (0-255). 255 is all zones
Byte 2 Sub Zone for which event applies to (0-255). 255 is all sub zones

Type=16 to 19 Reserved

These are reserved for other future speaker combinations

Type=20 (0x14) Select Disk

This is typically for selecting a disk for playback

Byte 0

- A value between 0 and 127 indicates the specific disk number
- A value between 128 and 159 is change down by the specified number of disks
- A value between 160 and 191 is change up by the specified number of disks
- A value of 200 means select a random disk
- A value of 255 means that this is and extended event and that the disk number is sent in byte 3 and after

Byte 1 Zone for which event applies to (0-255). 255 is all zones
Byte 2 Sub Zone for which event applies to (0-255). 255 is all sub zones

Type=21 (0x15) Select Track

This is typically for selecting a track for playback

Byte 0

- A value between 0 and 127 indicates the track number
- A value between 128 and 159 is change down by the specified number of tracks
- A value between 160 and 191 is change up by the specified number of tracks
- A value of 200 means select a random track
- A value of 255 means that this is and extended event and that the track number is sent in byte 3 and after

Byte 1 Zone for which event applies to (0-255). 255 is all zones
Byte 2 Sub Zone for which event applies to (0-255). 255 is all sub zones

Type=22 (0x16) Select Album

This is typically for selecting an album for playback

Byte 0

- A value between 0 and 127 indicates the album number
- A value between 128 and 159 is change down by the specified number of albums
- A value between 160 and 191 is change up by the specified number of albums
- A value of 200 means select a random album
- A value of 255 means that this is and extended event and that the album number is sent in byte 3 and after

Byte 1 Zone for which event applies to (0-255). 255 is all zones
Byte 2 Sub Zone for which event applies to (0-255). 255 is all sub zones

Type=23 (0x17) Select Channel

This is typically for selecting a TV Channel

Byte 0
- A value between 0 and 127 indicates the channel number
- A value between 128 and 159 is change down by the specified number of channels
- A value between 160 and 191 is change up by the specified number of channels
- A value of 200 means select a random channel
- A value of 255 means that this is and extended event and that the channel number is sent in byte 3 and after

Byte 1 Zone for which event applies to (0-255). 255 is all zones
Byte 2 Sub Zone for which event applies to (0-255). 255 is all sub zones

Type=24 (0x18) Select Page

This is typically for selecting a page of a film

Byte 0
- A value between 0 and 127 indicates the page number
- A value between 128 and 159 is change down by the specified number of pages
- A value between 160 and 191 is change up by the specified number of pages
- A value of 200 means select a random page
- A value of 255 means that this is and extended event and that the page number is sent in byte 3 and after

Byte 1 Zone for which event applies to (0-255). 255 is all zones
Byte 2 Sub Zone for which event applies to (0-255). 255 is all sub zones

Type=25 (0x19) Select Chapter

This is typically for selecting a chapter of a film

Byte 0
- A value between 0 and 127 indicates the chapter number
- A value between 128 and 159 is change down by the specified number of chapters
- A value between 160 and 191 is change up by the specified number of chapters
- A value of 200 means select a random chapter
- A value of 255 means that this is and extended event and that the chapter number is sent in byte 3 and after

Byte 1 Zone for which event applies to (0-255). 255 is all zones
Byte 2 Sub Zone for which event applies to (0-255). 255 is all sub zones

Type=26 (0x1A) Select Screen Format

This is for controlling screen format of a display device

Byte 0
- 0 = Auto
- 1 = Just
- 2 = Normal
- 3 = Zoom

Byte 1 Zone for which event applies to (0-255). 255 is all zones
Byte 2 Sub Zone for which event applies to (0-255). 255 is all sub zones

Type=27 (0x1B) Select Input Source

This is for controlling the input source of a playback device

Byte 0

- 0 = Auto
- 1 = CD
- 2 = AUX
- 3 = DVD
- 4 = SAT
- 5 = VCR
- 6 = Tape
- 7 = Phone
- 8 = Tuner
- 9 = FM
- 10 = AM
- 11 = Radio (9 – 10 are more specific
- 16 = Component
- 17 = VGA
- 18 = SVideo
- 19 = Video1
- 20 = Video2
- 21 = Video3

Byte 1 Zone for which event applies to (0-255). 255 is all zones
Byte 2 Sub Zone for which event applies to (0-255). 255 is all sub zones

Type=28 (0x1C) Select Output

This is for controlling the output of a playback device

Byte 0

- 0 = Auto
- 16 = Component
- 17 = VGA
- 18 = SVideo
- 19 = Video1
- 20 = Video2
- 21 = Video3

Byte 1 Zone for which event applies to (0-255). 255 is all zones
Byte 2 Sub Zone for which event applies to (0-255). 255 is all sub zones

Type=40 (0x28) Tivo Function

This is typically for accessing TIVO functions

Byte 0

- 1 = Box Office
- 2 = Services
- 3 = Program Guide
- 4 = Text
- 5 = Info
- 6 = Help
- 7 = Backup
- 20 = Red key
- 21 = Yellow key
- 22 = Green key
- 23 = Blue key

24 = White key
25 Black key

Byte 1 Zone for which event applies to (0-255). 255 is all zones
Byte 2 Sub Zone for which event applies to (0-255). 255 is all sub zones

Class = 100 (0x64) Phone

CLASS1.PHONE

Description

This class is for phone related functionality.

Type = 0 (0x00) Undefined.

General phone.

Type = 1 (0x00) Incoming call.

There is an incoming phone call. Usually a caller ID node just sends out numerical information. A database message can follow (later) that contains the real text information.

Byte 0	Id for the call. This is an incremental identity number for each call.
Byte 1	Index of phone message (base = 0). Each call can be broken up into fragments. This is the fragment number.
Byte 2	Total number of messages (fragments) for this call information.
Byte 3 –7	Caller information. Number or real text information.

Type = 2 (0x02) Outgoing call.

There is an outgoing phone call.

Byte 0	Id for the call. This is an incremental id number for each call.
Byte 1	Index of phone message (base = 0). Each call can be broken up into fragments. This is the fragment number.
Byte 2	Total number of messages (fragments) for this call information.
Byte 3– 7	Caller information. Number or real text information.

Type = 3 (0x03) Ring.

This is a message indicating that there is a “ring” for this call.

Byte 0 An id for the call. This can for instance be a number that increases by one for each call.

Type = 4 (0x04) Answer.

The call has been answered.

Byte 0 An id for the call. This can for instance be a number that increases by one for each call.
Byte 1 ID for answer location.

Type = 5 (0x05) Hangup.

The call has been terminated by the receiving end.

Byte 0 An id for the call. This can for instance be a number that increases by one for each call.

Type = 6 (0x06) Giveup.

The call has been terminated by the originating end.

Byte 0 An id for the call. This can for instance be a number that increases by one for each call.

Type = 7 (0x07) Transfer.

The call has been transferred.

Byte 0 An id for the call. This can for instance be a number that increases by one for each call.

Type = 8 (0x08) Database Info.

Byte 0 Id for the call. This is a number that is increased by one for each call. In this case the number is the same as for the incoming or outgoing messages.

Byte 1 Index of phone message (base=0). Each call can be broken up into fragments. This is the fragment number.

Byte 2 Total number of messages (fragments) for this call information.

Byte 3 – 7 Caller information. Real text information.

Class = 101 (0x65)   LIN I/f

CLASS1.LIN

Description

LIN interface. http://www.lin-subbus.org/

Class = 102 (0x66) Display

CLASS1.DISPLAY

Description

This is generic display related functionality. Show info on a screen, LED-display diode, etc

Type = 0 (0x00) Undefined.

General phone.

Class = 110 (0x6E) IR Remote I/f

CLASS1.IR

Description

This is the IR code sent/received from common remote controls.

Type = 0 (0x00) Undefined

Type = 1 (0x01) RC5 Send/Receive.

A RC5 remote code. http://www.xs4all.nl/~sbp/knowledge/ir/sirc.htm

Byte0	RC5 code.
Byte1	RC5 Address.
Byte2	Repeat count if any.

Type = 3 (0x02) SONY 12-bit Send/Receive.

A SONY remote code. http://www.xs4all.nl/~sbp/knowledge/ir/sirc.htm

Byte0	SONY code.
Byte1	SONY address.
Byte2	Repeat count if any.

Type = 32 (0x20) LIRC (Linux Infrared Remote Control).

Packed LIRC codes code. LRC Codes are normally sent as 64-bit codes or even larger codes. Only codes with a length less then 56 bits (7-bytes) are supported by VSCP and the most significant byte of the LIRC code is not transfered.
http://www.lirc.org/

Byte0	LIRC Code, MSB.
Byte1	LIRC Code.
Byte2	LIRC Code.
Byte3	LIRC Code.
Byte4	LIRC Code.
Byte5	LIRC Code
Byte6	LIRC Code, LSB.
Byte7	Repeat count if any.

Type = 48 (0x30) VSCP Abstract Remote Format.

Instead of sending codes that relates to a certain remote this format is general. And therefore more flexible

Byte0	Code, MSB.
Byte1	Code LSB.
Byte2	SubZone
Byte3	MainZone
Byte4	Repeat count if any.

Class = 200 (0xC8) 1-Wire protocol i/f

CLASS1.ONEWIR
E

Description

Carrier for the 1-wire API.

Type = 0 (0x00) Undefined.

General one wire.

Type = 1 (0x01) New ID

A new 1-wire device is discovered. The 1-wire device id is in the data field layed out as a standard 64-bit 1-wire ID.

Type = 2 (0x02) Convert.

Command a 1-wire node to do a conversion.

Type = 3 (0x03) Read ROM.

Execute a 1-wire Read ROM command. The 1-wire device id is in the data fieldlayed out as a standard 64-bit 1-wire ID.

Type = 4 (0x04) Match ROM.

Execute a 1-wire Match ROM command. The 1-wire device id is in the data fieldlayed out as a standard 64-bit 1-wire ID.

Type = 5 (0x05) Skip ROM.

Execute a 1-wire Skip ROM command.

Type = 6 (0x06) Search ROM.

Execute a 1-wire Search ROM command. The 1-wire device id is in the data fieldlayed out as a standard 64-bit 1-wire ID.

Type = 7 (0x07) Conditional Search ROM.

Execute a 1-wire Conditional Search ROM command. The 1-wire device id is in the data field layed out as a standard 64-bit 1-wire ID.

Type = 8 (0x08) Program.

Execute a 1-wire Program command. The 1-wire device id is in the data fieldlayed out as a standard 64-bit 1-wire ID.

Type = 9 (0x09) Overdrive skip ROM.

Execute a 1-wire Overdrive Skip ROM command.

Type = 10 (0x0A) Overdrive Match ROM.

Execute a 1-wire Overdrive Match ROM command. The 1-wire device id is in the data field.layed out as a standard 64-bit 1-wire ID.

Type = 11 (0x0B) Read Memory.

Execute a 1-wire Read Memory command. The 1-wire device id is in the data field.

Type = 12 (0x0C) Write Memory.

Execute a 1-wire Write Memory command. The 1-wire device id is in the data field.layed out as a standard 64-bit 1-wire ID.

Class = 201 (0xC9) X10 protocol i/f

CLASS1.X10

Description

X10 Protocol functionality.

Type = 0 (0x00) Undefined.

General Message.

Type = 1 (0x01) X10 Standard Message Receive.

Byte 0 Header/Code
Byte 1 Address or Function byte

The header has the following format

```
Bit:      7  6  5  4  3  2  1  0
Header:  <  Dim amount  >  1  F/A  0
```

Where

F = Function
A = Address

Note that bit 0 always is zero.

```
Bit:      7  6  5  4  3  2  1  0
Address:  <House Code >  <Device Code>
```

```
Bit:      7  6  5  4  3  2  1  0
Address:  <House Code >  <  Function >
```

Type = 2 (0x02)   X10 Extended message Receive.

Byte 0	Header Code (Always 0x03)
Byte 1	Function.
Byte 2	Unit Code.
Byte 3	Data.
Byte 4	Command.

Where

Function byte is

```
Bit:        7  6  5  4  3  2  1  0
Address:  <House Code >   0  1  1  1
```

The unit code contains the encoded unit in the lower four bits.

Type = 3 (0x03)   X10 Standard Message Send.

Byte 0	Node address
Byte 1	Header/Code
Byte 2	Address or Function byte

The format is the same as for Type=1 except for the Node address in the first byte.

Type = 4 (0x04)   X10 Standard Message Send.

Byte 0	Node address
Byte 1	Header Code (Always 0x03)
Byte 2	Function.
Byte 3	Unit Code.
Byte 4	Data.
Byte 5	Command.

The format is the same as for Type=2 except for the Node address in the first byte.

Type = 5 (0x05) Simple x10 message .

Byte 0	House code (Required)
Byte 1	Unit code (ignored if not relevant such as All lights off)
Byte 2	Command (bright, dim, on, off etc)
Byte 3	Dim level (if needed)
Byte 4	Repeats (1 or 0 to send once)

Instead of sending house code P, Unit code 1 and then sending house code P, Command On, once can simply send house code P, unit code 1, command On, which will then be translated into 2 x10 messages.

Class = 202 (0xCA) LON Works protocol i/f

CLASS1.LON

Description

LON Works functionality.

Type = 0 (0x00) Undefined.

General Message.

Class = 203 (0xCB) EIB protocol i/f

CLASS1.EIB

Description

EIB functionality.

Type = 0 (0x00) Undefined.

Class = 204 (0xCC) S.N.A.P. protocol i/f

CLASS1.SNAP

Description

You can read more about the **S.N.A.P.** protocol at http://www.hth.com/snap/

In this document http://www.hth.com/filelibrary/pdffiles/snap.pdf

Type = 0 (0x00) Undefined.

General Message.

Class = 205 (0xCD) CBUS

CLASS1.CBUS

Description

Control and interface for the CBUS protocol

Type = 0 (0x00) Undefined.

General Message.

Class = 206 (0xCE) GPS

CLASS1.GPS

Description

Control and interface for the GPS protocols (NMEA, SIRF etc)

Type = 0 (0x00) Undefined.

General Message.

Class = 509 (0x1FD) Log

CLASS1.LOG

Description

Logging functionality.

Type = 0 (0x00) Undefined.

General Log Message.

Type = 1 (0x01) Log Message.

Byte 0	id for message.
Byte 1	Log level for message.
Byte 2	Idx for this message.
Byte 3-7	Message.

Type = 2(0x01) Log Start.

Byte 0 id for log

Start logging.

Type = 3 (0x03) Log Stop.

Byte 0 id for log

Stop logging.

Type = 4 (0x04) Log Level.

Byte 0	id for log
Byte 1	Log level

Set level for logging.

Class = 510 (0x1FE) Laboratory use

CLASS1.LABORATORY

Description

This class is intended for lab usage. No production device should use this event type.

Type = 0 (0x00) Undefined.

General Message.

Class = 511 (0x1FF) Local

CLASS1.LOCAL

Description

This even type is for local defined events. It is thus possible to add user defined events here. In a public environment the risk for collisions with other devices that also use CLASS.LOCAL should be noted. It is good to make user events configurable in the device to give users a chance to avoid problems.

Class Definitions Level II

Class = 512-1023 Level 1 message

CLASS2.
LEVEL1

Description

Class 512-1023 are reserved for packets that should stay in the Level 2 network but that in all other aspects (the lower nine bits + type) are defined in the same manner as for Level I. For EVENT.CLASS.PROTOCOL all nickname id's should be replaced with the full GUID for the node and all bytes after shifted with the size of the GUID.

Class = 1024 (0x400) Level II Protocol Functionality

CLASS2.PROTOCOL

Description

For Level I events class=0 defines protocol control functionality. All messages of this class are repeated at class=512 for use on Level II networks. The only difference is that the GUID is used instead of the Level I nickname.

This class defines protocol functionality for Level II.

To simplify the handling of level II events, the data portion of the VSCP message can be considered as being made up of two parts.

An 8-byte **code portion** (size of long integer) followed by a **data portion** if required.

This is simply done to make processing level II messages a little easier.

The following events have been added to the level II control events to support configuration management.

Type = 1 (0x01) ReadRegisterII

Byte 0-3	Register to read (or start index).
Byte 4-5	Number of registers to read (max 487).
Byte 6-7	Reserved.
Byte 8-31	Contains the GUID of the target node.

Type = 2 (0x02) WriteRegisterII

Byte 0-3	Registers to write (or start index).
Byte 4-7	Reserved.
Byte 8-31	contains the GUID of the target node
Byte 32...	Data to write to register(s)

Type = 3 (0x03) ReadWriteResponseII

This is the response from a read and a write. Note that the data is returned in both cases and can be checked for validity.

Byte 0-3	Start index for register read/written.
Byte 4-7	Reserved.
Byte 8...	Data read/written.

Type = 4 (0x04) ConfigReadRequest

Byte 0 and 1	Index of configuration segment being requested (0xFFFF indicates all pages).
Byte 2–7	reserved.
Byte 8–31	contains the GUID of the target node.

Type = 5 (0x05) ConfigReadResponse

Byte 0 and 1	Index of configuration segment being sent.
Byte 2 and 3	Total number of segments that make up the full configuration.
Bytes 4 – 8	Reserved.

Bytes 9 till the end of data contains and XMLstring for the configuration

Note that the GUID of the node whose configuration this is, is the 'sender', which can be read from the VSCP header.

Type = 6 (0x06) ConfigChanged

This is a response from a node for which a copy of it's configuration has been modified and is available for download.

Bytes 0 to 7	Reserved.
Byte 8–31	Contains the GUID of the target node.

Type = 7 (0x07) ConfigUpdateRequest

Request a copy of a configuration form a node if GUID given or from any node willing to answer the request if no GUID given.

This message is broadcast by a node who's configuration had been remotely modified or a node that wants its configuration restored.

Bytes 0 to 7	Reserved.
Byte 8–31	Contains the GUID of the target node (can be left out see above).

Type = 8 (0x08) ConfigUpdateResponse

Byte 0 and 1	Index of configuration segment being sent.
Byte 2 and 3	Total number of segments that make up the full configuration.
Bytes 4 – 8	Reserved.
Bytes 9 – 31	Contains the GUID of the target node for this configuration.
Bytes 25 ...	Contain and XML string for the configuration

Class = 1026 (0x402) xAP

CLASS2.XAP

Description

Control and interface for the xAP protocol.xAP is a standard for home control use. This package type encapsulates xAP messages. More info about the standard can be found at. http://www.xapautomation.org/

Yet do be defined.

Class = 1027 (0x403) xPL

CLASS2.X
PL

Description

Control and interface for the xPL protocol.xPL is a standard for home control use. This package type encapsulates xPL messages. More info about the standard can be found at http://www.xplproject.org.uk/

Yet do be defined.

Class = 1028 (0x404) Text to speech

CLASS2.TEXT2SPEECH

Description

This is an interface that translates text to speech

Type = 0 (0x00) Undefined.

Type = 1 (0x01) Talk

Byte 0 and 1	Zone to talk in.
Byte 2 and 3	Voice to talk in.
Byte 4	Relative volume (0-100%).
Byte 5 and 7	Reserved.
Byte 8...	Text to seek,

Appendix A – Assigned Global Unique ID's

Series	Reserved to/for

`FF FF FF FF FF FF FF FF`
`YY YY YY YY YY YY YY YY`
	Dallas Semiconductor GUID's. This is a 64-bit id.

`FF FF FF FF FF FF FF FE`
`YY YY YY YY YY YY XX XX`
	Ethernet Device GUID's. so the holder of the address can freely use the two least significant bytes of the GUID.

`FF FF FF FF FF FF FF FD`
`YY YY YY YY XX XX XX XX`
	Internet version 4 GUID's. This is a 32-bit id so the holder of the address can freely use the four least significant bytes of the GUID.

`FF FF FF FF FF FF FF FC`
`XX XX XX XX XX XX XX XX`
	Private. Use for in-house local use.

`FF FF FF FF FF FF FF FB`
`YY YY YY XX XX XX XX XX`
	ISO id. This is a three byte id so the holder of the ISO ID can freely use the five least significant bytes of the GUID.

`FF FF FF FF FF FF FF FA`
`YY YY YY YY XX XX XX XX`
	CiA (CAN in Automation) vendor ID. This is a 32-bit id so the holder of the vendor ID can freely use the four least significant bytes of the GUID.

`FF FF FF FF FF FF FF FB 00 00 00 00`
`to`
`FF FF FF FF FF FF FF FF FF FF FF FF`
	Reserved.

`00 00 00 00 00 00 00 00 00 00 00 00`
`xx xx xx xx`
	Lab usage.

`FE YY YY YY YY YY YY YY`
`YY YY YY YY YY YY YY YY`
	Reserved for a generated 128 bit GUID where the most significant byte is replaced by FE
	
	Only use for Level II and on internal net.
	
	http://hegel.ittc.ku.edu/topics/internet/internet-drafts/draft-l/draft-leach-uuids-guids-01.txt

`01 00 00 00 00 00 00 00`
`00 00 00 00 XX XX XX XX`
	eurosource
	Brattbergavägen 17
	820 50 LOOS
	Sweden
	Phone: +46 657 413430
	Fax: +46 657 413503
	Email: info@eurosource.se
	Web: http://www.eurosource.se

`02 00 00 00 00 00 00 00`
`00 00 00 00 XX XX XX XX`
	Cedric Priest.
	9 Park Rd,
	Palmerston North,
	New Zealand.
	Phone:++64212671952.
	email: clpriest@orcon.net.nz

`03 00 00 00 00 00 00 00`
`00 00 00 00 XX XX XX XX`
	Xedia Systems Limited
	236 St. George Wharf
	Vauxhall
	London SW8 2LR
	Great Britain
	Phone: +44 207 020 7027
	Email: charles.v.tewiah@xedia.co.uk
	Web:

`04 00 00 00 00 00 00 00`
`00 00 00 00 XX XX XX XX`
	Nyholm Solutions
	Sursikvägen 47 A 1, 68910 Bennäs, Finland
	Phone: +358 50 505 1080
	Fax:: +358 4210 505 1080
	Email: andreas.nyholm@nyholmsolutions.fi
	Web: http://www.nyholmsolutions.fi

`05 00 00 00 00 00 00 00`
`00 00 00 00 XX XX XX XX`
	RTist BV
	Chromiumweg 45 - 3812NL Amersfoort - The Netherlands
	Phone: +31-6-5331-0072
	Email: robtu@rtist.nl
	Web: http://www.rtist.nl

`06 00 00 00 00 00 00 00`
`00 00 00 00 XX XX XX XX`
	Huitsing Embedded Systems
	Dr. Mondenweg 5
	7831 JA Nw. Weerdinge
	Phone: +31 (0)591 521222
	Email: info@huitsing.com
	Web: http://www. huitsing.com

`07 00 00 00 00 00 00 00`
`00 00 00 00 XX XX XX XX`
	Dr. David Harris
	1545 York Place
	Victoria, British Columbia
	Canada V8R 5X1
	Phone: 250-592-5654
	Fax: 250-592-5654
	Email: dpharris@telus.net
	Web: http://omniport.dpharris.ca/

LEGEND:
YY Stands for a rigid id such as a MAC address or the ISO id.
XX Freely assigned number by user.

Appendix B – References

VSCP Homepage
http://www.vscp.org

Open Source Can Driver for Linux and the 82257/SJA1000
http://can.sourceforge.net

BOSCH CAN website
http://www.can.bosch.com

CiA Can In Automation homepage
http://www.can-cia.de

IXXAT
http://www.ixxat.de/english/index.html

Kvaser
http://www.kvaser.se/

SI units etc.
http://www.unc.edu/~rowlett/nodes/index.html

Appendix C – VSCP over CAN

Just as for CANopen and DeviceNet the sample point should be set at 87.5%

VSCP in the standard CAN frame

Frame id	29 bit id	CAN usage	VSCP	Comment
1	-	**SOF**	**SOF**	
2	28	Standard identifier bit 11	Priority bit 2	
3	27	Standard identifier bit 10	Priority bit 1	
4	26	Standard identifier bit 9	Priority bit 0	
5	25	Standard identifier bit 8	Hardcoded	
6	24	Standard identifier bit 7	Class bit 6	
7	23	Standard identifier bit 6	Class bit 5	
8	22	Standard identifier bit 5	Class bit 4	
9	21	Standard identifier bit 4	Class bit 3	
10	20	Standard identifier bit 3	Class bit 2	
11	19	Standard identifier bit 2	Class bit 1	
12	18	Standard identifier bit 1	Class bit 0	
13	-	SRR	SRR	Substitute remote request.
14	-	IDE	IDE	Reserved for later use
15	17	Extended identifier bit 17	Class bit 1	
16	16	Extended identifier bit 16	Class bit 0	
17	15	Extended identifier bit 15	Type bit 7	
18	14	Extended identifier bit 14	Type bit 6	
19	13	Extended identifier bit 13	Type bit 5	
20	12	Extended identifier bit 12	Type bit 4	
21	11	Extended identifier bit 11	Type bit 3	
22	10	Extended identifier bit 10	Type bit 2	
23	9	Extended identifier bit 9	Type bit 1	
24	8	Extended identifier bit 8	Type bit 0	
25	7	Extended identifier bit 7	Node Address bit 7	
26	6	Extended identifier bit 6	Node Address bit 6	
27	5	Extended identifier bit 5	Node Address bit 5	
28	4	Extended identifier bit 4	Node Address bit 4	

29	3	Extended identifier bit 3	Node Address bit 3	
30	2	Extended identifier bit 2	Node Address bit 2	
31	1	Extended identifier bit 1	Node Address bit 1	
32	0	Extended identifier bit 10	Node Address bit 0	
33	-	RTR	RTR	

Format of the 29 bit CAN identifier in VSCP

Bit	Use	Comment
29	Priority	Highest priority is 000b (=0) and lowest is 111b (=7)
28	Priority	
27	Priority	
26	Hardcoded	If this bit is set the Nickname ID of the device is hardcoded
25	Class	The class identifies the message class. There are 512 possible classes. *This is the MSB bit of the class.*
24	Class	
23	Class	
22	Class	
21	Class	
20	Class	
19	Class	
18	Class	
17	Class	
16	Type	The type identifies the type of the message within the set message class. There are 256 possible types within a class. *This is the MSB bit of the type.*
15	Type	
14	Type	
13	Type	
12	Type	
11	Type	
10	Type	
9	Type	
8	Originating-Address	The address is a unique address in the system. It can be a hard-set address or (hardcoded bit set) or an address retrieved through the **nickname discovery process**. **0x00** is reserved for a segment master node. **0xff** is reserved for no assigned nickname.
7	Originating-Address	
6	Originating-Address	
5	Originating-Address	
4	Originating-Address	
3	Originating-Address	
2	Originating-Address	
1	Originating-Address	

If addressing of a particular node is needed the nickname address for the node is given as the first byte in the data part. This is a rare situation for VSCP and is mostly used in the register read/write events.

Appendix D -

VSCP over RF/IR/PLC/RS-232/RS-422 and RS-485

VSCP can be used over RS-232/RS-422 point-to-point links even if it's designed for a multi master/multi drop system. In this case we typically have only one master and one slave.

The 29-bit ID is laid out in a 32-bit word with the three most significant bits set aside for the Data Length Code (DLC). As the DLC must be coded in three bits and the CAN DLC normally is in the range 0-8 we skip the zero data packet and only allow packets with 1-8 data bytes represented by a DLC that is 0-7.

Packets have the following content:

Byte	Description
SYNC	One or several byte(s) of 0x54
ID	ID, Most significant byte.
ID	ID
ID	ID
ID	ID, Least significant byte.
Data	0-8 bytes of data.
CRC	MSB of 16-bit CRC (not calculated on sync char(s).
CRC	LSB of 16-bit CRC (not calculated on sync char(s).

Multi drop systems can be built and a node can send data if it detects silence on the bus for 100 ms or more. It tries to send data on the link/bus listening to it's own transmission and immediately stopping it's transmission if the data gets garbled. It then waits the number of milliseconds it gets by interpreting its least significant GUID byte as a timeout time expressed in milliseconds and makes a new try after this timeout has expired. If a collision is again detected it uses the least significant byte +1 of the GUID as a timeout value and continues in this fashion until all GUID bytes has been worked through in which case its starts with the least significant byte all over again.

Appendix D -

VSCP over TCP/IP, Ethernet

VSCP can be used over Ethernet as its own protocol. No Ethernet number is assigned at it will probably never be. In this case we have one or several masters and one or more slave nodes. A maximum of 128 nodes are allowed.

For UDP and TCP the packets are organized in the same way.

The 29-bit ID is laid out in a 32-bit word with the three most significant bits set aside for the Data Length Code (DLC). As the DLC must be coded in three bits and the CAN DLC normally is in the range 0-8 we skip the zero data packet and only allow packets with 1-8 data bytes represented by a DLC that is 0-7.

Packets have the following content:

Byte	Description
ID	ID, Most significant byte.
ID	ID
ID	ID
ID	ID, Least significant byte.
Data	0-8 bytes of data.

Appendix E - CRC checksums

8-bit checksum

The 8-bit checksum used is the DOW checksum.

Formula:	$X^8 + X^5 + x^4 + 1$
Polynomial:	0x18
Initial Reminder:	0x00

16-bit checksum

The 16-bit checksum used is the CCITT checksum.

Formula:	$X^{16} + X^{12} + x^5 + 1$
Polynomial:	0x1021
Initial Reminder:	0xFFFF

This checksum is used for VSCP Level II datagrams.

32-bit checksum

The 32-bit checksum used is the Ethernet checksum.

Formula: $X^{32} + X^{26} + X^{23} + X^{22} + X^{16} + X^{12} + X^{11} + X^{10} + X^8 + X^7 + X^5 + X^4 + X^2$
$+ X^1 + 1$

Polynomial:	0x04c11db7
Initial Reminder:	0xffffffff

Note that result is inverted.

http://www.netrino.com/Connecting/2000-01/ is a good pointer for code

Appendix F – Reversion History

2004-01-26 — Level 2 introduced.
2004-03-01 --- Settled for master less solution.
2004-04-05 --- Fixed typo max 100 meters should be 500 meters.
2004-04-06 --- Start for GUID storage should be 0xd0.
2004-04-06 --- Full address field used 1-254 instead of 1-127. 0x00 is reserved for master. 0xff is reserved for a node with no nickname assigned.
2004-04-07 --- **read** & **write** did not have the destination address given. This is now in byte 0.
2004-05-27 --- Fixed typo- in appendix A where the reserved GUID's ranges was wrong.
2004-06-01 --- For Level II messages the data part must not be larger then 487 bytes (512-25), This was previously 1024 bytes but a total package size of 512 bytes should not be to avoid fragmentation problems.
2004-06-01 --- Private GUID has been defines that can be used in the same way as 192.168.0.0 and 10.0.0.0 is used on the Internet.
2004-06-01 --- The initial reminder for the CCITT checksum was wrong (0x0000 instead of 0xffff).
2004-06-01 --- Fahrenheit allowed as an optional unit for temperature.
2004-06-05 --- SYNC and CRC removed from Ethernet packages.
2004-06-11 --- EDA Decision matrix format for uP added. Also control function to get size and offset to the decision matrix.
2004-06-11 --- **Dusk** and **Dawn** events added.
2004-06-11 -- X10 Message detail added. M.U.M.I.N. class replaced with CBUS class.
2004-06-21 – Added the database message for the phone class. Page read/write added.
2004-08-26 – **Clarified** the hard coded bit and the functionality.
2004-08-26 – Decision Matrix definition for uP final.
2004-08-26 – Module description file final.
2004-08-26 – GPS event added.
2004-08-26 – Register assigned for boot loader algorithm.
2004-08-26 – Page Read/Write final.
2004-09-20 – Bootloade rinfo in XML data.
2004-09-21 – index added for data and event class.
2004-10-22 – A all lights on/off and a dimmer control type added.
2004-09-21 – Clarified the UDP package format.
2004-12-05 – English proofreading done. Some new events added for logging.
2004-12-14 – Fixed an error in the data specification format byte definition.
2005-01-15 – CAN standard speed changed to 500 kbps.
2005-02-06 – Many, many changes and fixes. Mostly additions to Level II texts.
2005-02-15 – Bootloader suppor Register No bootloader suppoert = 0xff instead of 0x00.
2005-02-15 – Fixed typo in CLASS.CONTROL TYPE=9
2005-05-11 – First version of multimedia class added.
- Better explanation of paged read/write
- Max buffer size introduced for device.
- Decision matris format for Level I/Level II fina.
- Paged DM introduced to save register space.
- Replaced 32.bit CRC with 16-bit CRC for VSCP bootloader.
- Clarified CLASS1.PROTOCOL
- Event "Measurement Request " added.
- Event Chan Level/Relative Change Level added.

– Stream control and information events added.
– Many more changes.....