VSCP, Back to the Very Simple Control Protocol

Please consider supporting the VSCP project.

VSCP Specification

IMPORTANT
This specification and the methods described in it is placed in the Public Domain from the point is was created in August of year 2000. It is therefore totally free for anyone to use and/or implement in any way they like commercially or use for any other purpose.

This is version: 1.5.0

Back to Specification

Introduction

VSCP is an **open source** standard protocol for Home Automation and other remote control applications. It enables simple, low-cost devices to be networked together with high-end computers, whatever the communication media used.

There are a lot of technologies and protocols that claims to be the perfect solution for home automation and SOHO (Small Office/Home Office) control. X10 is the most well known system using power line. More and more RF solutions are now available too: Bluetooth, Zigbee, Z-wave, Wifi, etc. Many systems use a dedicated bus based on RS485, CAN, LIN, LON, TCP/IP, etc. or can sometimes support different transport layers: CANopen, KNX (EIB), C-Bus, LonWorks, etc.

Most of them are proprietary, some are somehow "open", meaning you can participate if you're part of the alliance and pay your yearly fees, or anything similar. Except the small companies that have their own proprietary and completely closed small and simple protocols, the more standard solutions are usually difficult to understand, implement and require quite a bit of ressources.

VSCP was designed with the following goals in mind:

- **Free and Open**. No usage, patent or other costs for its implementation and usage.
- **Low cost**.
- **K.I.S.S.** (Keep it simple stupid.) Simplicity usually rules.
- **Flexibility** and **performance**, we still want the system to do what we need, in any situation.

The VSCP Protocol was initially used in CAN networks. CAN is very reliable and cheap today and allow us to manufacture low cost nodes that can work reliably, efficiently and can be trusted in their day-to-day use. But VSCP can be used equally well in other environments than CAN.

To meet both low cost and performance, VSCP is divided into 2 levels. Level 1 is intended for low-end nodes (i.e. based on tiny microcontrollers) while Level 2 is intended for higher level and faster transport layers such as TCP/IP. All nodes can talk together but level 2 nodes achieve better performance when talking together, while level 1 nodes that don't require much processing can be implemented with very cheap technologies.

Furthermore, VSCP:

- uses standard components and cables;
- should be easy to configure.

Open? What does that mean

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.

There is an exception in the license from GPL/LGPL that make the tools more useful for commercial use.

Full license is here vscp_license

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

http://www.vscp.org [http://www.vscp.org]
http://sourceforge.net/projects/can [http://sourceforge.net/projects/can]

There are two mailing lists available on Sourceforge https://sourceforge.net/mail/?group_id=53560 [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 [http://lists.sourceforge.net/lists/listinfo/can-canal]
- To subscribe to the VSCP list go to http://lists.sourceforge.net/lists/listinfo/can-vscp [http://lists.sourceforge.net/lists/listinfo/can-vscp]

When was the project started?

Many of the thoughts behind this protocol come from work done as early as 1986 but high node costs at the time made it impossible to do the things VSCP can do today. The official start date/time is 2000-08-28 14:07 CET when the EDA project (http://sourceforge.net/projects/eda [http://sourceforge.net/projects/eda]) was registered at Sourceforge.

EDA stood for Event-Decision-Action and is still preserved in the decision matrix of VSCP.

Why?

Why is this the VSCP protocol needed?

Most people don't remember the world looked like for PC developers before Windows where introduced. This was a time when the developer needed to ship a big bunch of floppy disks with his/heres application. One big pile of floppys where for different printers. Another set of disk was for different graphical cards and yet another for different pointing devices etc. It was not uncommon to have one floppy for the application an fifteen to twenty for drivers.

Windows changed this. The OS introduced abstraction for devices and from that point drivers was something the OS dealt with and the application developer could concentrate on creating the application. This was the big reason behind the boom in software that made Microsoft and others successful.

VSCP does this for automation tasks. Look around and see how it looks today. We have applications that work for Zigbee, X10, KNX, LonWorks etc. Some try to combine them into one application like MrHouse and the like but still it's a different thing to turn on a lamp using Zigbee, X10, KNX or whatever. In the best case the same user interface component can be used but still there is a need to differentiate between the technologies use also at that level and most important the knowledge about the technology is needed on the top level.

VSCP try to hide this. Drivers implement the interface to the technology and they all talk to the system using the VSCP protocol and understand VSCP protocol events. Compare this with printing under a modern OS. It's no difference today if you print to a Laser printer or a ink jet printer. Also it all works the same if the printer use protocol x, y or z. You are also still able to configure and print with the printers. This is where the abstraction comes into place.

To switch on a lamp (or a group of lamps) in VSCP we send a

```
CLASS1.CONTROL, Type=5, TurnOn
```

event.

A driver translate this event to it's own format and does its specific work using it's own protocol and return a

```
CLASS1.INFORMATION, Type=3, ON
```

event when it's job is done as a confirmation for the rest of the system.

An application now does not need to bother how the actual control is done. On the application level it's enough to implement a button and some visual indication to indicate the outcome of the operation.

A system to present some measurement data is another example. Think of a system with temperature sensors. They all use different technologies but a driver for each translate the temperature readings to a common

```
CLASS1.MEASUREMENT, Type=6, Temperature measurement
```

event. This event is region independent and format independent. It is easy to create a driver that log this value into a database. Also here in a common format. You can now build a web applet that shows the temperature for every possible temperature sensor. As the format is common and easy to collect in a database in a common way you can also write a statistical application that show temperature data and work for any temperature sensor.

The above is the most important reason for VSCP but there is more.

Each VSCP device have a common way it can be configured by. This means one high level software can be used for all devices. Note that a device necessarily does not need to implement this. Instead a driver can do that and make it look like a VSCP device. Typically is a 1-wire sensor where the driver can implement the parameters for it and export it in a common way.

All VSCP devices is described in a common way. The device itself holds this information and therefore when a device is found all information about how it is configured and used is available. Again this information can be implemented in the driver.

VSCP defines a lot more functionality and can be used all the way out to the actual device. Still the most important part is the abstraction.

Back to Specification

Physical Level/Lower level protocols

When the protocol was designed it’s usefulness on the CAN bus was the main objective. The identifier length and many other things on Level I are therefore very “CAN-ish” in its design and behavior. CAN is however no prerequisite. The protocol works equally well over RS-232, RF-Links, Ethernet etc.

Level I is effective over bandwidth limited links but don't have the full GUID of the nodes in the frames and some sort of lower level addressing system must be used.

VSCP over CAN
VSCP over Serial Links
VSCP over RS-485
VSCP over TCP/IP
VSCP over RF and PLC

vscp_specification_level_ii_physical_level.txt · Last modified: 2008/04/09 11:16 (external edit)

Back to Specification
Physical Level

VSCP over CAN

* The maximum number of nodes VSCP can handle is 254. In practice, the number of nodes that can be connected to a CAN bus depends on the minimum load resistance a transceiver is able to drive. This is typically around 120 depending on all the transceivers and media (cables) used.

* The transmission speed (or nominal bit rate) is set by default at 125 kilobits per second. All nodes should support this speed but lower bitrates can be used too. See Using alternative bitrates.

* The maximum length of the cabling in a segment is typically 500 meters using AWG24 or similar (CAT5) and a nominal bit rate of 125kbps. Drops with a maximum length of 24M can be taken from this cable and the sum of all drops must not exceed a total of 120 meters counted together.

Bit Rate	Max Bus Length (m)	Max Drop Length (m)	Max Cumulative Drop Length (m)
1M	25*	2	10
800k	50*	3	15
500k	100	6	30
250k	250	12	60
125k	500	24	120
50k	1000	60	300
20k	2500	150	750
10k	5000	300	1500

* The cable should be terminated at both ends with 120 ohms resistors.

* VSCP requires Extended Data Frames with a 29-bit identifier.

* The protocol is compatible with elementary CAN nodes such as the Microchip MCP2502x/5x I/O CAN expander.

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

Format of the 29 bit CAN identifier in VSCP

Bit	Use	Comment
28	Priority	Highest priority is 000b (=0) and lowest is 111b (=7)
27	Priority	
26	Priority	
25	Hardcoded	If this bit is set the Nickname ID of the device is hardcoded
24	Class	The class identifies the message class. There are 512 possible classes. This is the MSB bit of the class.
23	Class	
22	Class	
21	Class	
20	Class	
19	Class	
18	Class	
17	Class	
16	Class	
15	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.
14	Type	
13	Type	
12	Type	
11	Type	
10	Type	
9	Type	
8	Type	

7	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. 0×00 is reserved for a segment master node. 0xff is reserved for no assigned nickname.
6	Originating - Address	
5	Originating - Address	
4	Originating - Address	
3	Originating - Address	
2	Originating - Address	
1	Originating - Address	
0	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.

RJ-XX pinout

Pin RJ45,RJ12,RJ11,RJ10	*Use*	RJ-11	RJ-12	RJ-45
1	+9-16V DC	-	-	RJ-45
2 1	+9-16V DC	-	RJ-12	RJ-45
3 2 1	+9-16V DC	RJ-11	RJ-12	RJ-45
4 3 2	CANH	RJ-11	RJ-12	RJ-45
5 4 3	CANL	RJ-11	RJ-12	RJ-45
6 5 4	GND	RJ-11	RJ-12	RJ-45
7 6	GND	-	RJ-12	RJ-45
8	GND	-	-	RJ-45

It is recommended that "professional" equipment use the voltage range +12V - 28V instead of the more domestic range +9V-16V.

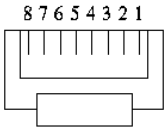


Figure 1:
End view of RJ45 Plug

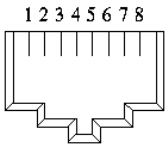


Figure 2:
Looking into an RJ45 Jack

Note that the schematics drawings for VSCP modules use a symbol that is numbered looking from the PCB side. It thus appear as to be numbered in the other direction but is actually the same.

Note also that CANOpen specify a different schema http://www.cd-systems.com/Can/can-cables.htm [http://www.cd-systems.com/Can/can-cables.htm] which is opposite numbered VSCP **but they are actually the same**. We use the numbering that look into the female connector and connt from the left. They use the Arabic style and count from the right.

Aways try to use a *pair* of wires for CANH/CANL fort best noise immunity. If the EIA/TIA 56B standard [http://images.google.se/imgres?imgurl=http://www.datalinkcom.net/images/rj45wiring.gif&imgrefurl=http://ww is used this condition will be satisfied. This is good as most ethernet networks already is wired this way.

Using alternative bitrates

The bitrate should normally be fixed at 500kbps but can be set at any lower rate on a segment where one have control of the bitrate setting. Just configure all you nodes to use the alternative bit rate.

The following method can also be used to avoid manual configuration of every node.

A segment can use a different speed than the default 500kbps. When a new node is initialized, it should first start up in a state where it doesn't send anything on the bus but only listen to all traffic. It should listen in turn at speeds of 1000kbps, 800kbps, 500kbps, 250kbps, 125kbps, 100kbps, 50kbps, 20kbps and 10kbps for at least two seconds each, until valid data is received. That speed is then kept and the nickname discovery process is started.

If no traffic is discovered on any of the speeds, the node is initialized with the default speed. This will be the case for the first node on the segment.

vscp_specification_-_vscp_level_i_over_can.txt · Last modified: 2008/05/17 14:28 by 87.249.175.45

Back to Specification
Physical Level

VSCP over RS-232

Level I

Preliminary

A byte stuffing binary interface is used to talk to the module through the serial interface.

The protocol is created for VSCP Level I events but can be used for other purposes also, as the 16 byte data content indicates. In this case the class and type bytes can be used freely by the implementor.

frame formats

[DLE]
[STX]
Operation 0 = No operation 1 = Level I event to/from node. 2 = Level II event to/from node. 3 = Poll for event. 4 = No events to send. 5 = CAN Frame. 10 = Reserved. 249 = Sent ACK. 250 = Sent NACK. 251 = ACK. 252 = NACK. 253 = error. 254 = Command Reply. 255 = Command.
Flags + bit nine of class bit 0-4 - Number of databytes bit 5 - Bit nine of Class. bit 6-7 - Reserved and should be set to 0.
Channel
Sequence number
Class (Lower 8 bits).
Type
Data byte(s) 0 - 15
Frame checksum (counted from address (low 8-bits of it). XOR checksum is used.
[DLE]
[ETX]

There can be 0 to 16 databytes.

The checksum is calculated from, and including, flags up to the last databyte.

The **channel** byte can be used to logically separate different channels on the same physical line. Typically this can also be a serial line with many nodes which are polled for data.

The **sequence number** byte marks a number for a sequence of a special frame. For instance four frame are send after each other and they will get different counter numbers. The NACK/ACKS that are responded from the node use the same counter values to separate the frames from others. Can, for example, be used to implement a sliding window protocol (http://en.wikipedia.org/wiki/Sliding_window [http://en.wikipedia.org/wiki/Sliding_window]).

As noted there can be two types of ACK's/NACK's one (251/252) is a general ACK/NACK that is the response when a frame is received and put in a buffer. This is generally all that is needed. The other version (249/250) can optionally be sent when the frame is actually sent on the interface.

Command frames have the following format

[DLE]
[STX]
255 = Command.
Flags Bit 7-5 reserved. bit 0-4 - Number of databytes.
Channel
Sequence number
Command MSB
Command LSB
Optional command Data byte(s) 0-15
Frame checksum
[DLE]
[ETX]

There can be 0 to 16 argument bytes.

When a command is sent their should always be a response. This response could be an ACK/NACK/Response/Error frame so all types should be expected.

Error frames have the following format

[DLE]
[STX]
252 = error.
Flags Bit 7-5 reserved. bit 0-4 - Number of databytes.
Channel
Sequence number
Error code MSB
Error code LSB
Optional error data byte(s) 0-15
Frame checksum
[DLE]
[ETX]

There can be 0 to 16 argument bytes.

Command Reply frames have the following format

[DLE]
[STX]
254 = response.
Flags Bit 7-5 reserved. bit 0-4 - Number of databytes.
Channel
Sequence number
Command Reply code MSB

Command Reply code LSB
Optional error data byte(s) 0-15
Frame checksum
[DLE]
[ETX]

There can be 0 to 16 argument bytes.

ACK frame have the following format

[DLE]
[STX]
251 = ACK.
Channel
Sequence number
Frame checksum
[DLE]
[ETX]

NACK frame have the following format

[DLE]
[STX]
252 = NACK.
Channel
Sequence number
Frame checksum
[DLE]
[ETX]

Special characters

The DLE character is sent as [DLE][DLE]

```
[DLE] = 0x10
[STX] = 0x02
[ETX] = 0x03
```

Sending CAN frames with the protocol

For generic CAN frames the operation byte is set to 5 The first four databytes form the CAN ID.
The four last databytes can be a timestamp (big endian).
Bit 31 (offset 0) of the id is Extended frame.
Bit 30 is remote frame.
Bit 29 is error frame.

Command codes

Command codes and data format is decided by the implementor.

Response codes

Response codes and data format is decided by the implementor.

Error codes

Error codes and data format is decided by the implementor.

vscp_specification_-_vscp_level_i_over_serial_links.txt · Last modified: 2008/04/17 13:51 by 87.249.175.45

Back to Specification
Physical Level

VSCP over RS-485/RS-422

This is a description of a low level protocol for RS-485. The protocol is a server based protocol and is low speed but very cheap to implement. At a very low additional cost a CAN network with full VSCP functionality and server less functionality can be constructed and this is recommended for the majority of systems.

```
Baudrate:** 9600 (or any other suitable baudate)
Format:** 8N1 for packages. 9N1 for address.
```

Addressing

Addressing is done with *nine bit addressing*. Address 0 is reserved for the server and the rest of the addresses are free to use for clients. Addresses are hardcoded within devices and each device should have its own unique address.

Packet format

Nine bit address
Operation 0 = No operation 1 = Level I event to/from node. 2 = Poll for event. (No data content follows, just CRC). 3 = No events to send (No data content follows, just CRC).
Flags + bit nine of the class **bit 0-3** - Number of databytes **bit 4** - Bit nine of Class. **bit 5-7** - reserved and should be set to 0.
Class (Lower 8 bits).
Type
Data byte 0
Data byte 1
Data byte 2
Data byte 3
Data byte 4
Data byte 5
Data byte 6
Data byte 7
Packet CRC (counted from address (low 8-bits of it). DOW checksum is used http://www.vscp.org/wiki/doku.php?id=vscp_specification_crc_polynoms [http://www.vscp.org/wiki/doku.php?id=vscp_specification_crc_polynoms].

If a node has a message to send it sends it directly after the poll message. The node should respond within one maximum packet time (14 * 8 * 104uS = 10 ms. Note that the event may be addressed to another node on the segment not only to the host.

If the node has no events to send it can either not reply to the poll or reply with "No Events to send" which is the preferable way.

The messages is standard VSCP Level I events in every other aspect.

A master on a 485 segment must check for new nodes on regular intervals. With 127 possible nodes this process could take 1.3 seconds to perform. Instead of doing all this at once it may be better to spread it over the standard polling range. That is to do a full polling of all active nodes and then try to discover a new node on an unused address etc etc.

vscp_specification_-_vscp_level_i_over_rs-485.txt · Last modified: 2008/04/09 11:15 (external edit)

Back to Specification
Physical Level

VSCP over TCP/IP

VSCP level II 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 is ideal to always have a full addresses in packets for this type of media. That means that the address part in the header which is 8-bits for Level I is instead 129 bits for Level II. That is the full GUID is available in the frame.

There are two different versions over TCP/IP. One UDP version which is broadcasted on a segment and the other which is more secure and use the TCP low level protocol.

VSCP over UDP

Port

```
Default Port: 9598
```

The port was 1681 up to release 0.1.6

Format

Information is 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 (**Set to zero!**)
```

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, Big Endian). 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 II.

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.

VSCP TCP Interface

This is the introduction of a TCP interface for the daemon. This would have a fair better chance to work over wireless and links that are hard to get reliable with UDP. Again it is not fully secure but just as we from time to time miss a character in our e-mails this can also happen here.

Secure enough is the design criteria.

GUID assigned to the i/f

Just as with all clients that connect to the daemon each TCP client interface get a GUID assigned to it. This GUID is assigned by setting the ethernet MAC address of the computer canald is running on using the VSCP Ethernet assignment method (se VSCP spec, Appendix A). The two lower byted are set to the interface id (obid). This result in a GUID of the following form

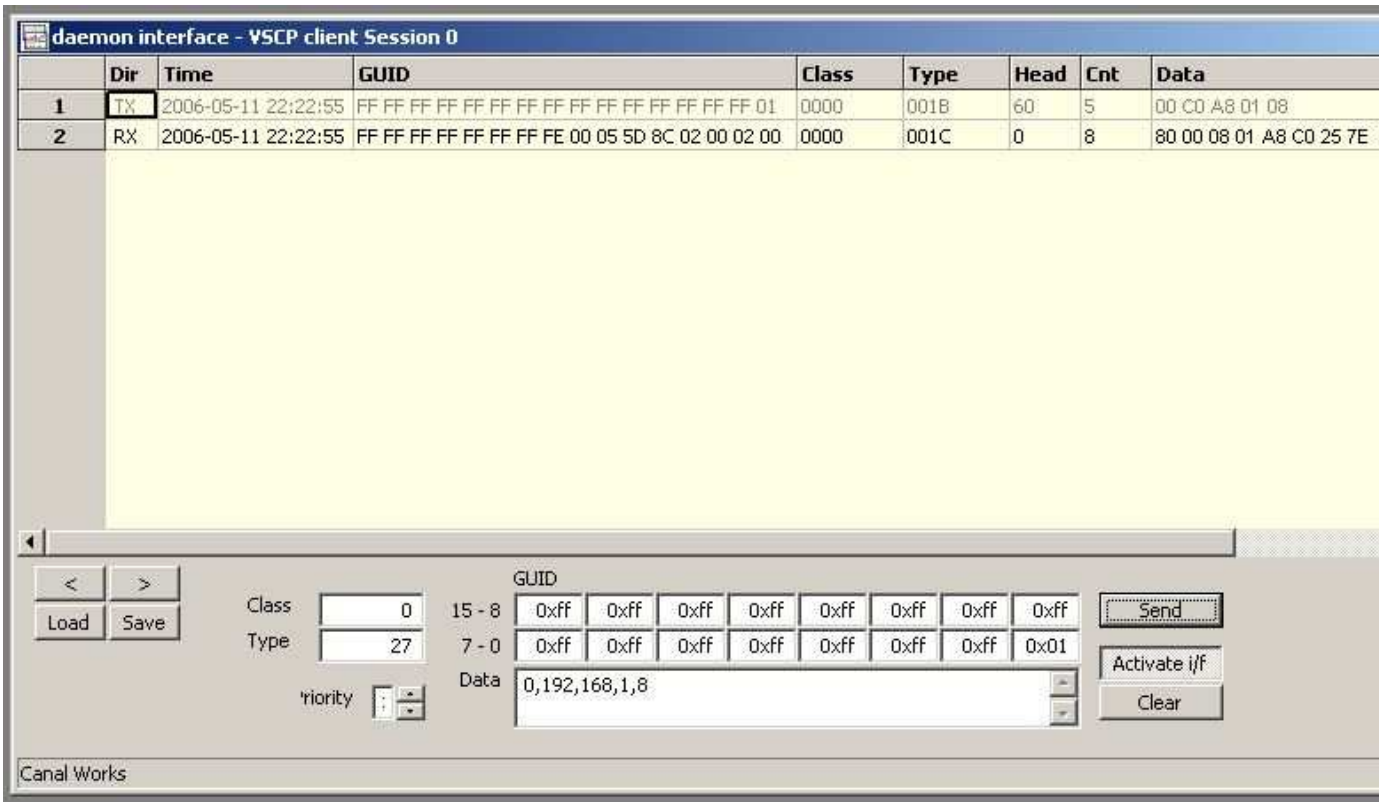
```
FF FF FF FF FF FF FF FE yy yy yy yy yy yy OBID_MSB OBID_LSB
```

where yy is the Ethernet address and FF is decimal 255.

Server discovery

A node that need a TCP connection to a server broadcast **High end server probe** Class=0, Type = 27 (0x1B) on the segment and waits for **High end server response** Class=0, Type = 28 (0x1C) from one or more servers that serve the segment. If a suitable server has responded it can decide to connect to that server.

A daemon like the VSCP daemon can span multiple segments and a reply can therefore be received from a remote segment as well. This can be an advantage in some cases and unwanted in some cases. The server configuration should have control on how it is handled.



As an example: In this picture CanalWorks has been used to broadcast a `High end server probe` from the machine with IP address 192.168.1.8 - Data is 0,192,168,1,8 - The initial zero indicates that this is a TCP/IP based device.

The server that in this case is on the same machine have answered with a `High end server response` with data set to

```
80 00 08 01 A8 C0 25 7E
```

The first two bytes is the bitfiled that tells the Code for server capabilities. After that follows the server IP address (192.168.1.8) followed by the VSCP default port 9598.

In clear text. This server have a VSCP TCP/IP interface available at the standard port.

Other scenarios could be possible of course such as several servers responding and each of the servers supporting different capabilities.

Secure the TCP link

The "USER" and the "PASS" sequency allow a username password to be used to secure the TCP link. Default is none.

To enable this two registry enteries are needed

```
tcpuser - should be set to the username (max 64 characters).
tcppassword - should be set to the hex representation of the 16 bytes of the 16 byte of the MD5 sum of the
```

A simple program will be supplied that calculate the MD5 for a given password.

The `High end server probe` and `High end server response` is described here Class=0 (0x00) VSCP Protocol Functionality - CLASS1.PROTOCOL

VSCP TCP Protocol Description

Port

```
Default Port: 9598
```

Command and response format

The VSCP TCP protocol is very much like the POP3 protocol.

- A command is sent (ending with CRLF)
- A response line is received starting with either ”+OK” for success or ”-OK” for failure. The initial “token” is followed by some descriptive text.

For some commands there can be data in between the two lines. For instance the “VERS” sommand looks like this

```
send: 'VERS<CR><LF>'
received line 1: '1,0,0<CR><LF>'
received line 2: '+OK - <CR><LF>'
```

Commands

```
 +    - Repeat the last command (added in version 0.2.0).
 NOOP - No operation.
 QUIT - Close the connection.
 USER - Username for login.
 PASS - Password for login.
 SEND - Send an event.
 RETR - Retrive one or several event(s).
 RCVLOOP - Will retrive events in an endless loop until the connection is closed by the client (added in ve
 CDTA - Check if there are events to retrive.
 CLRA - Clear all events in inqueue
 STAT - Get statistics information.
 INFO - Get status information.
 CHID - Get channel ID.
 SGID - Set GUID for channel.
 GGID - Get GUID for channel.
 VERS - Get CANAL/VSCP daemon version.
 SFLT - Set incoming event filter.
 SMSK - Set incoming event mask.
 BIN1/FAST - Enter binary mode 1.
 BIN2 - Enter binary mode 2.
 IFS1 - Lists Level I (CANAL) interfaces connected to the daemon (Added in version 0.2.0).
 IFS2 - Lists Level II interfaces connected to the daemon (Added in version 0.2.0).
 OUT1 - Open outgoing stream of events using standard text format (Added in version 0.2.0).
 OUT2 - Open outgoing stream of events using binary format (Added in version 0.2.0).
 HELP - Give command help (Added in version 0.2.0).
 TEST - Do test sequence. Only used for debugging (Added in version 0.2.0).
 SHUTDOWN - Shutdown the daemon (Added in version 0.2.0).
 RESTART  - Restart the daemon (Added in version 0.2.0).
```

Description of commands

NOOP - No operation.

This operation does nothing it just reply with ”+OK”.

QUIT - Close the connection.

Close the connection to the host.

USER - Username for login.

Used to enter username for a password protected server.

Used on the folowing form:

```
USER username<CR><LF>
```

PASS - Password for login.

Used to enter username for a password protected server.

Used on the following form:

```
PASS password<cr><lf>
```

SEND - Send an event.

Send an event.

Used on the following form:

```
send head,class,type,obid,timestamp,GUID,data1,data2,data3....
```

The GUID is given on the form MSB-byte:MSB-byte-1:MSB-byte-2……. The GUID can also be given as ”-” in which case the GUID of the interface is used for the event. This GUID is constructed from the Ethernet MAC address and some other parameters to be globaly unique.

If timestamp is set to zero a timestamp will be provided by the daemon. This timestamp will be set as soon as the the line is parsed.

Note: obid is just a place holder here to have a similar line as the receive command and is used internally by the daemon as an interface index. The value you use will always be overwritten by the daemon.

Example:

```
SEND 0,20,3,0,0,0:1:2:3:4:5:6:7:8:9:10:11:12:13:14:15,0,1,35
```

to send a full GUID event

```
SEND 0,20,3,0,0,-,0,1,35
```

is the same as above but the GUID of the interface will be used as GUID.

Both send the CLASS1.INFORMATION TYPE=3 ON event, for zone=1, subzone=35

RETR - Retrive one or several event(s).

This command can be used to retreive one or several events from the input queue. Events are returned as

```
head,class,type,obid,timestamp,GUID,data0,data1,data2,..........
```

GUID with MSB first.

Used on the following form:

```
RETR 2<CR><LF>
0,20,3,0,0,255:255:255:255:255:255:255:254:0:5:93:140:2:32:0:1
0,20,4,0,0,255:255:255:255:255:255:255:254:0:5:93:140:2:32:0:1
+OK -
```

If no events are available in the queue

1. OK - No event(s) available

is received by the client.

RCVLOOP - Send events to client as soon as they arrive.

This command set the channel in a closed loop that only can be interrupted by a client closing the connection. The server will now send out an event as soon as it is reserved. This is done in a very effective way with high throughput. This means the client does not have to poll for new events. It just open one channel where it sends events and do control tasks and one channel where it receive evens.

To help in determining that the line is alive

+OK

is sent with a two second interval. The format for the events is the same as for RETR command.

CDTA - Check if there are events to retrive.

This command are used to check how many events are in the input queue waiting for retrival.

Used on the following form:

```
CDTA<CR><LF>
8
+OK -
```

CLRA - Clear all events in inqueue

This command are used to clear all events in the input queue.

Used on the following form:

```
CLRA<CR><LF>
+OK - All events cleared.
```

STAT - Get statistics information.

Get interface statistics information. The returned format is

```
cntBusOff,cntBusWarnings,cntOverruns,cntReceiveData,cntReceiveFrames,cntTransmitData,cntTransmitFrames
```

Example:

```
STAT<CR><LF>
0,0,0,12356,56,9182,20<CR><LF>
+OK - <CR><LF>
```

INFO - Get status information.

This command fetch the status information for the interface. Returned format is

```
channel_status,lasterrorcode,lasterrorsubcode,lasterrorstr
```

Example:

```
INFO<CR><LF>
7812,12,0,"Overrun"<CR><LF>
-+OK - <CR><LF>
```

CHID - Get channel ID.

Get the channel id for the communication channel. This is the same parameter as the obid which is present in events.

SGID - Set GUID for channel.

Set the GUID for this channel. The GUID is givven on the form

The format is:

```
SGID 0:1:2:3:4:5:6:7:8:9:10:11:12:13:14:15<CR><LF>
+OK - <CR><LF>
```

GGID - Get GUID for channel.

Get the GUID for this channel. The GUID is received on the form

```
0:1:2:3:4:5:6:7:8:9:10:11:12:13:14:15<CR><LF>
+OK - <CR><LF>
```

VERS - Get CANAL/VSCP daemon version.

Get the current version for the daemon. The returned format is

```
major-version,minor-version,sub,minor-version
```

SFLT - Set incoming event filter.

Set the incoming filter. The format is

```
filter-priority, filter-class, filter-type, filter-GUID
```

SMSK - Set incoming event mask.

Set the incoming mask. The format is

```
mask-priority, mask-class, mask-type, mask-GUID
```

BIN1/FAST and BIN2 - Enter binary mode.

Note the 'FAST' keyword is deprected use BIN1 instead.

Enter binary or fast mode.

Note the two versions. FAST and BIN1 is a version that require the slave to request frames. The BIN2 version send outgoing frames to the slave directly when they arrive.

Constant for this mode is defined in canal.h and have “BINARY_” as prefix.

VSCP frame

Send this frame to the sever to send an event. The server will respond with an error frame but the error indication will be positive for a success. This is also the frame that will be returned after a read request if one or more events are available on the server.

```
0x55
0xaa
frame type: == 0 for VSCP frame
MSB of packet len (counted from packet type to last data byte)
LSB of packet len
```

```
VSCP head
VSCP class MSB
VSCP class LSB
VSCP type MSB
VSCP type LSB
VSCP GUID MSB
...
VSCP GUID LSB
data 0
data 1
....
data n
```

Error frame

```
0x55
0xaa
Frame type: == 1 for  Error Frame
0x00
0x01
error code, 0 is OK. Error codes defined in canal.h
```

CAN frame

```
0x55
0xaa
Frame type: == 3 for  CAN Frame
0x00
5 + number of data bytes.
flag (bit 0 - Extended id, bit 1 - RTR frame)
32 bit id msb
32 bit id
32 bit id
32 bit id lsb
data byte 0 .. 7
```

Command frames

Currently three commands are available

- 0 - No operation, NOOP.
- 1 - Read request
- 2 - Close

NOOP command frame

The noop command do nothing. The server will return a positive error frame.

```
0x55
0xaa
Frame type. == 2 for Command frame
0x00
0x01
0 for NOOP command
```

Read Request

A read request will get a VSCP frame in return or an erro frame. Typically indicating that there is no event(s) available.

```
0x55
0xaa
Frame type. == 2 for Command frame
0x00
0x01
1 for Read Request command
```

Close Request

The close request will return to the standard TCP/IP interface and if the intention is to close the TCP/IP interface a “QUIT” also have to be issued after this command.

```
0x55
0xaa
Frame type. == 2 for Command frame
0x00
0x01
3 for Close Request command
```

History

- 2006-07-27 OUT1/OUT2 commands added.
- 2006-07-27 Start of history.

Back to Specification
Physical Level

Description of the VSCP RF protocol

Preliminary

Comparision to "standard" protocols

Most standard protocols expect delivery to be point to point. One source have one consumer. For VSCP this should be rephrased to one source and an unknown number of consumers. So to implement VSCP over a standard protocol a frame is sent to each node in turn expecting an acknowledge frame in return from the node. Thus the number of frames sent is 2n where n is the number of nodes assuming transmission without faults. In a mesh network nodes has to know their neighbors so some additional traffic is needed for this.

For the VSCP RF Low level protocol one message is broadcasted from the originating node. All nodes that receive it will repeat it. Thus a total of n messages will be sent out and at the same time mesh networks will be handled automatically. The bandwidth taken is half that of the standard approach. Built into this is also the fact that a node that misses a message get several chances to acquire the message from repeating nodes. Something that is particularly good for RF where the most common disturbance is short bursts.

In VSCP when we send an event we don't want to know too much about the receiving nodes. We send out something we think might interest the world and each individual in this world makes a decision if they are interested in it or not. This helps us to keep the abstraction level high and we don't have to keep track of other nodes on the bus. Note however that for this protocol an originating node can keep track of its neighbors and retransmit a message if some node(s) don't repeat the message (confirms the message).

Operation Frequency

Nothing is said about the used transmission frequency in this document. VSCP over RF is designed to work on 433MHz as well as 868/915MHz and 2.4GHZ and other frequencys of choice

How many nodes can be connected?

Theoretically 254 nodes can be connected. This number is theoretical and will be limited by real world implemention data such as transmission speeds and other limiting factors.

How is the address for a node set?

A node address can be programmed in hardware or through switches. In this case the node address is hard coded and the hard coded bit, (bit 15), of the address should be set. For hard coded nodes the address is made up of two parts a common segment code (upper seven bits) and a unique Node code (lower eight bits).

For soft coded addresses a nickname discovery mechanisms is used when the node flirts is connected to the network. This nickname address lives with the node forever after it has been set.

The nickname discovery is simple and implemented in the following way:

1. A node connected to the network sends out a “nickname probe” with its own address as 0xff and the destination address set to zero. It then waits for a response from node zero (possible master). If received it will wait and let the master node set a nickname. If not received the discovery goes on.
2. The node send out a “nickname probe” with node id 1 and waits for an answer. And continue

with this with increasing nicknames until it gets a timeout.
3. To make certain that the node found is indeed a free nickname it repeats the discovery three times with this nickname id before acquiring it as its own. If a repose is revived during this the process is started all over from the beginning.

As a variant of the above suitable for installers a node can be setup by using the nickname discovery mechanism and have a installer tool which is setup as master. The node will then find the master when it is started up and wait for the master to set its address. After that other configuration parameters can also be set and the node is ready for installation and the installation tool /the master is not needed anymore.

Packet format

Preamble

Preamble to sync the data reception. Preamble is **0xdd** followed by **0×16**

Flags byte

A flag byte with protocol information.

```
Bit 7 - VSCP Class bit 9.
Bit 6 - House Code MSB.
Bit 5 - House Code
Bit 4 - House Code LSB
Bit 3 - Datasize MSB.
Bit 2 - Datasize.
Bit 1 - Datasize LSB.
Bit 0 - Datasize LSB.
```

The node id

The address for a node is set by a 8-bit number where the top most bit, bit 7, is used as a hard coded bit. This leaves room for 2^7-2 hard coded and 2^7-1 nickname nodes.

Two addresses are reserved. 0 which is reserved for a master and 0xff which is the broadcast address.

```
Node address 0x00 is master address (may or may not be present).
Node address 0x80 of master for hard coded nodes.
Node address 0xff is broadcast for both hard coded and soft coded nodes.
```

Hardcoded nodes are devided into segment code (upper three bits) and Node code (lower four bits).

index byte

The index byte consist of two parts.

```
bit 7-4 This index is increase by one for every new message a node sends out.
bit 3-0 Is set to zero for a new message. And increased by one before retransmiting a frame.
```

When a node received a message it should resend the message if

```
If the node is not the originating node of the message.
If it already received a message with and id built from the node-id + bit 7-4 of the index byte.
```

In all other cases the node retransmit the message as it is with the original node id and bits 7-4 of this byte intact but it increase bit 3-0 by one.

Originating address

The address for the node that send this message.

This is the same address used as above for the originating node but all nodes that repeat a message

inset there own node id here.

class

VSCP class MSB byte. Bit nine is in flags byte.

type

VSCP type MSB byte.

optional data block

VSCP Level I data. Max eight bytes.

CRC MSB/LSB

An implementor specific CRC from node-id and the full package length excluding the CRC itself.

Node id + index byte forms a unique message id (UMID).

To be able to determine if it has received messages a node should have a cache where it stores UMID's of sent and received messages. The cache should delete the oldest id when it puts in a new id into the cache. The minimum size of this cache is yet to be defined.

```
In a none mesh environment the implementor can choose to just retransmit the header including the CRC to sa
```

Message transmission

A node check for a carrier signal on the selected frequency band. If it is present it waits for as many milliseconds as stated by its MSB GUID byte. It then tries again and if a carrier is still present it waits the amount of milliseconds given by its GUID MSB less one and continue in that fashion until it can complete the transmission.

When a frame has been sent the node should listen for at least one repeat of its own transmitted frame and resend it if no repeat is received. The timeout is still to be defined.

Message Reception.

A received messages UMID is checked (the 20 most significant bits) if its in the cache. If its not there it is retransmitted after increasing the lower four bits by one.

If the node has a message of its own to send this retransmission should be performed before that transmission.

A protocol implementor can, but should preferable not, design a node that don't retransmit received frames. This can be useful for systems where some extra layer is present that secure arrival of frames. It should be noted that mesh capabilitiues will be list by doing this.

Dumb Nodes

For some nodes it is important to keep the processing overhead down to a minimum. This can be due to the node being battery powered or extremly low on hardware resources. A typical dumb node is one that sends out a measurement such as a temperature on even intervals. It normally does not matter if this data is lost from time to time.

A dumb node can have a hard coded address and only send out data. It does not have to repeat data or implement the nickname algorithm. It can also have the nickname discovery algorithm implemented and acquire its address that way.

Back to Specification

Level I Specifics

Level I Node types

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

Dynamic nodes

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 in 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 [http://www.vscp.org]

Hard coded nodes

This 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 sends out an event at certain times is typical and a button node that sends out an on-event when the button is pressed is another example.

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 accomplish this.

Node segment initialization

Dynamic nodes

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

Step 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.

Step 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 0×07. 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 (good practice also for CAN) it is recommended that the Probe is sent several time in a row to make sure that the nickname actually is free. This is actually a good method on all low level protocols and at least three tests are recommended.

Step 3

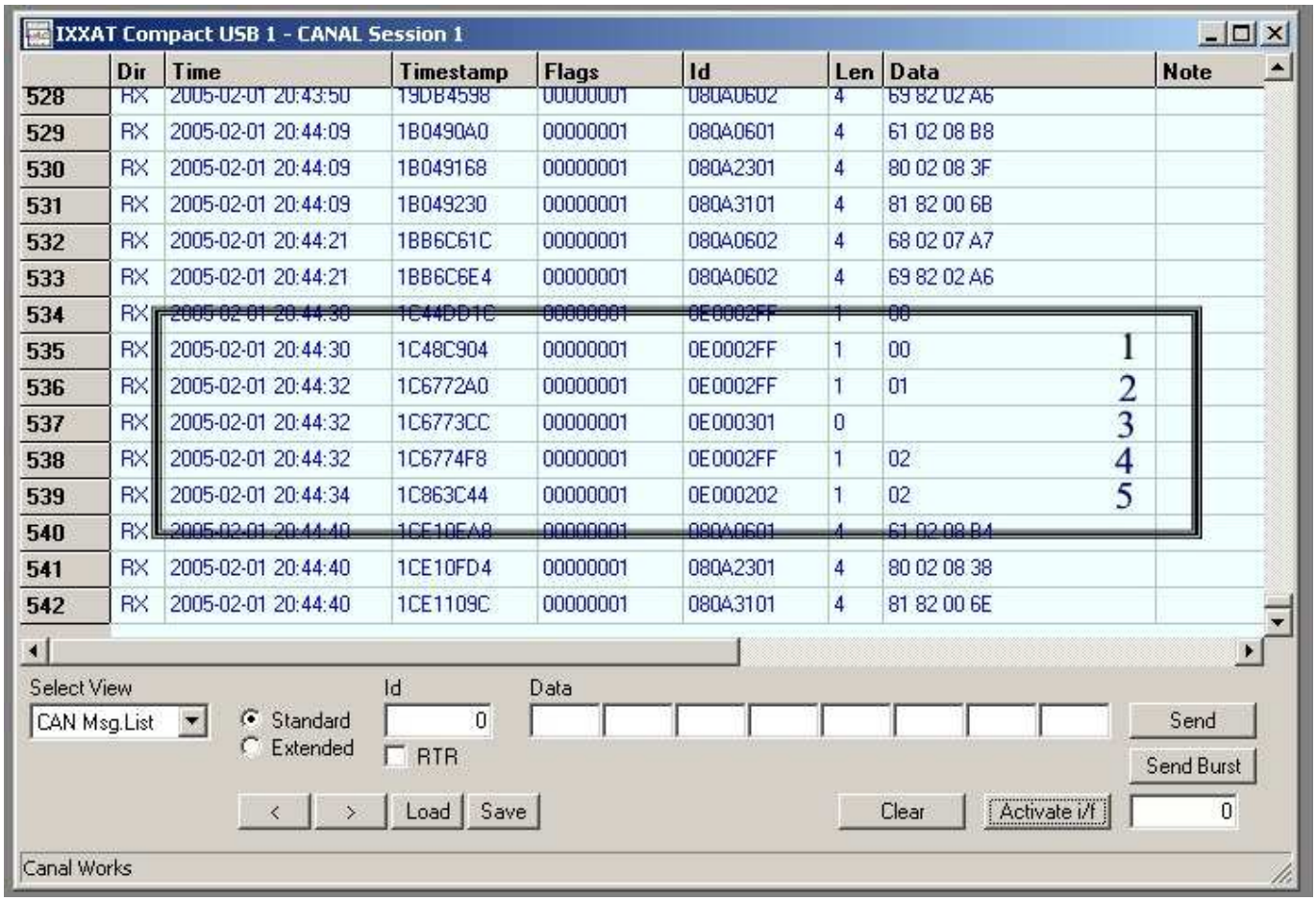
After it assigns a nickname to itself the node sends Nickname id accepted using its new nickname id to inform the segment of its identity.

Step 4

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

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

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



1. The node which initially has its nickname set to 0xff probe for a segment controller. Class = 0, Type = 2
2. No segment controller answers the probe and a new probe is therefore sent to a node with nickname=1. Again Class=0, Type=2.
3. There is a node with nickname= 1 already on the segment and it answers 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 concludes that the nickname=2 is free and assigns it to itself. It then sends a probe again with the new nickname assigned reporting a "new node on line".

Before an installation in a big system it is better to preassign id's to the nodes. This is just done by connecting each of the nodes to a PC or similar and assigning id's to each of them, one at the time. After that they can be installed at there location and will use this id for the rest if there life or until told otherwise.

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.

Silent dynamic nodes

Sometimes it can be an advantage to build modules that start up as silent nodes. This is typical for a RS-485 or similar segment. This type of nodes only listen to traffic before they get initialized by a host. In this case the nickname discovery process is not started for a node when it is powered up for the first time. This type on node instead starts to listen for the **CLASS1.PROTOCOL, Type=23 (GUID Drop nickname id / Reset Device.) event**. When this series of events is received and the GUID is the same as for the module the module starts the nickname discovery procedure as of above.

Using this method it is thus possible, for example, to let a user enter the GUID of a module and let some software search and initialize the node. As only the last four bytes of the GUID are unknown if the manufacturer GUID is known this is easy to enter for a user with the addition of a listbox for the manufacturer.

The active nickname process is still a better choice as it allows for automatic node discovery without user intervention and should always be the first choice.

This is an example on the way it works

You have two nodes and assign unique id's from there serial numbers

```
Node 1 have serial number 0001
Node 2 have serial number 0002
```

Combined with your GUID this will be

```
Node 1 have GUID aa bb cc dd 00 00 00 00 00 00 00 00 00 00 00 01
Node 2 have GUID aa bb cc dd 00 00 00 00 00 00 00 00 00 00 00 02
```

When you start up your nodes they see they don't have assigned nickname id (= 0xff) so they just sit back and listen.

When your PC app want to initialize the new nodes it sends

```
CLASS1.PROTOCOL Type 23
Data 00 aa bb cc dd
```

```
CLASS1.PROTOCOL Type 23
Data 01 00 00 00 00
```

```
CLASS1.PROTOCOL Type 23
Data 02 00 00 00 00
```

```
CLASS1.PROTOCOL Type 23
Data 03 00 00 00 01
```

At this stage your node knows it should enter the initialization phase and it will try to discover a new nickname using 0xff as its nickname.

This will be several **CLASS.PROTOCOL Type 2** staring with 0 (server) as tge probe id and increasing the probe id as long as **CLASS.PROTOCOL Type 3** is received (i.e other nodes say they use that id).

If this is the first node on the bus it will claim nickname id = 1. This id should (normally(be stored in EEPROM and will be used without the sequence above next time the node is started.

The PC now continue with the other node sending

```
CLASS1.PROTOCOL Type 23
Data 00 aa bb cc dd
```

```
CLASS1.PROTOCOL Type 23
Data 01 00 00 00 00
```

```
CLASS1.PROTOCOL Type 23
Data 02 00 00 00 00
```

```
CLASS1.PROTOCOL Type 23
Data 03 00 00 00 02
```

and this node will start its initialization procedure and find a free id.

Note that this process is only done the first time you put a fresh new node on a bus. Next time the id the node finds will be used directly from the time the node starts up.

If you always have a PC present let it send **CLASS1.PROTOCOL, Type 6** to the uninitialized node when the node send the probe to the server (.**CLASS.PROTOCOL Type 2** with probe id=0xff and origin 0xff)

Example

A typical scenario for a segment without a Master 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 checks 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 note of the id's. 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 turn are initialized and the segment unique nickname noted.

At this stage the switches and the relay 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 is received from node with nickname n3 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 is received from node with nickname n3 set relay off.

As the decision 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 decision matrix would look:

- 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 is received from node with nickname n3 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 like that 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 a switch to the scenario above and it can be even easier if the zone concepts are 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 I want a remote control that controls the lighting?

Just let the remote control interface have a decision matrix element that sends out the on-event to the zone when the selected key is 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….

vscp_specification_level_i_specifics.txt · Last modified: 2008/04/09 11:16 (external edit)

Back to Specification

Level II Specifics

Also check VSCP over TCP/IP which have more Level II specifics for the lower levels.

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.

Level II events

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 CRC polynoms used by VSCP

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.

XML Representation

VSCP level II events can be presented as XML data. Format is

```
<?xml version = "1.0" encoding = "UTF-8" ?>

<!-- Version 0.0.1      2007-09-27 -->

<event>
    <class>Eventclass is numerical form or CLASSx:numerical form</class>
    <type>Event type in numerical form.</type>
    <guid>ff:ee:dd:cc:bb:aa:99:88:77:66:55:44:33:22:11:00</guid>   <!-- GUID of sending node -->
    <timestamp>YYYY-MM-DD HH:MM:SS|millieconds since epoc</timestamp>
    <data>Comma separated list with event data. Hex values (precede with '0x')and decimal values allowed.</
</event>
```

Back to Specification

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 [http://www.vscp.org]

Back to Specification

Register Abstraction Model

Byte wide bit registers are a central part of the VSCP specification. All nodes Level I as well as Level II should be possible to be configured by reading/writing these registers. Some of the register are the same for all nodes and are also the same between Level I and Level II nodes. The difference is that Level II can have a lot more registers defined.

Registers 0×00 – 0x7f are application specific. Registers between 0×80-0xff are reserved for VSCP usage. If the node has implemented the decision matrix it is stored in application register space.

With Node control flags (0×83) 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 (0×92/0×93) 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 128 * 65535 bytes (65535 pages of 128 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.

0×98 Buffer size for device is information for control nodes of limitations of the buffer space for a Level II node. Level I nodes should have eight (8) 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 than eight and the implementor whishes to do so. This even if the event is a Level I event.

The VSCP registers are defined as follows:

Address	Access Mode	Description
0×00 – 0x7f	—	Device specific
128/0×80	Read Only	Alarm status register content (!= 0 indicates alarm). Condition is reset by a read operation. The bits represent different alarm conditions.
129/0×81	Read Only	VSCP Major version number this device is constructed for.
130/0×82	Read Only	VSCP Minor version number this device is constructed for.
131/0×83	Read/Write	**Node control flags** bit 7 –Startup control bit 6 – Startup control bit 5 – r/w control of registers below 0×80. (1 means write enabled) bit 4 – Reserved bit 3 – Reserved bit 2 – Reserved bit 1 – Reserved bit 0 – Reserved Startup control set to 01xxxxxx means initialized
132/0×84	Read/Write	User ID 0 – Client settable node id byte 0.
133/0×85	Read/Write	User ID 1 – Client settable node id byte 1.
134/0×86	Read/Write	User ID 2 – Client settable node id byte 2.
135/0×87	Read/Write	User ID 3 – Client settable node id byte 3.
136/0×88	Read/Write	User ID 4 – Client settable node id byte 4.
137/0×89	Read only	Manufacturer device ID byte 0.
138/0x8a	Read only	Manufacturer device ID byte 1.
139/0x8b	Read only	Manufacturer device ID byte 2.
140/0x8c	Read only	Manufacturer device ID byte 3.

141/0x8d	Read only	Manufacturer sub device ID byte 0.
142/0x8e	Read only	Manufacturer sub device ID byte 1.
143/0x8f	Read only	Manufacturer sub device ID byte 2.
144/0×90	Read only	Manufacturer sub device ID byte 3.
145/0×91	Read only	Nickname id for node if assigned or 0xff if no nickname id assigned.
146/0×92	Read/Write	Page select register MSB
147/0×93	Read/Write	Page Select register LSB
148/0×94	Read Only	Firmware major version number.
149/0×95	Read Only	Firmware minor version number.
150/0×96	Read Only	Firmware sub minor version number.
151/0×97	Read Only	Boot loader algorithm used. 0Xff for no boot loader support. Codes for algorithms are specified here VSCP Specification Class 0 for Type = 12
152/0×98	Read Only	Buffer size. The value here gives an indication for clients that want to talk to this node if it can support the larger mid level Level I control messages which has the full GUID. If set to 0 the default size should used. That is 8 bytes for Level I and 512-25 for Level II.
153/0×99	Read Only	Number of register pages used.
154/0x9A-207/0xcf	—	Reserved for future use. If not implemented one page is assumed.
208/0xd0-223/0xdf	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).
224/0xe0-255/0xff	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. For a node with an embedded MDF return a zero string. The CLASS1.PROTOCOL, Type=34/35 can then be used to get the information if available.

The page select registers can be used to switch pages for the lower 128 byte. Its important that an application that uses the page functionality switch back to page 0×0000 when its ready. Applications can use the registers as a Mutex which is signaled when not zero.

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 has a configuration model where data for a node can be read and written in XML format. Its up to the design of the node which method to use as long as the required registers are implemented. It is however recommended that both methods are present so a register read/write could be used as a last resort to configure also a high end device.

Back to Specification

Decision Matrix

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 model

To be able to build a software model that is common for several device/machine/situation types and environments we have to 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 1	byte 2	byte 3	byte 4	byte 5	byte 6	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. 0×00 is segment controller and 0xff is a node without a nickname. If bit 6 of flags is set oaddr will not be checked and events from all nodes will be accepted.

flag

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

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

The zone and use subzone bits can be activated to have a check on the zone/subzone information of an event. That is the zone/subzone of the machine must match the one of the event to trigger the DM row.

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

Think of the mask as having ones at positions that are of interest and the filter telling what the value should be for those bit positions.

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

- To accept all classes set the mask to 0. In this case filter don't care.

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 0×00 is predefined and means No-Operation. All other codes are application specific and typical application defined codes could 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 Decision 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 (< 0×80) or in an optional page in this area.

Note that there is no demand that a node implements a decision matrix. If not implemented the Get Decision 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 0×80 - 8 i.e. Starts at 0×78 (120) it is implied that the register position just below 0×77(119) contains a register that is an index to the active DM.

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:

byte 0-3	byte 4-7	byte 8-11	byte 12-13	byte 14-n
mask	filter	control	Action	action-param

where the action-param has device specific length.

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 #	Description
31	Enabled if set to 1. If disabled the decision matrix element is ignored.
30	GUID should be checked (=1) or not checked (=0). Action Param should start with GUID at offest 0
29	Marks end of matrix. No more valid enteris in the DM after this line. Used to save parsing resources.
28	Reserved.
27	Reserved.
26	Reserved.
25	Reserved.
24	Reserved.
23	Reserved.
22	Reserved.
21	Reserved.
20	Reserved.
19	Reserved.
18	Reserved.
17	Reserved.
16	Reserved.
15	Reserved.
14	Reserved.
13	Reserved.
12	Reserved.
11	Reserved.
10	Reserved.
9	Reserved.
8	Reserved.
7	Reserved.
6	Reserved.
5	Match index of this module to trigger DM entry. Byte 0 of data.
4	Match Zone of this module to trigger DM entry. Byte 1 of data.
3	Match Subzone of this module to trigger DM entry. Byte 2 of data.
2	Reserved.
1	Reserved.
0	Reserved.

Specific node implement ions decide how to interpret bits or not. Just some or none of the bits can be used if that suits the implement ion.

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. 0×0000 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 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 0×80 - dm-row-size (a DM row aligned to the upper register border) it is implied that the register position just below, 0×80 -dm-row-size - 1, contains a register that is an index to the row that has its first byte at 0×80 - dm-row-size.

The index byte is used to select rows and the selected row is available from the byte after the index byte to the 0×80 border.

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.

The Level II Decision Matrix has no entry for originating address. The action parameter field can be used for this information if needed.

vscp_specification_decision_matrix.txt · Last modified: 2008/04/09 11:16 (external edit)

Back to Specification

Data coding

For the measurement class and the data class 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.

Definitions for bits in byte 0

Tells how data that follows should be interpreted. This is used for CLASS1.MEASUREMENT and CLASS1.DATA

Bits 5,6,7

Bits	Descriptions
5,6,7	Represent one of several numerical representations in which the data that follows is represented in

000b Bit Format

The data should be represented as a set of bits. This can be used for picture coding etc.

001b Byte Format(0x20)

The data should be represented as a set of bytes.

010b String Format(0x40)

The data should be represented as an ASCII string (possibly with language extensions (8-bit)).

011b Integer Format(0x60)

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 total event length=2 the data is a 1 byte integer or 1 byte.
If total event length=3 the data is a 16-bit integer or 2 bytes.
If total event length=4 the data is a 24-bit integer or 3 bytes.
If total event length=5 the data is a 32-bit integer or 4 bytes.
If total event length=6 the data is a 40-bit integer or 5 bytes.
If total event length=7 the data is a 48-bit integer or 6 bytes.
If total event length=8 the data is a 56-bit integer or 7 bytes.
```

100b Normalized Integer Format(0x40)

Data is coded as a normalized integer. In this case the format byte is followed by the normalizer byte which have bit 7 set if the value that is represented by the integer that follow should have a decimal point that is shifted to the right. I bit 7 is reset the decimal point should be shifted to the left. Thus a set bit 7 represent a number between -1 amd +1.

The normalizer bytes lower six bits tells how many position the decimal point should be shifted left or right. If bit 7 is set i.e. indicates a value < 1 the decimal point should be shifted in the left direction and if itbit 7 is cleared it is a value > 1 and the decimal point should be shifted in the right direction.

The actual integer is coded in the bytes that follows and can be 1-6 bytes where the most significant byte always is in byte 2. This integer is always signed and always given as a two's complement number.

example

```
0x02 0x1B 0x22
0x1B22 =  6946 Decimal
0x02 has bit 7 cleared meaning the decimal point be two steps to the right i.e. the value is 69.46
```

example

```
0x85 0xF1 0x07
0xF107 =  -3833 Decimal
0x85 has bit 7 set meaning the decimal point be five steps to the left i.e. the value is -0.03833
```

101b Floating point value(0xA0)

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 [http://www.psc.edu/general/software/packages/ieee/ieee.html] and further info at http://en.wikipedia.org/wiki/IEEE_754-1985 [http://en.wikipedia.org/wiki/IEEE_754-1985]

110b Reserved.(0xC0)

The format is yet to be defined.

111b Reserved(0xE0).

The format is yet to be defined.

Bits 3,4

Bits	Descriptions
3,4	Indicates how the data should be interpreted.

00b

Standard format. All other codes in this field are message class/type specific.

Bits 0,1,2

Bits	Descriptions
0,1,2	Zero based sensor index which can be used if there are more then one sensor handled by the node.

Back to Specification

Module Description File

The VSCP registers 0xe0-0xff specifies the Module Description File URL (without "http://" which is implied). 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.

```
Coding: UTF-8
```

Real life file samples

example1 - eurosource Kelvin SHT sensor - http://www.dofscandinavia.com/sht_001.xml [http://www.dofscandinavia.com/sht_001.xml]
example2 - eurosource Kelvin Smart2 sensor - http://www.dofscandinavia.com/smart2_001.xml [http://www.dofscandinavia.com/smart2_001.xml]

XML Format Specification

Some notes
Register definitions must be available for all nodes (if it has registers defined). A register which does not have a an abstraction defined will be handled with a default abstraction constructed from its offset as

```
register''offset''
```

for example

```
register1
register40
```

The type will always be "uint8_t" in this case

```
<?xml version = "1.0" encoding = "UTF-8" ?>

<!-- Version 0.0.4      2008-01-09
  "string" - Text string
  "bitfield"  - a field of bits. Width tells how many bits the field consist if (max eight bits).
  "bool"      - 1 bit number specified as true or false
  "int8_t"    - 8  bit number. Hexadecimal if it starts with "0x" else decimal
  "uint8_t"   - Unsigned 8  bit number. Hexadecimal if it starts with "0x" else decimal
  "int16_t"   - 16 bit signed numbr. Hexadecimal if it starts with "0x" else decimal
  "uint16_t"  - 16 bit unsigned number. Hexadecimal if it starts with "0x" else decimal
  "int32_t"   - 32 bit signed numbr. Hexadecimal if it starts with "0x" else decimal
  "uint32_t"  - 32 bit unsigned number. Hexadecimal if it starts with "0x" else decimal
  "int64_t"   - 64 bit signed number. Hexadecimal if it starts with "0x" else decimal
  "uint64_t"  - 64 bit unsigned number. Hexadecimal if it starts with "0x" else decimal
  "decimal"   - 128 bit number. Hexadecimal if it starts with "0x" else decimal
  "date"      - Must be passed in the format dd-mmm-yyyy
  "time"      - Must be passed in the format hh:mm:ss where hh is 24 hour clock
-->

<!-- General section -->


<vscp>
<module>  <!-- one file can contain one or several modules -->

   <name>aaaaaaaa</name>
   <model>bbbbb</model>
   <version>cccccc</version>
```

```
   <description lang = "en">yyyyyyyyyyyyyyyyyyyyyyyyyyyy</description>
   <!-- Site with info about the product -->
   <infourl>http://www.somewhere.com</infourl>
   <!-- Max package size a node can receive -->
   <buffersize></buffersize>
   <manufacturer>
        <name>ttttttttttttttttttt</name>
        <address>
           <street>ttttttttttttt</street>
           <town>lllllllll</town>
           <city>London</city>
           <postcode>HH1234</postcode>
           <!-- Use region or state -->
           <state></state>
           <region></region>
           <country>ttttt</country>
       </address>
       <!-- One or many -->
       <telephone>
           <number>123456789</number>
           <description lang="en" >Main Reception</description>
       </telephone>
       <!-- One or many -->
       <fax>
           <number>1234567879</number>
           <description lang="en">Main Fax Number</description>
       </fax>
       <!-- One or many -->
       <email>
           <address>someone@somwhere.com</description>
           <description> lang="en">Main email address</description>
       </email>
       <!-- One or many -->
       <web>
           <address>www.somewhere.com</address>
           <description>Main web address</address>
       </web>
    </manufacturer>


<!-- Settings Section
 Types are defined here. An abstraction is something that maps to a register which is
 specified by page/offset and is of a predefined type
 The id is the tag that can be used in Level II configuration events.
-->

<abstractions>

    <!--
        The abstract variable "somename" is defined as a boolean type which can have a value 0 or 1
        (system also recognize false/true). This variable is located at page=0, offset=1 at bit=0. As this
        boolean the system knows it ocupies one bit. Other types may need "width" to be defined also. The v
        can be read and written.

        Note that the name of the "variable" can be set to different thing depending on locale.

        Note the difference between "id" and "name". "id" is the same for a certain abstraction for all lan
        and what is used internally by system software. "name" is what is presented to the user.
    -->
    <abstraction id="somename"
                 type="bool"
                 default="false" page = "0" offset = "1" bit="0" >
        <name lang="en">tttt</name>
        <description lang="en">yyy</description>
        <help type="text/url"  lang="en">tttt</help>
        <access>rw</access>
    </abstraction>

   <!--
       The abstract variable "alsoaname" is just defined as a short. That is a 16-bit signed integer.
       It has a default value of 182 and is located at page=0 offset=15 and 16 (Bigendian). The variable ca
       read and written.
   -->
   <abstraction id="alsoaname"
                type="short"
                default="182" page = "0" offset = "15" >
        <name lang="en">tttt</name>
        <description lang="en">yyy</description>
        <help type="text/url"  lang="en">tttt</help>
        <access>rw</access>
   </abstraction>

   <!--
       Here a abstract variable "adescriptivename" of type string is defined. A width is needed here and th
       string is stored in page=0 at offset=20-21. Read write access is possible
   -->
   <abstraction id="adescriptivename"
                type="string"
                width="12"
```

```
                default="" page = "0" offset = "20" >
        <name lang="en">tttt</name>
        <description lang="en">yyy</description>
        <help type="text/url"  lang="en">tttt</help>
        <access>rw</access>
    </abstraction>

    <!--
        This example shows a valuelist stored in an integer. This is typically presented to the user as
        a listbox or a combobox with values to choose from.
        If register space is limited it can be more efficient to use a bitfield for the options.
    -->
    <abstraction id="namedlist"
                type="integer"
                default="" page = "0" offset = "100" >
        <name lang="en">tttt</name>
        <description lang="en">yyy</description>
        <help type="text/url"  lang="en">tttt</help>
        <access>r</access>
        <valuelist>
            <item value = "0x0">
                <name lang="en">item0</name>
                <description lang="en">item0_description</description>
            </item>
            <item value = "0x1">
                <name lang="en">item1</name>
                <description lang="en">item1_description</description>
            </item>
            <item value = "0x2">
                <name lang="en">item2</name>
                <description lang="en">item2_description</description>
            </item>
            <item value = "0x3">
                <name lang="en">item3</name>
                <description lang="en">item3_description</description>
            </item>
            <item value = "0x4">
                <name lang="en">"item4</name>
                <description lang="en">item4_description</description>
            </item>
            <item value = "0x5">
                <name lang="en">item5</name>
                <description lang="en">item5_description</description>
            </item>
            <item value = "0x6">
                <name lang="en">item6</name>
                <description lang="en">item6_description</description>
            </item>
            <item value = "0x7">
                <name lang="en">item7</name>
                <description lang="en">item7_description</description>
            </item>
            <item  value = "0x8">
                <name lang="en">item8</name>
                <description lang="en">item8_description</description>
            </item>
        </valuelist>

    </abstraction>

</abstractions>


<!-- Register section -->

<registers>

    <!-- The following is abstraction "alsoaname"
        described in register space by two reg items.
    -->
    <reg page="0" offset="15" default="0" >
        <name lang="en">alsoaname_msb</name>
        <description lang="en">MSB of alsoaname</description>
        <help type="text/url"  lang="en">tttt</help>
        <access>rw</access>
    </reg>
    <reg page="0" offset="16" >
        <help type="text/url"  lang="en">tttt</help>
        <name lang="en">alsoaname_lsb</name>
        <description lang="en">LSB of alsoaname</description>
        <access>rw</access>
    </reg>

    <!-- The following is abstraction "adescriptivename"
        described in register space. Note the use of "width"
        which can be used to tell how many registers an abstraction
        see instead of having many register defines. Default width
        is one byte.
    -->
```

```
<reg page="0" offset="15" width="12" >
    <help type="text/url"  lang="en">tttt</help>
    <name lang="en">abcdefghih</name>
    <description lang="en">The string adescriptivename</description>
    <access>rw</access>
</reg>

<!--
     This example shows how individual bits in a register is defined. Note that each bit can be named.
     Note also at pos 4 (a bit position) where a bit field has been defined which is four bits wide. He
     a valuelist also could have been defined describing the possible values.

     All eight bites in register at page=0, offset=1 is described here.
-->
<reg page="0" offset="1" >

    <help type="text/url"  lang="en">tttt</help>
    <name lang="en">tttt</name>
    <description lang="en">yyy</description>
    <access>rw</access>

    <bit pos="0" default="false" >
        <name lang="en">tttt</name>
        <description lang="en">yyy</description>
        <help type="text/url"  lang="en">tttt</help>
    </bit>

    <bit pos="1">
        <name lang="en">tttt</name>
        <description lang="en">yyy</description>
        <help type="text/url"  lang="en">tttt</help>
    </bit>

    <bit pos="2">
        <name lang="en">tttt</name>
        <description lang="en">yyy</description>
        <help type="text/url"  lang="en">tttt</help>
    </bit>

    <bit pos="3">
        <name lang="en">tttt</name>
        <description lang="en">yyy</description>
        <help type="text/url"  lang="en">tttt</help>
    </bit>

    <bit pos="4" width="4">  <!-- example for bit groups -->
        <name lang="en">tttt</name>
        <description lang="en">yyy</description>
        <help type="text/url"  lang="en">tttt</help>
    </bit>
</reg>

<!--
     Here a bitfield with width of six bits has been defined. Note the access rights for the field. If
     access rights is not given read+write access is presumed.
     The register iteself is not give a name here just the bit field.
-->
<reg page="0" offset="2">

    <bit pos="2" width="6">

        <name lang="en">tttt</name>
        <description lang="en">yyy</description>
        <help type="text/url"  lang="en">tttt</help>
        <access>rw</access>

        <valuelist>
            <item value = "0x0">
                <name lang="en">undefined</name>
                <description lang="en">yyy</description>
                <help type="text/url"  lang="en">tttt</help>
            </item>
            <item value = "0x1">
                <name lang="en">Normal</name>
                <description lang="en">yyy</description>
                <help type="text/url"  lang="en">tttt</help>
            </item>
            <item value = "0x2">
                <name lang="en">Error</name>
                <description lang="en">yyy</description>
                <help type="text/url"  lang="en">tttt</help>
            </item>
            <item value = "0x3">
                <name lang="en">Disabled</name>
                <description lang="en">yyy</description>
                <help type="text/url"  lang="en">tttt</help>
            </item>
            <item value = "0x4">
                <name lang="en">"Test</name>
```

```
                <description lang="en">yyy</description>
                <help type="text/url"  lang="en">tttt</help>
            </item>
            <item value = "0x5">
                <name lang="en">Disposed</name>
                <description lang="en">yyy</description>
                <help type="text/url"  lang="en">tttt</help>
            </item>
            <item value = "0x6">
                <name lang="en">PowerSaving</name>
                <description lang="en">yyy</description>
                <help type="text/url"  lang="en">tttt</help>
            </item>
            <item value = "0x7">
                <name lang="en">Stopped</name>
                <description lang="en">yyy</description>
                <help type="text/url"  lang="en">tttt</help>
            </item>
            <item  value = "0x8">
                <name lang="en">Paused</name>
                <description lang="en">yyy</description>
                <help type="text/url"  lang="en">tttt</help>
            </item>
        </valuelist>
    </bit>
</reg>

<!--
     Here all bits of a register is used as a value list which is only readable.
-->
<reg page = "0" offset = "88">
    <name lang="en">tttt</name>
    <description lang="en">yyy</description>
    <help type="text/url"  lang="en">tttt</help>
    <access>r</access>
    <valuelist>
        <item value = "0x0">
            <name lang="en">undefined</name>
            <description lang="en">yyy</description>
            <help type="text/url"  lang="en">tttt</help>
        </item>
        <item value = "0x1">
            <name lang="en">Normal</name>
            <description lang="en">yyy</description>
            <help type="text/url"  lang="en">tttt</help>
        </item>
        <item value = "0x2">
            <name lang="en">Error</name>
            <description lang="en">yyy</description>
            <help type="text/url"  lang="en">tttt</help>
        </item>
        <item value = "0x3">
            <name lang="en">Disabled</name>
            <description lang="en">yyy</description>
            <help type="text/url"  lang="en">tttt</help>
        </item>
        <item value = "0x4">
            <name lang="en">"Test</name>
            <description lang="en">yyy</description>
            <help type="text/url"  lang="en">tttt</help>
        </item>
        <item value = "0x5">
            <name lang="en">Disposed</name>
            <description lang="en">yyy</description>
            <help type="text/url"  lang="en">tttt</help>
        </item>
        <item value = "0x6">
            <name lang="en">PowerSaving</name>
            <description lang="en">yyy</description>
            <help type="text/url"  lang="en">tttt</help>
        </item>
        <item value = "0x7">
            <name lang="en">Stopped</name>
            <description lang="en">yyy</description>
            <help type="text/url"  lang="en">tttt</help>
        </item>
        <item  value = "0x8">
            <name lang="en">Paused</name>
            <description lang="en">yyy</description>
            <help type="text/url"  lang="en">tttt</help>
        </item>
    </valuelist>
</reg>

<!-- Example where min/max is used -->
<reg page = "0" offset = "88" min="8" max="32">
    <name lang="en">Trust</name>
    <description lang="en">yyy</description>
    <help type="text/url"  lang="en">tttt</help>
```

```
    </reg>

</registers>



<!-- Decison matrix -->
<dmatrix>

    <help type="text/url"  lang="en">tttt</help>

    <!-- Can durrently be 1 or 2 -->
    <level>1</level>
    <!-- Where is matrix located -->
    <start page="0" offset="1"/>
    <!-- # of rows in matrix -->
    <rowcnt>10</rowcnt>
    <!-- Size in bytes for one row - only for level II -->
    <rowsize></rowsize>
    <!-- Code for action !-->
    <action code="0x0">
        <name lang="en"></name>
        <description lang="en"></description>
        <help type="text/url"  lang="en">tttt</help>
        <!-- Descriptions of parameters - one or many -->
        <param>
            <name lang="en"></name>
            <description lang="en"></description>
            <help type="text/url"  lang="en">tttt</help>
            <!-- Just one pos for Level I -->
            <data offset="1" >
                <name lang="en">tttt</name>
                <description lang="en">yyy</description>
                <help type="text/url"  lang="en">tttt</help>
                <bit pos="0">
                    <name lang="en">tttt</name>
                    <description lang="en">yyy</description>
                    <help type="text/url"  lang="en">tttt</help>
                </bit>
                <!-- valuelist could also be used in bit groups and for hole byte -->
            </data>
        </param>
    </action>
</dmatrix>


<!-- Events this module can generate -->
<events>
    <event class="0" type="10" >

        <help type="text/url"  lang="en">tttt</help>

        <!-- Optional: user event name -->
        <name lang="en"></name>
        <!-- Why and when event is sent -->
        <description lang="en"></description>
        <help type="text/url"  lang="en">tttt</help>
        <!-- Optional: What priority it will be sent as -->
        <priority>3</priority>
        <!-- Information about evenet data -->
        <data offset="1" >
            <name lang="en">tttt</name>
            <description lang="en">yyy</description>
            <help type="text/url"  lang="en">tttt</help>
            <bit pos="0">
                <name lang="en">tttt</name>
                <description lang="en">yyy</description>
                <help type="text/url"  lang="en">tttt</help>
            </bit>
        </data>
    </event>
</events>

<!-- A valuelist can be used here as well -->



<!-- Description/specification for alarm bits -->
<alarm>
    <bit pos="1">
        <name lang="en">tttt</name>
        <description lang="en">yyy</description>
        <help type="text/url"  lang="en">tttt</help>
    </bit>
</alarm>


<!-- bootlader information -->
```

```
<boot>
    <!-- bootloader algorithm tha can be used on this module -->
    <algorithm>1</algorithm>
    <!-- Size of boot block/sector -->
    <blocksize>20</blocksize>
    <!-- Number of available boot blocks/sectors -->
    <blockcount>66</blockcount>
</boot>

</module>
</vscp>
```

vscp_specification_mdf.txt · Last modified: 2008/05/03 14:46 by 87.249.175.45

Back to Specification

Configuration Model

Preliminary Level II configuration **Do not use!!!**

For the level II part of the protocol a higher level 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 and specify this in the settings section of the MDF file. Module Description File

XML format for configuration events (requests/responses)

```
<?xml version = "1.0" encoding = "UTF-8" ?>
<config xmlns:xs=http://www.w3.org/2001/XMLSchema>
   <name>20</name>
</config>
```

Where name is taken either from `name` in

```
<name type = "string" lang = "en" description = "Description of this particular device parameter"/>
```

or from the `id` value in

```
<abstraction id="somename"
              type="bool"
              default="false" page = "0" offset = "1" bit="0" >
     <name lang="en">tttt</name>
     <description lang="en">yyy</description>
     <access>rw</access>
 </abstraction>
```

The type of the value is thus defined in the MDF.

Also the register`offset` implied names can be used. See Module Description File

The abstraction tag is also valid for Level I and maps registers and abstract types together. A higher level node can from this information decide if it wants to look at the data/parameters a node holds from the register perspective (which must be implemented for all VSCP nodes, both Level I and Level II) or from a higher level.

This can be used to specify a `short` which in register space is stored as a MSB byte and a LSB byte but which can be handled as the higher layer type `short` by all routines above the actual register read/write which use the information in the MDF to know how to store the abstract value.

Level II High Level Configuration

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" (CLASS1.PROTOCOL1, TYPE= 1).

Nodes respond to this message by sending a NodeHeartbeat (CLASS1.INFORMATION, TYPE=9).

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

1. A `ConfigReadRequest` event is sent. Index set to 0. The target node is addressed with its GUID.
2. Node responds with one or several `ConfigReadResponse` with the code portion indicating the current page / total pages and data in byte 9 and onwards.
3. Requesting node receives all the `ConfigReadResponse` events togehter forming the configuration. Note that if a `ConfigReadRequest` is detected whilst compiling the

`ConfigReadResponse` message then the process is abandoned.

1. The requesting node can edit the configuration for a node and then send out a `ConfigChanged` event to indicate that the config has changed for this node by supplying its GUID.
2. The node interested in this can now request the new configuration by issuing a `ConfigUpdateRequest`.
3. The node that changed the configuration will now notice this event and send out a series of `ConfigUpdateResponse` events.
4. The target node received the events.

Example of response or config send from/to node

```
<?xml version = "1.0" encoding = "UTF-8" ?>
   <settings>
      <MSensor1>20</MSensor1>
      <MSensor2>800</MSensor2>
   </settings>
```

is a response from a node

vscp_specification_configuration_model.txt · Last modified: 2008/05/04 22:27 by 87.249.175.45

Back to Specification

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.

VSCP Multicast

For VSCP multicast the address

224.0.23.158 VSCP

should be used.

Please see the following: http://www.iana.org/assignments/multicast-addresses [http://www.iana.org/assignments/multicast-addresses]
and http://www.tldp.org/HOWTO/Multicast-HOWTO.html [http://www.tldp.org/HOWTO/Multicast-HOWTO.html]

vscp_specification_vscp_multicast.txt · Last modified: 2008/04/09 11:16 (external edit)

Back to Specification

Level I Events

Class=0 (0x00) VSCP Protocol Functionality - CLASS1.PROTOCOL
Class=1 (0x01) Alarm - CLASS1.ALARM
Class=2 (0x02) Security - CLASS1.SECURITY
Class=10 (0x0A) Measurement - CLASS1.MEASUREMENT
Class=15 (0x0F) Data - CLASS1.DATA
Class=20 (0x14) Information - CLASS1.INFORMATION
Class=30 (0x1E) Control - CLASS1.CONTROL
Class=40 (0x28) Multimedia - CLASS1.MULTIMEDIA
Class=100 (0x64) Phone - CLASS1.PHONE
Class=102 (0x66) Display i/f - CLASS1.DISPLAY
Class=110 (0x6E) IR Remote I/f - CLASS1.IR
Class=200 (0xC8) 1-Wire protocol i/f - CLASS1.ONEWIRE
Class=201 (0xC9) X10 protocol i/f - CLASS1.X10
Class=202 (0xCA) LON Works protocol i/f - CLASS1.LON
Class=203 (0xCB) EIB protocol i/f - CLASS1.EIB
Class=204 (0xCC) S.N.A.P. protocol i/f - CLASS1.SNAP
Class=205 (0xCD) CBUS protocol i/f - CLASS1.CBUS
Class=206 (0xCE) GPS i/f - CLASS1.GPS
Class=212 (0xD4) Wireless i/f - CLASS1.WIRELESS
Class=509 (0x1FD) Log i/f - CLASS1.LOG
Class=510 (0x1FE) Laboratory use - CLASS1.LABORATORY
Class=511 (0x1FF) Local use - CLASS1.LOCAL

Back to Specification
Back to Level I Events

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 with the only difference that GUID's are used instead of nicknames.

Type = 0 (0x00) Undefined.

Undefined protocol function.

Type = 1 (0x01) Segment Controller Heartbeat.

Implement in device if needed by application. Not mandatory.

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. *A node that receive (and recognize) this event should respond with a CLASS1.INFORMATION, Type=9 event.*

Other nodes can originate this event on the segment. For these nodes the data part, as specified below, should be omitted. A better choice for periodic heartbeat events from a node may be CLASS1.INFORMATION, Type=9

All nodes that recognize this event 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 and is not mandatory.

byte #	Description
Byte 0	8-bit CRC of the segment controller GUID.
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.

Mandatory. Must be implemented by all devices.

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 #	Description
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.

Mandatory. Must be implemented by all devices.

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

Type = 4 (0x04) Reserved for future use.

Reserved.

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

Reserved.

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

Mandatory. Must be implemented by all devices.

This event can be used to change the nickname for a node.

byte #	Description
byte 0	Old nickname for node.
byte 1	The new nickname for the node.

Type = 7 (0x07) Nickname id accepted.

Mandatory. Must be implemented by all devices.

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.

Mandatory. Must be implemented by all devices.

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.

There is a variant of this where the GUID is used instead of the nickname to identify the device, Type = 23.

Type = 9 (0x09) Read register.

Mandatory. Must be implemented by all devices.

Read a register from a node.

byte #	Description
byte 0	Node address.
byte 1	Register to read.

A read/write response message is returned on success.

The following format can be used for nodes on a Level II segment as a midway between a full Level II handling as specified in Class=1024 and Level I.

byte #	Description
byte 0-15	GUID.
byte 16	Register to read.

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

Mandatory. Must be implemented by all devices.

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 #	Description
byte 0	Register read/written.
byte 1	Content of register.

Type = 11 (0x0B) Write Register.

Mandatory. Must be implemented by all devices.

Write register content to a node.

byte #	Description
byte 0	Node address.
byte 1	Register to write.
byte 2	Content for register.

A read/write response message is returned on success.

The following format can be used for nodes on a Level II segment as a midway between a full Level II handling as specified in Class=1024 and Level I.

byte #	Description
byte 0-15	GUID.
byte 16	Register to write.
Byte 17	Content of register.

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

Mandatory. Send NACK (Type=14 if no bootloader implemented)

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

byte #	Description
byte 0	The nickname for the node.
byte 1	Code that select boot loader algorithm to use
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 0×92, Page select MSB.
byte 7	Content of register 0×93, Page select LSB.

The following format can be used for nodes on a Level II segment as a midway between a full Level II handling as specified in Class=1024 and Level I.

byte #	Description
byte 0-15	GUID.
byte 16	Bootloader algorithm code.

Bootloader Code

Code	Description
1	Microchip PIC algorithm 0 (VSCP PIC Boot loader)
2-8	Reserved
9	Atmel AVR algorithm 0
10-15	Reserved
16	NXP ARM algorithm 0
17-23	Reserved
24	ST ARM algorithm 0
25-255	Reserved

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

Only needed if a VSCP bootloader algorithm is used.

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 #	Description
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.

Mandatory. Should be implemented by all devices.

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 #	Description
Byte 0	Optional user defined error code.

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

Only needed if a VSCP bootloader algorithm is used.

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 #	Description
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.

Only needed if a VSCP bootloader algorithm is used.

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 #	Description
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.

Only needed if a VSCP bootloader algorithm is used.

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 #	Description
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.

Only needed if a VSCP bootloader algorithm is used.

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 #	Description
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

Only needed if a VSCP bootloader algorithm is used.

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 #	Description
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

Only needed if a VSCP bootloader algorithm is used.

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 #	Description
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

Only needed if a VSCP bootloader algorithm is used.

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 #	Description
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

Only needed if a VSCP bootloader algorithm is used.

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. NA CK boot loader mode will be sent if the CRC is not correct and the node will leave boot loader mode.

byte #	Description
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) GUID Drop nickname id / Reset Device.

Mandatory. Should be implemented by all devices.

```
Added in version 1.4.0
```

This is a variant of CLASS1_PROTOCOL, Type=8 where the full GUID is used instead of the nickname to identify the node that should drop its current nickname and enter the nodename discovery procedure.

As the GUID is 16 bytes this is a multiframe event. To ease the storage requirements on the nodes only for GUID bytes are send in each frame. The framse must be sent out within one second interval.

Byte	Content
0	index
1	GUID byte
2	GUID byte
3	GUID byte
4	GUID byte

where index goes from 0-3 and GUID bytes are sent MSB first, like

Index	Byte 2	Byte 3	Byte 4	Byte 5
Index = 0	GUID byte 15	GUID byte 14	GUID byte 13	GUID byte 12
Index = 1	GUID byte 11	GUID byte 10	GUID byte 9	GUID byte 8
Index = 2	GUID byte 7	GUID byte 6	GUID byte 5	GUID byte 4
Index = 3	GUID byte 3	GUID byte 2	GUID byte 1	GUID byte 0

A device can use just one byte to detect this. This byte is initialized to zero and holds four bits that match correct frames. That is, when this register is equal to 0x0f the nickname should be dropped and the nickname discovery sequence started. The node must also have a timer that reset this byte one second after any of the above frames have been received or when the nickname discovery sequence is started.

Hi-level software must take this one second interval into account when more then one node should be initialized. This event can be used to assign nickname id's to silent nodes. This is nodes that does not start the nickname discovery process on startup and instead just sits and wait until they are assigned an id with this event.

Type = 24 (0x18) Page Read

Mandatory. Should be implemented by all devices.

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 #	Description
Byte 0	Node ID who's registers should be read.
Byte 1	Index into page.
Byte 2	Number of bytes to read.

The following format can be used for nodes on a Level II segment as a midway between a full Level II handling as specified in Class=1024 and Level I.

byte #	Description
byte 0-15	GUID.
byte 16	Index into page.
byte 17	Number of bytes to read.

Type = 25 (0x19) Page Write

Mandatory. Should be implemented by all devices.

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 #	Description
Byte 0	Base index.
Byte 1-7	Data.

The following format can be used for nodes on a Level II segment as a midway between a full Level II handling as specified in Class=1024 and Level I.

byte #	Description
byte 0-15	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

Mandatory. Should be implemented by all devices.

This is a response frame for the read page command. The Sequence number goes from 0 up to the last sent frame for a read page request.

byte #	Description
Byte 0	Sequence number.
Byte 1-7	Data.

The following format can be used for nodes on a Level II segment as a midway between a full Level II handling as specified in Class=1024 and Level I.

byte #	Description
byte 0-15	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) High end server probe

Should be implemented by all devices that work on Ethernet/Internet.

This event can be broadcasted on a segment by a node to get information about available servers.

byte #	Description
Byte 0	Code for node type
Byte 1-7	Meaning depends on code in byte 0

Code node type byte

Code	Description
0	this is a TCPIP node, byte 1-4 is the V4 IP address of the node.

Type = 28 (0x1C) High end server response

Should be implemented by all devices that work on Ethernet/Internet.

byte #	Description
Byte 0	Code for server capabilities MSB
Byte 1	Code for server capabilities LSB
Byte 2	Server IP address MSB - or other relevant data as of server capabilities (Network byte order)
Byte 3	Server IP address - or other relevant data as of server capabilities (Network byte order)
Byte 4	Server IP address - or other relevant data as of server capabilities (Network byte order)
Byte 5	Server IP address LSB - or other relevant data as of server capabilities (Network byte order)
Byte 6	Server Port MSB - or other relevant data as of server capabilities
Byte 7	Server Port LSB - or other relevant data as of server capabilities

Bit #	Description

15	VSCP TCP server - data is V4 IP address and port
14	Reserved
12	Reserved
11	VSCP UDP server - data is V4 IP address and port
10	Reserved
9	Reserved
8	VSCP TCP SSL secure server link - data is V4 IP address and port
7	Reserved
6	Reserved
5	Reserved
4	Reserved
3	Reserved
2	Reserved
1	Reserved
0	Reserved

A node that need a TCP connection to a host. Broadcast HIGH END SERVER PROBE on the segment and waits for HIGH END SERVER RESPONSE from one or more servers to connect to. If a suitable server has responded it can decide to connect to that server.

A daemon like the canal daemon can span multiple segments and a reply can therefore be received from a remote segment as well. This can be an advantage in some cases and unwanted in some cases. The server configuration should have control on how it is handled.

Type = 29 (0x1D) Increment register

Mandatory. Should be implemented by all devices.

Increment a register content by one with no risk of it changing in between

byte #	Description
Byte 0	Value to increment by

If no value is supplied the register is incremented with one.

Node should answere with Read/Write register response. CLASS1.PROTOCOL, Type=10

Type = 30 (0x1E) Decrement register

Mandatory. Should be implemented by all devices.

Decrement a register content by one with no risk of it changing in between

byte #	Description
Byte 0	Value to decrement by

If no value is supplied the register is decremented by one.

Node should answere with Read/Write register response. CLASS1.PROTOCOL, Type=10

Type = 31 (0x1F) Who is there?

Mandatory. Must be implemented by all devices.

This event can be used as a fast way to find out which nodes there is on a segment. All nodes receiving it should respond.

byte #	Description
Byte 0	Node id or 0xff for all nodes.

Response from node(s) looks like this

byte 0	byte 1	byte 2	byte 3	byte 4	byte 5	byte 6	byte 7
0	GUID0	GUID1	GUID2	GUID3	GUID4	GUID5	GUID6
1	GUID7	GUID8	GUID9	GUID10	GUID11	GUID12	GUID13
2	GUID14	GUID15	MDF0	MDF1	MDF2	MDF3	MDF4
3	MDF5	MDF6	MDF7	MDF8	MDF9	MDF10	MDF11
4	MDF12	MDF13	MDF14	MDF15	MDF16	MDF17	MDF18
5	MDF19	MDF20	MDF21	MDF22	MDF23	MDF24	MDF25
6	MDF26	MDF27	MDF28	MDF29	MDF30	MDF31	0

Type = 32 (0x20) Get decision matrix info

Mandatory if the device got a decision matrix.

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

byte #	Description
Byte 0	Node address.

The following format can be used for nodes on a Level II segment as a midway between a full Level II handling as specified in Class=1024 and Level I.

byte #	Description
byte 0-15	GUID.

Type = 33 (0x21) Decision matrix info response

Mandatory if the device got a decision matrix.

Report the size for the decision 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 0×00 (See the register model in this document). If the size returned is zero the node does not have a decision matrix. *A node without a decision matrix can also skip to implement this event.*

byte #	Description
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 matrix can as noted be stored in paged registers and if so it must be accessed with the paged read/write. Level I If the offset given is 0×80-8 = 0×78 the matrix is paged and it is implied that rows are selected in a register located at register position **0×80-8–1 = 0×77**. 0 means row 1 is in the register space, 1 = row 2 is in register space and so on.

Register pos	Description
0×77	Index for row in decision matrix.
0×78-0x7F	Level I decision matrix row.

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 register position **0×80-dm-row-size-1** is an index to the row that is available in the decision matrix space.

Register pos	Description
(0×80-dm-row-size-1)	Index for row in decision matrix.
(0×80-dm-row-size)- 0x7F	Level II decision matrix row.

Type = 34 (0x22) Get embedded MDF.

Optional. Nodes that does not implement this event just don't respond to it.

A node that get this event and has an embedded MDF description in flash or simular respond with the description .

Type = 35 (0x23) Embedded MDF response.

Optional. See Type=34

This is the response from a *Get embedded MDF*. The response consist of several frames where an index in byte0/1 is incremented for each frame and MDF data is in byte 2-7.

byte #	Description
byte 0	High byte of MDF description index.
byte 1	Low byte of MDF description index.
byte 2-7	MDF data.

Type = 36 (0x24) Extended Read register.

Mandatory. Must be implemented by all devices.

Read a register from a node with page information.

byte #	Description
byte 0	Node address.
byte 1	MSB of page where the register is located.
Byte 2	LSB of page where the register is located.
byte 3	Register to read (offset into page).
byte 4	Optional: 1,2,3,4,5,6 for bytes to read.

An extended read/write response message is returned on success.

This means that a register (or a maximum of six consecutive registers) located on any page can be read in a single operation.

The following format can be used for nodes on a Level II segment as a midway between a full Level II handling as specified in Class=1024 and Level I.

byte #	Description
byte 0-15	GUID.
byte 16	MSB of page where the register is located.
Byte 17	LSB of page where the register is located.
byte 18	Register to read.
byte 19	Optional: bytes to read (1-255).

Type=37 (0x25) Extended Read/Write response.

Mandatory. Must be implemented by all devices.

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 #	Description
byte 0	MSB of page where the register is located.
byte 1	LSB of page where the register is located.
byte 2	Register read/written.
byte 3	Content of register.
byte 4-7	Content of register if multi register read.

Type = 38 (0x26) Extended Write Register.

Mandatory. Must be implemented by all devices.

Write register content to a node.

byte #	Description
byte 0	Node address.
byte 1	MSB of page where the register is located.
byte 2	LSB of page where the register is located.
byte 3	Register to write.
byte 4	Content for register.
byte 5,6,7; Optional extra data bytes to write.	

A read/write response message is returned on success.

Event allows a register (or a maximum of four consecutive registers) located on any page can be written in a single operation.

The following format can be used for nodes on a Level II segment as a midway between a full Level II handling as specified in Class=1024 and Level I.

byte #	Description
byte 0-15	GUID.
byte 16	MSB of page where the register is located.
byte 17	LSB of page where the register is located.
byte 18	Register to write.
Byte 19	Content of register.
byte 20-buffersize	Optional extra data bytes to write.

Type = 39 (0x27) Extended Page Read

Mandatory. Must be implemented by all devices.

This is the same event as CLASS1.PROTOCOL, Type=24 but the page is included in the event data so there is no need to change the page in register space. Repliee are the same as for the above event.

byte #	Description
Byte 0	Node ID who's registers should be read.
byte 1	MSB of page where the register is located.
byte 2	LSB of page where the register is located.
Byte 3	Index into page.
Byte 4	Number of bytes to read.

The following format can be used for nodes on a Level II segment as a midway between a full Level

II handling as specified in Class=1024 and Level I.

byte #	Description
byte 0-15	GUID.
byte 16	MSB of page where the register is located.
byte 17	LSB of page where the register is located.
byte 18	Index into page.
byte 19	Number of bytes to read.

Type = 40 (0x28) Extended Page Write

Mandatory. Must be implemented by all devices.

This is the same event as CLASS1.PROTOCOL, Type=25 but the page is included in the event data so there is no need to change the page in register space. Repliee are the same as for the above event.

byte #	Description
byte 0	MSB of page where the register is located.
byte 1	LSB of page where the register is located.
Byte 2	Base index.
Byte 3-7	Data.

The following format can be used for nodes on a Level II segment as a midway between a full Level II handling as specified in Class=1024 and Level I.

byte #	Description
byte 0-15	GUID.
byte 16	MSB of page where the register is located.
byte 17	LSB of page where the register is located.
byte 18	Base index.
byte 19-…	Data.

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

Back to Specification
Back to Level I Events

Class=1 (0x01) Alarm

CLASS1.ALARM

Description

Alarm events that indicate that something 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.

byte #	Description
byte 0	User defined data.
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

If both or one of zone/sub zone are omitted they should be interpreted as if they where 255.

Type = 2 (0x02) Alarm occurred.

Indicates an alarm condition.

byte #	Description
byte 0	User defined data.
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

If both or one of zone/sub zone are omitted they should be interpreted as if they where 255.

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

Alarm sound should be turned on or off.

byte #	Description
Byte 0	If equal to zero turn off else turn on.
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

If both or one of zone/sub zone are omitted they should be interpreted as if they where 255.

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

Alarm light should be turned on or off.

byte #	Description
Byte 0	If equal to zero turn off else turn on.
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

If both or one of zone/sub zone are omitted they should be interpreted as if they where 255.

Type = 5 (0x05) Power on/off

Power has been lost or is available again.

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

If both or one of zone/sub zone are omitted they should be interpreted as if they where 255.

Type = 6 (0x06) Emergency Stop

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

byte #	Description
Byte 0	User defined data.
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

If both or one of zone/sub zone are omitted they should be interpreted as if they where 255.

Type = 7 (0x07) Emergency Pause

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

byte #	Description
Byte 0	User defined data.
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

If both or one of zone/sub zone are omitted they should be interpreted as if they where 255.

Type = 8 (0x08) Emergency Reset

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

byte #	Description
Byte 0	User defined data.
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

If both or one of zone/sub zone are omitted they should be interpreted as if they where 255.

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 .

byte #	Description
Byte 0	User defined data.
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

If both or one of zone/sub zone are omitted they should be interpreted as if they where 255.

Back to Specification
Back to Level I Events

Class=2 (0x02) Security

CLASS1.SECURITY

Description

Security related events for alarms and simular devices.

Type = 0 (0x00) undefined

Undefined security issue.

Type = 1 (0x01) Motion Detect

A motion has been detected.

byte #	Description
byte 0	User defined data.
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

If both or one of zone/sub zone are omitted they should be interpreted as if they where 255.

Back to Specification
Back to Level I Events

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 #	Description
byte 0	Data coding.
byte 1-7	Data with format defined by byte 0

Type = 2 (0x02) Length/Distance

```
Default unit: Meter.
```

This is a measurement of a length or a distance.

byte #	Description
byte 0	Data coding.
byte 1-7	Data with format defined by byte 0

Type = 3 (0x03) Mass

```
Default unit: Kilogram.
```

This is a measurement of a mass.

byte #	Description
byte 0	Data coding.
byte 1-7	Data with format defined by byte 0

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 #	Description

byte 0	Data coding.
byte 1-7	Data with format defined by byte 0

Type = 5 (0x05) Electric Current

```
Default unit: Ampere.
```

This is a measurement of an electric current.

byte #	Description
byte 0	Data coding.
byte 1-7	Data with format defined by byte 0

Type = 6 (0x06) Temperature

```
Default unit: Kelvin.
Opt. unit: Degree Celsius (1), Fahrenheit (2)
```

This is a measurement of a temperature.

byte #	Description
byte 0	Data coding.
byte 1-7	Data with format defined by byte 0

Type = 7 (0x07) Amount of substance

```
Default unit: Mole.
```

This is a measurement of an amount of a substance.

byte #	Description
byte 0	Data coding.
byte 1-7	Data with format defined by byte 0

Type = 8 (0x08) Intensity of light

```
Default unit: Candela.
```

This is a measurement of luminous intensity.

byte #	Description
byte 0	Data coding.
byte 1-7	Data with format defined by byte 0

Type = 9 (0x09) Frequency

```
Default unit: Hertz.
```

This is a measurement of regular events during a second.

byte #	Description
byte 0	Data coding.
byte 1-7	Data with format defined by byte 0

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 #	Description
byte 0	Data coding.
byte 1-7	Data with format defined by byte 0

Type = 11 (0x0B) Force

```
Default unit: Newton.
```

This is a measurement of force.

byte #	Description
byte 0	Data coding.
byte 1-7	Data with format defined by byte 0

Type = 12 (0x0C) Pressure

```
Default unit: Pascal.
```

This is a measurement of pressure.

byte #	Description
byte 0	Data coding.
byte 1-7	Data with format defined by byte 0

Type = 13 (0x0D) Energy

```
Default unit: Joule.
```

This is a measurement of energy.

byte #	Description
byte 0	Data coding.
byte 1-7	Data with format defined by byte 0

Type = 14 (0x0E) Power

Default unit: Watt.

This is a measurement of power.

byte #	Description
byte 0	Data coding.
byte 1-7	Data with format defined by byte 0

Type = 15 (0x0F) Electrical Charge

Default unit: Coulomb.

This is a measurement electrical charge.

byte #	Description
byte 0	Data coding.
byte 1-7	Data with format defined by byte 0

Type = 16 (0x10) Electrical Potential (Voltage)

```
Default unit: Volt.
```

This is a measurement of electrical potential.

byte #	Description
byte 0	Data coding.
byte 1-7	Data with format defined by byte 0

Type = 17 (0x11) Electrical Capacitance

```
Default unit: Farad.
```

This is a measurement of electric capacitance.

byte #	Description
byte 0	Data coding.
byte 1-7	Data with format defined by byte 0

Type = 18 (0x012) Electrical Resistance

```
Default unit: Ohm.
```

This is a measurement of resistance.

byte #	Description
byte 0	Data coding.
byte 1-7	Data with format defined by byte 0

Type = 19 (0x13) Electrical Conductance

```
Default unit: Siemens.
```

This is a measurement of electrical conductance.

byte #	Description
byte 0	Data coding.
byte 1-7	Data with format defined by byte 0

Type = 20 (0x14) Magnetic Field Strength

```
Default unit: Ampere meters.
```

This is a measurement of magnetic field strength.

byte #	Description
byte 0	Data coding.
byte 1-7	Data with format defined by byte 0

Type = 21 (0x15) Magnetic Flux

```
Default unit: Weber.
```

This is a measurement of magnetic flux.

byte #	Description
byte 0	Data coding.
byte 1-7	Data with format defined by byte 0

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 #	Description
byte 0	Data coding.
byte 1-7	Data with format defined by byte 0

Type = 23 (0x17) Inductance

```
Default unit: Henry.
```

This is a measurement of inductance.

byte #	Description
byte 0	Data coding.
byte 1-7	Data with format defined by byte 0

Type = 24 (0x18) Flux of light

```
Default unit: Lumen.
```

This is a measurement of flux of light.

byte #	Description
byte 0	Data coding.
byte 1-7	Data with format defined by byte 0

Type = 25 (0x19) Iluminance

```
Default unit: Lux.
```

This is a measurement of luminance (illumination)

byte #	Description
byte 0	Data coding.
byte 1-7	Data with format defined by byte 0

Type = 26 (0x1A) Radiation dose

```
Default unit: Gray.
Opt unit: Sievert.
```

This is a measurement of a radiation dose.

byte #	Description
byte 0	Data coding.
byte 1-7	Data with format defined by byte 0

Type = 27 (0x1B) Catalytic activity

```
Default unit: Katal.
```

This is a measurement of catalytic activity used in biochemistry.

byte #	Description
byte 0	Data coding.
byte 1-7	Data with format defined by byte 0

Type = 28 (0x1C) Volume

```
Default unit: stere, square meter.
Opt. Unit: Liter (dm3).
```

This is a measurement of volume.

byte #	Description
byte 0	Data coding.
byte 1-7	Data with format defined by byte 0

Type = 29 (0x1D) Sound intensity

```
Default unit: Bel.
Opt Unit: Neper.
```

This is a measurement of sound intensity.

byte #	Description
byte 0	Data coding.
byte 1-7	Data with format defined by byte 0

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 #	Description
byte 0	Data coding.
byte 1-7	Data with format defined by byte 0

Type = 31 (0x1F) Position

```
Default unit: Longitude/Latitude.
```

This is a measurement of a position.

byte #	Description
byte 0	Data coding.
byte 1-7	Data with format defined by byte 0

Type = 32 (0x20) Speed

```
Default unit: Meters per second.
```

This is a measurement of a speed.

byte #	Description
byte 0	Data coding.
byte 1-7	Data with format defined by byte 0

Type = 33 (0x21) Acceleration

```
Default unit: Meters per second/second.
```

This is a measurement of acceleration.

byte #	Description
byte 0	Data coding.
byte 1-7	Data with format defined by byte 0

Type = 34 (0x22) Tension

```
Default unit: N/m.
```

This is a measurement of tension.

byte #	Description
byte 0	Data coding.
byte 1-7	Data with format defined by byte 0

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

```
Default unit: Relative percentage 0-100%.
```

This is a measurement of relative moistness (Humidity).

byte #	Description
byte 0	Data coding.
byte 1-7	Data with format defined by byte 0

Type = 36 (0x24) Flow

```
Default unit: Square meters/second.
Opt Unit: Liter/Second.
```

This is a measurement of flow.

byte #	Description
byte 0	Data coding.
byte 1-7	Data with format defined by byte 0

Type = 37 (0x25) Thermal resistance

```
Default unit: Thermal ohm K/W.
```

This is a measurement of thermal resistance.

byte #	Description
byte 0	Data coding.
byte 1-7	Data with format defined by byte 0

Type = 38 (0x26) Rafractive power

```
Default unit: Diopter (dpt) m-1.
```

This is a measurement of refractive power.

byte #	Description
byte 0	Data coding.
byte 1-7	Data with format defined by byte 0

Type = 39 (0x27) Dynamic viscosity

```
Default unit: Poiseuille (Pl) Pa . s.
```

This is a measurement of dynamic viscosity.

byte #	Description
byte 0	Data coding.
byte 1-7	Data with format defined by byte 0

Type = 40 (0x28) Sound impedance

```
Default unit: Rayal Pa . s/m.
```

This is a measurement of sound impedance.

byte #	Description
byte 0	Data coding.
byte 1-7	Data with format defined by byte 0

Type = 41 (0x29) Sound resistance

```
Default unit: Acoustic ohm Pa . s/ m3.
```

This is a measurement of refractive power.

byte #	Description
byte 0	Data coding.
byte 1-7	Data with format defined by byte 0

Type = 42 (0x2A) Electric elastance

```
Default unit: Darag F-1.
```

This is a measurement of electric elasticity.

byte #	Description
byte 0	Data coding.
byte 1-7	Data with format defined by byte 0

Type = 43 (0x2B) Luminous energy

```
Default unit: Talbot lm . s.
```

This is a measurement of luminous energy.

byte #	Description
byte 0	Data coding.
byte 1-7	Data with format defined by byte 0

Type = 44 (0x2C) Luminance

```
Default unit: Nit (nt) cd/m2.
```

This is a measurement of luminance.

byte #	Description
byte 0	Data coding.
byte 1-7	Data with format defined by byte 0

Type = 45 (0x2D) Chemical concentration

```
Default unit: Molal mol/kg.
```

This is a measurement of chemical concentration.

byte #	Description
byte 0	Data coding.
byte 1-7	Data with format defined by byte 0

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

```
Default unit: Gray J/Kg.
```

This is a measurement of absorbed dose of ionizing radiation.

byte #	Description
byte 0	Data coding.
byte 1-7	Data with format defined by byte 0

Type = 47 (0x2F) Dose equivalent

Default unit: Sievert J/Kg.

This is a measurement of dose equivalent.

byte #	Description
byte 0	Data coding.
byte 1-7	Data with format defined by byte 0

Type = 48 (0x30) Light flux

Default unit: Lumen cd . sr.

This is a measurement of light flux.

byte #	Description
byte 0	Data coding.
byte 1-7	Data with format defined by byte 0

Type = 49 (0x31) Dew Point

```
Default unit: Kelvin.
Opt. unit: Degree Celsius (1), Fahrenheit (2)
```

This is a measurement of the Dew Point.

byte #	Description
byte 0	Data coding.
byte 1-7	Data with format defined by byte 0

Type = 50 (0x32) Relative Level

```
Default unit: Relative value.
```

This is a relative value for a level meaurement without a unit. It is just relative to the min/max value for the selected data representation.

byte #	Description
byte 0	Data coding.
byte 1-7	Data with format defined by byte 0

Type = 51 (0x33) Altitude.

Altitude in meters.

Default unit: Meter. Opt. unit: Feet(1), inches (2)

byte #	Description
byte 0	Data coding.
byte 1-7	Altitude.

Back to Specification
Back to Level I Events

Class=15 (0x0f) Data

CLASS1.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 #	Description
byte 0	Data coding.
byte 1-7	Data with format defined by byte 0

Type = 2 (0x02) A/D value

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

byte #	Description
byte 0	Data coding.
byte 1-7	Data with format defined by byte 0

Type = 3 (0x03) D/A value

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

byte #	Description
byte 0	Data coding.
byte 1-7	Data with format defined by byte 0

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 #	Description
byte 0	Data coding.
byte 1-7	Data with format defined by byte 0

Type = 5 (0x05) Signal Level

Signal Level is a **relative** strength value that (as default) has its maximum at 100 and minimum at 0 interpreted as percentage. For a digital transmission Signal Level ca be used to give an indication of the analogue signal and Type = 6 (0×06) Signal Quality can be used to give an indication of the quality of the digital part as BER (Bit Error Ratio) for example.

Default coding: percentage. Optional codings: Scale 0-255 (1), Scale 0-65535 (2)

byte #	Description
byte 0	Data coding.
byte 1-7	Data with format defined by byte 0

Type = 6 (0x06) Signal Quality

Signal quality is a **relative** strength value that (as default) has its maximum at 100 and minimum at 0 interpreted as presentage. For a digital transmission Signal Level ca be used to give an indication of the analogue signal and Type = 6 (0×06) Signal Quality be used to give an indication of the quality of the digital part as BER (Bit Error Ratio) for example.

Default coding: percentage. Optional codings: Scale 0-255 (1), Scale 0-65535 (2)

byte #	Description
byte 0	Data coding.
byte 1-7	Data with format defined by byte 0

Back to Specification
Back to Level I Events

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 #	Description
byte 0	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 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
byte 3	MSB of code for button.
byte 4	LSB of code for button.
byte 5	MSB of optional code-page.
byte 6	LSB of optional code-page.

Type = 2 (0x02) Mouse

A mouse movement has occurred.

byte #	Description
byte 0	If zero absolute coordinates follow. If equal to one relative coordinates follow.
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
byte 3	MSB of normalized X-coordinate.
byte 4	LSB of normalized X-coordinate.
byte 5	MSB of normalized Y-coordinate.
byte 6	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 #	Description
byte 0	index.
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

Byte 0 is often used as an index for channels within a module.

Type = 4 (0x04) Off

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

byte #	Description
byte 0	Index.
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 (0x05) Alive

A node tells the world that it is alive.

byte #	Description
byte 0	User specified.
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 (0x06) Terminating

A node tells the world that it is terminating.

byte #	Description
byte 0	User specified.
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 (0x07) Opened

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

byte #	Description
byte 0	User specified.
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 (0x08) Closed

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

byte #	Description
byte 0	User specified.
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 (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" (CLASS1.PROTOCOL, Type=1) in order to provide a method of finding out what is connected to the network.
The data bytes from byte 3 and forward can be used to send a descriptive/user friendly name if desired.
Not mandatory but recommended that all nodes send this event on regular intervals.

byte #	Description
byte 0	User specified.
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 (0x0A) Below limit

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

byte #	Description
byte 0	Reserved.
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 (0x0B) Above limit

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

byte #	Description
byte 0	Reserved.
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 (0x0C) Pulse

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

byte #	Description
byte 0	Reserved.
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 (0x0D) Error

A node indicates that an error occurred.

byte #	Description
byte 0	User specified.
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 (0x0E) Resumed

A node indicates that it has resumed operation.

byte #	Description

byte 0	Reserved.
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 (0x0F) Paused

A node indicates that it has paused.

byte #	Description
byte 0	Reserved.
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 (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 #	Description
byte 0	Reserved.
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 = 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 #	Description
byte 0	Reserved.
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 = 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 #	Description
byte 0	Reserved.
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 = 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 #	Description
byte 0	Reserved.

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) 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 #	Description
byte 0	Reserved.
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) 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 #	Description
byte 0	Reserved.
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) 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 #	Description
byte 0	Reserved.
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) Stop

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

byte #	Description
byte 0	Reserved.
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) Start

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

byte #	Description
byte 0	Reserved.
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) ResetCompleted

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

byte #	Description
byte 0	Reserved.
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) Interrupted

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

byte #	Description
byte 0	Reserved.
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) PreparingToSleep

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

byte #	Description
byte 0	Reserved.
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 = 29 (0x1D) WokenUp

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

byte #	Description
byte 0	Reserved.
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 = 30 (0x1E) Dusk

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

byte #	Description
byte 0	Reserved.
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 = 31 (0x1F) Dawn

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

byte #	Description
byte 0	Reserved.
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 = 32 (0x20) Active

A node indicates that its active.

byte #	Description
byte 0	Reserved.
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 = 33 (0x21) Inactive

A node indicates that its inactive.

byte #	Description
byte 0	Reserved.
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 = 34 (0x22) Busy

A node indicates that its busy.

byte #	Description
byte 0	Reserved.
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 = 35 (0x23) Idle

A node indicates that its idle.

byte #	Description
byte 0	Reserved.
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 = 36 (0x24) Stream Data.

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

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

Type = 37 (0x25) Token Activity

This event is used for cards, RFID's, iButtons, GSM phones and other identification devices. The event is generated when the token device is attached/detached to/from the system. Level II has a counterpart to this event that can take more data. CLASS2.INFORMATION Type=1

Depending on the Token device type a number of this event are sent on the segment with frame index increase for each event. The total expected number can be decuced from the type.

byte #	Description
byte 0	Bit 0,1 - Event code Bit 2-7 - Token device type code
byte 1	Zone
byte 2	Subzone

byte 3	Frame index
byte 4-7	Token data

Event codes

Code	Description
0	Touched and released.
1	Touched.
2	Released.
3	Reserved.

Token device type codes

Code	Description
0	Unknown Token. 128-bits
1	iButton 64-bit token. 64-bits
2	RFID Token. 64-bits
3	RFID Token. 128-bits
4	RFID Token. 256-bits
5-8	Reserved.
9	ID/Credit card. 128-bits
10-15	Reserved.
16	Biometri device. 256-bits
17	Biometri device. 64-bits
18	Bluetooth device. 48-bits
19	GSM IMEI code (International Mobile Equipment Identity) AA-BBBBBB-CCCCCC-D packed in 64 bits. http://en.wikipedia.org/wiki/IMEI [http://en.wikipedia.org/wiki/IMEI]
20	GSM IMSI code (International Mobile Subscriber Identity) packed in 64 bits. http://en.wikipedia.org/wiki/IMSI [http://en.wikipedia.org/wiki/IMSI]
21-63	Reserved.

Type = 38 (0x26) Stream Data with zone.

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

byte #	Description
byte 0	Zone
byte 1	Subzone
byte 2	Sequence number which is increased by one for every new event with stream data.
byte 3-7	Stream data.

Type = 39 (0x27) Confirm.

This event can be used as a general confirm event for zoned and stream data.

byte #	Description
byte 0	Zone
byte 1	Subzone
byte 2	Sequence number
byte 3	Class MSB
byte 4	Class LSB
byte 5	Type MSB
byte 6	Type LSB

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 #	Description
byte 0	Relative or 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 = 41 (0x29) Warning

A node indicates that a warning condition occurred.

byte #	Description
byte 0	User specified.
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 = 42 (0x2A) State

A node indicates that a state change has occured. Th numerical id for the current state and the state that is about to become active is supplied.

byte #	Description
byte 0	User specified.
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
byte 3	Current State
byte 4	New State

Type = 43 (0x2B) Action Trigger

A node optionally indicates that an action has beed triggered by this event.

byte #	Description
byte 0	Action id.
byte 1	Zone for which event applies to (0-255). 255 is all zones
byte 2	

Type = 44 (0x2C) Sunrise

A node indicates that sunrise is detected/calculated.

byte #	Description
byte 0	User specified.
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 = 45 (0x2D) Sunset

A node indicates that sunset is detected/calculated.

byte #	Description

byte 0	User specified.
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 = 46 (0x2E) Start of record

This event is used to mark the start of a multiframe data transfer. This can typically be a GPS received which sends a train of events from one GPS record. The index byte can be used to distinguish record between each other.

byte #	Description
byte 0	Index for record.
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
byte 3	Number of frames to follow or zero for not used

Type = 47 (0x2F) End of record

This event is used to mark the end of a multiframe data transfer. The index byte can be used to distinguish record between each other.

byte #	Description
byte 0	Index for record.
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 = 48 (0x30) Pre-set active

This event is used to tell the system that a pre-set configuration is active. Usually a response from a node after a CLASS1.CONTROL, Type=28 has been received by a node.

byte #	Description
byte 0	0
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
byte 3	Code for pre-set that has been set.

Back to Specification
Back to Level I Events

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 #	Description
byte 0	If equal to zero no mute else mute.
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 (0x02) (All) Lamp(s) on/off

Turn on/off lamps on nodes in zone.

byte #	Description
byte 0	If equal to zero off else on.
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 (0x03) Open

Perform open on all nodes in zone.

byte #	Description
byte 0	Optional byte that have a meaning given by the issuer of the event.
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 (0x04) Close

Perform close on all nodes in zone.

byte #	Description
byte 0	Optional byte that have a meaning given by the issuer of the event.

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 (0x05) TurnOn

Turn On all nodes in a zone.

byte #	Description
byte 0	Optional byte that have a meaning given by the issuer of the event.
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 (0x06) TurnOff

Turn Off all nodes in a zone.

byte #	Description
byte 0	Optional byte that have a meaning given by the issuer of the event.
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 (0x07) Start

Start all nodes in a zone.

byte #	Description
byte 0	Optional byte that have a meaning given by the issuer of the event.
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 (0x08) Stop

Stop all nodes in zone.

byte #	Description
byte 0	Optional byte that have a meaning given by the issuer of the event.
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 (0x09) Reset

Perform Reset on all nodes in zone.

byte #	Description
byte 0	Optional byte that have a meaning given by the issuer of the event.
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 (0x0a) Interrupt

Perform Interrupt on all nodes in zone.

byte #	Description
byte 0	Interrupt level. (0 – 255 , zero is lowest interrupt 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 = 11 (0x0b) Sleep

Perform Sleep on all nodes in zone.

byte #	Description
byte 0	Optional byte that have a meaning given by the issuer of the event.
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 (0x0c) Wakeup

Wakeup all nodes in zone.

byte #	Description
byte 0	Optional byte that have a meaning given by the issuer of the event.
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 (0x0d) Resume

Resume all nodes in zone.

byte #	Description
byte 0	Optional byte that have a meaning given by the issuer of the event.
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 (0x0e) Pause

Pause all nodes in zone.

byte #	Description
byte 0	Optional byte that have a meaning given by the issuer of the event.
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 (0x0f) Activate

Activate all nodes in zone.

byte #	Description
byte 0	Optional byte that have a meaning given by the issuer of the event.
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 (0x10) Deactivate

Deactivate all nodes in zone.

byte #	Description
byte 0	Optional byte that have a meaning given by the issuer of the event.

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 = 20 (0x14) Dim lamp(s)

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

byte #	Description
byte 0	Value (0 – 100) . 0 = off, 100 = full on. 254 dim down one step. 255 dim up one step
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) Change Channel

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

byte #	Description
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 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) Change Level

Change an absolute level.

byte #	Description
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 #	Description
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 #	Description
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 Data)

byte #	Description

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

Use this event for streamed data out from a node. The source is then given by the nickname. If a specific received is needed use Zoned Stream.

Type = 26 (0x1A) Sync

Syncronize events on a segment.

byte #	Description
byte 0	Index for subunits within modules. 255 is all subunits.
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 (0x1C) Zoned Stream

byte #	Description
byte 0	Sequence number which is increase by one for each stream data event sent.
byte 1	Zone which should received stream.
byte 2	SubZone which should received stream.
byte 3-7	Stream data

Type = 28 (0x1D) Set Pre-set

Some nodes may have pre-set configurations to choose from. With this event a pre-set can be set for a zone/subzone.

A npode that receive and act on this event send CLASS1.INFORMATION, Type=48 as a response event.

byte #	Description
byte 0	Code for pre-set to set.
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

Back to Specification
Back to Level I Events

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.

The Play/Pause/Stop etc. events in the CLASS1.CONTROL class can also be used for media control.

Type=1 (0x1) Playback

This is for controlling playback functionality

byte #	Description
byte 0	Function
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

Function Codes

Function	Description
0	Stop
1	Pause
2	Play
3	Forward
4	Rewind
5	Fast Forward
6	Fast Rewind
7	Next Track
30	Previous Track
31	Toggle repeat mode
32	Repeat mode ON
33	Repeat mode OFF
34	Toggle Shuffle mode
35	Shuffle ON
36	Shuffle mode OFF
37	Fade in, Play
38	Fade out, Stop

Appropriate CLASS1.INFORMATION events should be sent from the controled device as response to this event.

Type=2 (0x2) NavigatorKey English

This is typically for navigation functions or DVD controls

byte #	Description

byte 0	Function
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

Function Codes

Function	Description
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)

Type=3 (0x3) Adjust Contrast

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

byte #	Description
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 #	Description
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 #	Description
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 #	Description
byte 0	Reserved.
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 #	Description
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 #	Description
byte 0	Reserved.
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 #	Description
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 #	Description
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 #	Description
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 #	Description
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 #	Description
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 #	Description
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 #	Description
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 #	Description
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 #	Description
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/Playlist

This is typically for selecting an album or playlist for playback

byte #	Description
byte 0	A value between 0 and 127 indicates the album/playlist number. A value between 128 and 159 is change down by the specified number of albums/playlists. 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 #	Description
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 #	Description
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 #	Description
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 #	Description
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 #	Description
byte 0	Device code
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.

Device Code

Code	Description
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
22	Sat1
23	Sat2
24	Sat3
25	mp3 source
25	mpeg source

Type=28 (0x1C) Select Output

This is for controlling the output of a playback device

byte #	Description
byte 0	Output Code
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.

Output Code

Code	Description
0	Auto
16	Component
17	VGA
18	SVideo
19	Video1
20	Video2
21	Video3

Type=29 (0x1D) Record

Control a recording device.

byte #	Description
byte 0	0 - Start to record 1 - Stop record 2 - Disable AGC 3 - Enable AGC
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=30 (0x1E) Set Recording Volume

Control a recording device.

byte #	Description
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=40 (0x28) Tivo Function

This is typically for accessing TIVO functions

byte #	Description
byte 0	TIVO Code
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.

TIVO Code

Code	Description
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

Type=50 (0x32) Get Current Title

Get the title for the current active media.

byte #	Description
byte 0	Reserved.
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=51 (0x33) Set mediaposition in milliseconds

This is for controlling the position in the stream/file of a playback device

byte #	Description
byte 0	Reserved
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.
byte 3-7	Position in milliseconds

Bytes 3-7 can contain an integer with a size specified by the event size. This 0xff, 0xffff, 0xffffff, x0ffffffff and 0xfffffffff is the maximum that can be sent for different sizes.

Type=52 (0x34) Get media information

Get various media information from a device.

byte #	Description
byte 0	Type of media information requested. 1 - Current Title 1 - Get Folders 2 - Get Disks 3 - Get Tracks 4 - Get Albums/Playlists 5 - Get Channels 6 - Get Pages 7 - Get Chapters
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.

If a device does not support the requested type of media information its sends a CLASS1.INFORMATION error event or does not respons.

Type=53 (0x35) Remove Item from Album

Remove an item from an album.

byte #	Description
byte 0	0-128 - Pos to remove from album/playlist 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=54 (0x36) Remove all Items from Album

Remove all items from an album.

byte #	Description
byte 0	Reserved.
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=55 (0x37) Save Album/Playlist

Save album/playlist to permanent storage.

byte #	Description
byte 0	0 - Do not overwrite if it already exists 1 - Overwrite if it exist.
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=60 (0x3c) Multimedia Information/Control

Send multimedia information. This can be the title for the current active media. It can be sent as a response to a "Get Title" or similar event or by its own when a new title is playing or other multimedia information has changed.

byte #	Description
byte 0	0 = Active Title (url). 1 = Set Title(url). 2 = Active Folder(url). 3 = Set Active Folder(url). 4 = Artist(string). 5 = Year(string). 6 = Genre(string).

	7 = Album(string). 8 = Comment(string). 9 = Track(integer). 10 = Picture(url). 11 = Samplerate(integer) 12 = Bitrate(integer) 13 = Channels(integer) 14 = Media size bytes(integer) 15 = Time(string) 16 = Mpeg version(string) 17 = Mpeg layer(string) 18 = Frequency(integer) 19 = Channel Mode 20 = CRC(integer) 21 = Copyright(string) 22 = Original(string) 23 = Emphasis 24 = Mediaposition in milliseconds(integer) 25 = Medialength in milliseconds(integer) 26 = Version(string) 27 = Album/Playlist(string) 28 = Play file(url) 29 = Add file to album/playlist(url) 30 = Current Folder (url) 31 = Folder content(url) 32 = Set Folder(url) 33 = Get Folder content(url) 34 = Get Folder content albums/playlists(url) 35 = Get Folder content filter(string) 36 = Disks list(String) 37 = Folders list(String) 38 = Tracks list(String) 39 = Albums/Playlist list(String) 40 = Channels list(String) 41 = Pages list(String) 42 = Chapters list(String) 43 = New Album/Playlist(url)
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.
byte 3	Index. Base 0. Increase by one for every fragment of the title sent.
byte 4-7	Data.

The last fragment is sent with no data.

Lists in string form have list items seperated with a zero (0×00).

Album can be looked upon as a playlist which is a term used for many other multimedia products.

Back to Specification
Back to Level I Events

Class=100 (0x64) Phone

CLASS1.PHONE

Description

This class is for phone related functionality.

Type = 0 (0x00) Undefined.

General event.

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.

Phone calls are reported in the following form

```
from,to
```

where *from* is the originating number and *to* is the receiving phone. Numbers is preferable presented in an international form. So a call from England to a Swedish phone should take the following form

```
44-123-1122334,46-657-413430
```

which is sent in tree frames. Some device can't seperate country and areacode and therefore the form

```
441231122334,46657413430
```

will also be valid.

a database connected appliation can later resolv this and present

```
A customer,Eurosource
```

This is the type=8 event, database info, (see below). Note that the comma cant be used in the descriptive names.

Calls from unlisted numbers are presented as

,to

byte #	Description
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 #	Description
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 #	Description
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 #	Description
byte 0	An id for the call. This can for instance be a number that increases by one for each call.
byte 1	Zone for answer location.
byte 2	Subzone for answer location.

Type = 5 (0x05) Hangup.

The call has been terminated by the receiving end.

byte #	Description
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 #	Description
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 #	Description
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 #	Description
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.

Back to Specification
Back to Level I Events

Class=102 (0x66) Display

CLASS1.DISPLAY

Description

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

The New York module is an example on the how this can be implemeneted in a module.

Escape sequencys

An escape sequency is preceided with a %. As a result to display ”%” %% should be used.

The first character after the % is the escape-type. This character is case sensitive. That is “e” is not the same as “E”.

Escape character	Description
r	**Display a register content.** The second character tells how the register should be interepreted and it is followed by the register pos. As an example %rn2 can be used to display a nomalized integer that is at register pos=2 and forward. **$** zero terminated string **!** boolean value **b** signed char **B** unsigned char **s** signed short **S** unsigned short **i** Signed int **I** unsigned int **l** signed long **L** unsigned long **f** floating point decimal value **d** Date format **t** Time **n** Normalized integer The intended use is to have actions that store data from events in registers and that they are displayed by the escape.
p	**Parameter data** Display parameter escapes. The format is %p001 where “001” is the id that identifies the parameter. This escape is used for hard parameters displayed by the display maker. Se Type=6 below.
e	**Event data** Event data escapes. The format is %eclass,type,r where class and type tells which event is of interest and r have the same format as the r escape

The above is just a recommendation. Anyone can of course use any format they like.

Type = 0 (0x00) Undefined.

General event.

Type = 1 (0x01) - Clear Display

Clear the display on displays in a certain zone,subzone.

byte #	Description
byte 0	Code - not yet defined
byte 1	Zone
byte 2	Subzone

Type = 2 (0x02) - Position cursor

Move the cursor to a specific position on displays in a certain zone,subzone.

byte #	Description
byte 0	Code - not yet defined
byte 1	Zone
byte 2	Subzone
byte 3	Row to move to (first row is 0).
byte 4	Column to move to (first column is 0).

Type = 3 (0x03) - Write Display

Write to display(s) in a certain zone,subzone. The update of the display is immediate.

byte #	Description
byte 0	index - Increase by one for each event sent for specific text to display.
byte 1	Zone
byte 2	Subzone
byte 3-7	Display data

Index is increased by one for each event that builds up a specific message. If needed an empty (no data) can be sent as the last event else sending data to fill the display buffer will give the end automatically.

Type = 4 (0x04) - Write Display buffer

Write to the buffers of displays in a certain zone,subzone. The update of the display is is not done right away but is instead done when the Show Buffer event is received by the display unit.

byte #	Description
byte 0	index - Increase by one for each event sent for specific text to display.
byte 1	Zone
byte 2	Subzone
byte 3-7	Display data

Index is increased by one for each event that builds up a specific message. If needed an empty (no data) can be sent as the last event else sending data to fill the display buffer will give the end automatically.

Many LCD displays allow definition of special characters. Use this event to define custom matrixes buy defining a subzone for the user defined matrix(es).

Type = 5 (0x05) - Show Display Buffer

Tells displays in a certain zone,subzone to display the content in their display buffers. The update of the display is immediate.

byte #	Description
byte 0	index - Increase by one for each event sent for specific text to display.

byte 1	Zone
byte 2	Subzone

Type = 6 (0x06) - Set Display Buffer Parameter

With this call a display buffer parameter can be sent to a display. This parameter is inserted at the escape position %pn in the string in the buffer *when the buffer is transfered to the display*.

Note that there are no zone and subzone defined for this event and the escapes must instead be choosen to be distinct in a system.

byte #	Description
byte 0	Data coding byte as of VSCP Specification.

Type = 6 (0x06) - Set Display Buffer Parameter

With this call a display buffer parameter can be sent to a display. This parameter is inserted at the escape position %pn in the string in the buffer *when the buffer is transfered to the display*.

Note that there are no zone and subzone defined for this event and the escapes must be choosen to be distinct in a system.

byte #	Description
byte 0	Data coding byte as of VSCP Specification.
byte 1-7	Data

Type = 32 (0x20) - Show Text

This event contains information that should be displayed on displays pointed out by zone/subzone.

This event can have the same functionality as **Write Display** or be set on an higher abstraction level.

byte #	Description
byte 0	Index - Increase by one for each event sent for specific text to display.
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
byte 3-7	Message

Index is increase by one for each event that builds up a specific message. If needed an empty (no data) can be sent as the last event else sending data to fill the display buffer will give the end automatically.

The text sent to a node can contain escape characters that themself display data or other display events. Se the New York node for examples of this.

For a multi line display one can use different subzones o address diffrent lines. One can also us macro characters to map display events to a line.

Type = 48 (0x30) - Show LED

This event contains information that should be displayed on LED(s) pointed out by zone/subzone.

byte #	Description
byte 0	Index
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

byte 3	0 =off, 1=on, 2=blink
byte 4	Blink period: MSB milliseconds for ON.
byte 5	Blink period: LSB milliseconds for ON.
byte 6	Blink period: MSB milliseconds for OFF.
byte 7	Blink period: LSB milliseconds for OFF.

Blink period can be omitted if not used or if blink period is defined hard.

Type = 49 (0x31) - Show LED RGB Color

This event set the color for LED(s) pointed out by zone/subzone.

byte #	Description
byte 0	Index
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
byte 3	Color R to display 0-255
byte 4	Color G to display 0-255
byte 5	Color B to display 0-255

For a two or three color display byte 3 can be used to index the colors.

Back to Specification
Back to Level I Events

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 [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 [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/ [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 Zone Byte3 Sub zone Byte4 Repeat count if any.

Back to Specification
Back to Level I Events

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

CLASS1.ONEWIRE

Description

Carrier for the 1-wire API. Its often better to have a translation layer in between the 1-Wire bus and VSCP but to be able to control 1-Wire nodes fully this class has been defined.

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.

byte #	Description
byte 0-7	1-Wire ID with MSB first

Type = 2 (0x02) Convert.

Command a 1-wire node to do a conversion.

byte #	Description
byte 0-7	1-Wire ID with MSB first

Type = 3 (0x03) Read ROM.

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

byte #	Description
byte 0-7	1-Wire ID with MSB first

Type = 4 (0x04) Match ROM.

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

byte #	Description
byte 0-7	1-Wire ID with MSB first

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 field layed out as a standard 64-bit 1-wire ID.

byte #	Description
byte 0-7	1-Wire ID with MSB first

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.

byte #	Description
byte 0-7	1-Wire ID with MSB first

Type = 8 (0x08) Program.

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

byte #	Description
byte 0-7	1-Wire ID with MSB first

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.

byte #	Description
byte 0-7	1-Wire ID with MSB first

Type = 11 (0x0B) Read Memory.

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

byte #	Description
byte 0-7	1-Wire ID with MSB first

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.

byte #	Description
byte 0-7	1-Wire ID with MSB first

Back to Specification
Back to Level I Events

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

CLASS1.X10

Description

X10 Protocol functionality.

Type = 0 (0x00) Undefined.

General event.

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

byte #	Description
byte 0	Header/Code
byte 1	Address or Function byte

The header has the following format

Bit #	Description
0	0
1	F/A
2	1
3	Dim amount
4	Dim amount
5	Dim amount
6	Dim amount
7	Dim amount

Where
F = Function
A = Address

Note that bit 0 always is zero.

Address

0	Device Code
1	Device Code
2	Device Code
3	Device Code
4	House Code
5	House Code
6	House Code
7	House Code

Function

0	Function Code

1	Function Code
2	Function Code
3	Function Code
4	House Code
5	House Code
6	House Code
7	House Code

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

byte #	Description
byte 0	Header Code (Always 0×03)
byte 1	Function.
byte 2	Unit Code.
byte 3	Data.
byte 4	Command.

Where

Function byte

0	1
1	1
2	1
3	0
4	House Code
5	House Code
6	House Code
7	House Code

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

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

byte #	Description
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 #	Description
byte 0	Node address
byte 1	Header Code (Always 0×03)
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 #	Description
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.

vscp_specification_class_201.txt · Last modified: 2008/04/09 11:15 (external edit)

Back to Specification
Back to Level I Events

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

CLASS1.LON

Description

LON Works functionality.

Type = 0 (0x00) Undefined.

General Event.

Back to Specification
Back to Level I Events

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

CLASS1.EIB

Description

EIB functionality.

Type = 0 (0x00) Undefined.

General Event.

Back to Specification
Back to Level I Events

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/ [http://www.hth.com/snap/]
or in this document http://www.hth.com/filelibrary/pdffiles/snap.pdf [http://www.hth.com/filelibrary/pdffiles/snap.pdf]

Type = 0 (0x00) Undefined.

General Event.

Back to Specification
Back to Level I Events

Class=205 (0xCD) CBUS protocol i/f

CLASS1.CBUS

Description

Control and interface for the CBUS protocol

Type = 0 (0x00) Undefined.

General event.

Back to Specification
Back to Level I Events

Class=206 (0xCE) GPS

CLASS1.GPS

Description

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

- For **Angle** use CLASS1.MEASUREMENT, Type = 30 (0x1E) Angle
- For **Altitude** use CLASS1.MEASUREMENT, Type = 51 (0×33) Altitude.
- For **Speed** use CLASS1.MEASUREMENT, Type = 32 (0×20) Speed
- For **Signal Quality** use CLASS1.DATA, Type = 5 (0×05) Signal Quality
- For **Timestamp** use CLASS1.MEASUREMENT, Type = 4 (0×04) Time

Typically one NMEA, SIRF etc frame with information needs to be translated to many VSCP events. It is possible to group the events together with

- CLASS1.INFORMATION, Type = 46 (0x2E) Start of record
- CLASS1.INFORMATION, Type = 47 (0x2F) End of record

if preferred to hold the evenst togehter.

Type = 0 (0x00) Undefined.

General event.

Type = 1 (0x01) Position.

Position information as decimal Latitude + Longitude.

byte #	Description
byte 0-3	Latitude as floating point value.
byte 4-7	Longitude as floating point value.

Type = 2 (0x02) Satellites.

Number of satellites used.

byte #	Description
byte 0	# of satellites used.

Back to Specification
Back to Level I Events

Class=212 (0xD4) Wireless

CLASS1.WIRELESS

Description

This class of events is used for wireless equipments such as cellular phones etc.

- For **IMEI** code use CLASS1.INFORMATION, Type = 37 (0×25) Token Activity.
- For **IMSI** code use CLASS1.INFORMATION, Type = 37 (0×25) Token Activity .
- For **Signal Quality** use CLASS1.DATA, Type = 5 (0×05) Signal Quality
- For **Timestamp** use CLASS1.MEASUREMENT, Type = 4 (0×04) Time

Type = 0 (0x00) Undefined.

General event.

Type = 1 (0x01) GSM Cell.

Event with ID for the GSM cell. Normally this is a 16-bit value but a 32-bit value is used in VSCP.

byte #	Description
byte 0-3	GSM Cell ID.

Back to Specification
Back to Level I Events

Class=509 (0x1FD) Logging i/f

CLASS1.LOG

Description

Logging functionality.

Type = 0 (0x00) Undefined.

General Log Message.

Type = 1 (0x01) Log Message.

Message for Log. Several frames have to be sent for a message that take up more the five bytes which is the maximum for each frame. In this case the zero based index (byte 2) should be increased for each frame.

byte #	Description
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.

Start logging.

byte #	Description
byte 0	id for log

Type = 3 (0x03) Log Stop.

Stop logging.

byte #	Description
byte 0	id for log

Type = 4 (0x04) Log Level.

Set level for logging.

byte #	Description
byte 0	id for log
byte 0	Log level

Back to Specification
Back to Level I Events

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 event.

vscp_specification_class_510.txt · Last modified: 2008/04/09 11:15 (external edit)

Back to Specification
Back to Level I Events

Class=511 (0x1FF) Local use

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.

vscp_specification_class_511.txt · Last modified: 2008/04/09 11:15 (external edit)

Back to Specification

Level II Events

Class=512-1023 VSCP Level II Level I events - CLASS2.LEVELI

Class=1024 (0x400) VSCP Level II Protocol Functionality - CLASS2.PROTOCOL
Class=1025 (0x401) VSCP Level II Control - CLASS2.CONTROL
Class=1026 (0x402) VSCP Level II Information - CLASS2.INFORMATION
Class=1028 (0x404) VSCP Level II Text to speech - CLASS2.TEXT2SPEECH
Class=1029 (0x405) VSCP Level II Custom - CLASS2.CUSTOM
Class=1030 (0x406) VSCP Level II Display - CLASS2.DISPLAY
Class=65535 (0xFFFF) VSCP Daemon internal events - CLASS2.VSCPD

vscp_specification_level_ii_events.txt · Last modified: 2008/04/09 11:16 (external edit)

Back to Specification
Level II Events

Level II Mirror Level I events

CLASS2.PROTOCOL1

Description

Class 512-1023 are reserved for events 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 CLASS2.PROTOCOLL1 the first 16 bytes of the datafield is the GUID of the node the event is intended for.

This is used for translation in the VSCP daemon for instance where a level II client can send events that are automatically sent to the correct interface and is adressed to the correct device in question. To use this feature send events with the GUID of the i/f where the device is located when addressing is needed. The correct nickname is needed and it should be set in GUID byte 16.

An event with a class >= 512 but < 1024 will be sent to all Level II clients and to the correct i/f (the one that have the addressed GUID). A response form the device will go out as as a Level II event using the GUID of the interface but class will always have a value < 512 for a response event just as all events originating from a device.

Note that the LSB of an interface GUID is always the nickname id for a device interface.

Some Examples

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

To set a new nickname for a node send the following event

```
Class = 512  = Level I Protocol
Type = 6 (0x06) Set Nickname id for node.
Data byte 0-15: GUID for Interface where node is located.
Data byte 16: Old nickname for node.
Data Byte 17: The nickname for the node.
```

Response is

```
Class = 0  = Level I Protocol
Type = 7 (0x07) Nickname id accepted.
No data bytes
```

Note the the LSB of the GUID contains the nickname is in both cases but that this is of no use when the event is sent but should be used to veryfy that the correct node answered when the response is received.

Type = 9 (0x09) Read register.

To read a register of a node send the following event

```
Class = 512  = Level I Protocol
Type = 9 (0x09) Read register.
Data byte 0-15: GUID for Interface where node is located. LSB is nickname for node.
Data byte 16: Nickname for node.
Data Byte 17: Register to read.
```

Response is

```
Class = 0  = Level I Protocol
Type=10 (0x0A) Read/Write response.
Data byte 0: Register read/written.
Data Byte 1: Content of register.
```

vscp_specification_class_512-1023.txt · Last modified: 2008/04/09 11:15 (external edit)

Back to Specification
Level II Events

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 = 0 (0x0000) Undefined.

General event.

Type = 1 (0x0001) ReadRegisterII

Read a Level II register

byte #	Description
byte 0-3	Register to read (or start index).
byte 4-5	Number of registers to read (max 487).
byte 6-7	Reserved.
byte 8-23	Contains the GUID of the target node.

Number of registers to read can also be restricted by the buffer size on the board (register 0×98). If this register is set to somethig else then 0 (default) this is the max size for data.

This means that buffer_size - 8 is maximum data bytes read.

Type = 2 (0x0002) WriteRegisterII

byte #	Description
byte 0-3	Register to write (or start index).
byte 4-7	Reserved.
byte 8-23	GUID of the target node
byte 24…	Data to write to register(s)

Number of registers to write can also be restricted by the buffer size on the board (register 0×98). If this register is set to something else then 0 (default) this is the max size for data.

This means that buffer_size - 24 is maximum data bytes written.

Type = 3 (0x0003) 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 #	Description
byte 0-3	Start index for register read/written.
byte 4-7	Reserved.
byte 8…	Data read/written.

Type = 4 (0x0004) ConfigReadRequest

byte #	Description
byte 0-1	Index of configuration segment being requested (0xFFFF indicates all pages).
byte 2-7	Reserved.
byte 8-23	GUID of the target node.

Type = 5 (0x0005) ConfigReadResponse

byte #	Description
byte 0-1	Index of configuration segment being sent.
byte 2-3	Total number of segments that make up the full configuration.
byte 4-7	Reserved.
byte 8…	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 (0x0006) ConfigChanged

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

byte #	Description
byte 0-7	Reserved.
byte 8-23	GUID of the target node.

Type = 7 (0x0007) ConfigUpdateRequest

Request a copy of a configuration from a node if a GUID is given or from any node willing to answer the request if no GUID given. This event is sent by a node who's configuration had been remotely modified or a node that wants its configuration restored.

byte #	Description
byte 0-7	Reserved.
byte 8-23	GUID of the target node. (can be left out see above)

Type = 8 (0x0008) ConfigUpdateResponse

byte #	Description
byte 0-1	Index of configuration segment being sent.
byte 2-3	Total number of segments that make up the full configuration.
byte 4-7	Reserved.
byte 8-23	GUID of the target node for this configuration.
byte 24…	XML string for the configuration.

Type = 9 (0x0009) ConfigParamRequest

Request specific parameters from a node. The requested parameters are specified on the form

```
<?xml version = "1.0" encoding = "UTF-8" ?>
   <param>
      <name1/>
      <name2/>
      <name3/>
      ...
      <name-n/>
   </param>
```

which requests the parameters name1 . name-n

byte #	Description
byte 0-7	Reserved
byte 8-23	GUID of the target node.
byte 24…	Param request XML data.

Type = 10 (0x000A) ConfigParamResponse

Respnse from a parameer request The requested parameters are received on the form

```
<?xml version = "1.0" encoding = "UTF-8" ?>
   <param>
      <name1>data</name1>
      <name2>data</name2>
      <name3>data</name3>
      ...
      <name-n>data</namen>
   </param>
```

which gives the data for the parameters name1 . name-n

byte #	Description
byte 0-1	Index of configuration segment being sent.
byte 2-3	Total number of segments that make up the full configuration.
byte 4-7	Reserved.
byte 8-23	GUID of the target node.
byte 24…	Param request XML data.

Back to Specification
Level II Events

Level II Control

CLASS2.CONTROL

Description

Level II Control functionality.

Type = 0 (0x0000) Undefined.

General event.

Back to Specification
Level II Events

Level II Information

Class=1026 (0×402) Information

CLASS2.INFORMATION

Description

Level II Information events.

Type = 0 (0x0000) Undefined.

General event.

Type = 1 (0x0001) Token Activity

This event is used for cards, RFID's, iButtons and other identification devices. The event is generated when the token device is attached/detached to/from the system.

byte #	Description
byte 0	Event code
byte 1	Zone
byte 2	Subzone
byte 3	Token device type code
byte 4-7	Reserved
byte 8..	Unique ID of token-device

Event codes

Code	Description
0	Touched and released.
1	Touched.
2	Released.
3-254	reserved.
255	Error.

Token device type codes

Code	Description
0	Unknown Token.
1	iButton 64-bit token.
2	RFID Token.
3	Philips mifare® RFID Token.
4-8	Reserved.
9	ID/Credit card.
10-15	Reserved.
16	Biometri device. 256-bits

17	Biometri device. 64-bits	
18	Bluetooth device. 48-bits	
19		GSM IMEI code (International Mobile Equipment Identity) AA-BBBBBB-CCCCCC-D packed in 64 bits. http://en.wikipedia.org/wiki/IMEI [http://en.wikipedia.org/wiki/IMEI]
20		GSM IMSI code (International Mobile Subscriber Identity) packed in 64 bits. http://en.wikipedia.org/wiki/IMSI [http://en.wikipedia.org/wiki/IMSI]
21-255	Reserved.	

Back to Specification
Level II Events

Level II Mirror Level II Text to speech

CLASS2.TEXT2SPEECH

Description

This is an interface that translates text to speech

Type = 0 (0x0000) Undefined.

General event.

Type = 1 (0x++01) Talk

byte #	Description
byte 0-1	Zone to talk in.
byte 2-3	Subzone to talk in.
byte 4	Relative volume (0-100%).
byte 5-7	Reserved.
byte 9	Text to speak

Back to Specification
Level II Events

Level II Custom

CLASS2.CUSTOM

Description

An implementor can design their own events in this class. Care must be taken not to cause conflicts by selecting types used by other implementors.

Type = 0 (0x0000) Undefined.

General event.

Back to Specification
Level II Events

Level II Display

CLASS2.DISPLAY

Description

Level II specific display functionality. Also look at CLASS1.DISPLAY Class=102 (0x66) Display i/f - CLASS1.DISPLAY

vscp_specification_class_1030.txt · Last modified: 2008/04/09 11:15 (external edit)

Back to Specification
Level II Events

VSCP Daemon intenal events - CLASS2.VSCPD

CLASS2.VSCPD

Description

This class is reserved for internal events used by the decision matrix mechanism of the VSCP daemon. Events of this type should never be visible on a physical bus.

Type = 0 (0x0000) Undefined.

General event.

Type = 1 (0x0001) Loop

Event is generated each loop in the DM or if no events are received every 100 ms (configurable value).

Type = 3 (0x0003) Pause

The machine/daemon is going to a pause state.

Type = 4 (0x0004) Activate

The machine/daemon is going from a pause state.

Type = 5 (0x0005) Second

Event is generated each new second

Type = 6 (0x0006) Minute

Event is generated each new minute

Type = 7 (0x0007) Hour

Event is generated each new Hour

Type = 8 (0x0008) Noon

Event is generated at 12:00 each day

Type = 9 (0x0009) Midnight

Event is generated at 00:00 each day.

Type = 11 (0x000B) Week

Event is generated when a new week starts (config setting).

Type = 12 (0x000C) Month

Event is generated when a new month starts.

Type = 13 (0x000D) Quarter

Event is generated when a new quarter starts.

Type = 14 (0x000E) Year

Event is generated when a new year starts.

Type = 15 (0x000F) random-minute

Event is generated randomly over a minute as is sent once each minute.

Type = 16 (0x0010) random-hour

Event is generated randomly over an hour and is sent once each hour.

Type = 17 (0x0011) random-day

Event is generated randomly over a day and is sent once each day.

Type = 18 (0x0012) random-week

Event is generated randomly over a week and is sent once each week.

Type = 19 (0x0013) random-month

Event is generated randomly over a month and is sent once each month.

Type = 20 (0x0014) random-year

Event is generated randomly over a year and is sent once each year.

Type = 21 (0x0015) Dusk

Event is from calculated dusk.

Type = 22 (0x0016) Dawn

Event is from calculated dawn.

Type = 23 (0x0017) Starting up

The machine/daemon is starting up. This is the first event sent after a machine start up. Shutting down

Type = 24 (0x0018) Starting up

The machine/daemon is shutting down. This is the last event sent before a machine is shutting of

Back to Specification

CRC polynoms used by VSCP

8-bit checksum

The 8-bit checksum used is the DOW checksum.

```
Formula: X8 + X5 + x4 + 1
Polynomial: 0x18
Initial Reminder: 0x00
```

16-bit checksum

The 16-bit checksum used is the CCITT checksum.

```
Formula: X16 + X12 + x5 + 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: X32 + X26 + X23 + X22 + X16 + X12 + X11 + X10 + X8 + X7 + X5 + X4 + X2 + X1 + 1
Polynomial: 0x04c11db7
Initial Reminder: 0xffffffff
Note that result is inverted.
```

http://www.netrino.com/Connecting/2000-01/ [http://www.netrino.com/Connecting/2000-01/] is a good pointer for code. Free code is alos available in the CANAL/VSCP distribution.

Back to Specification

Reversion History

- 2008-05-17 — The recommended bitrate over CAN is changed from 500 kbps to 125 kbps.
  - Class=0, Type = 31 (0x1F) Who is there? Is now mandatory.
  - Class=0, Type = 36 (0×24) Extended Read register. Is now mandatory.
  - Class=0, Type = 37 (0×25) Extended Read/Write response. Is now mandatory.
  - Class=0, Type = 38 (0×26) Extended Write Register. Is now mandatory.
  - Class=0, Type = 39 (0×27) Extended Page Read. Is now mandatory.
  - Class=0, Type = 40 (0×28) Extended Page Write. Is now mandatory.
- 2008-04-20 — New common register added (153/0×99) Number of register pages used.
- 2007-11-04 — Added wireless class + GPS events.
- 2007-10-29 — Sunrise/Sunset events added for Class=20 (0×14) Information
- 2007-05-04 — Changed MDF types to ISO types.
- 2007-05-03 — Added Blutooth type to CLASS1.INFORMATION, Type=37 for Bluetooth proximity.
- 2006-04-24 — Event CLASS1.INFORMATION, Action Trigger (43) has been added.

Old

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. 0×00 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 (0×0000 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 0×00.
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.

2005-09-01 – Spec moved to wiki. Version 1.40

- Added a switch to the DM control for VSCP Level II that can be used to mark the ending row in the current decision matrix matrix. This can save a lot of time for the EDA parsing especially if the matrix is in external EEPROM.

vscp_specification_reversions.txt · Last modified: 2008/05/17 14:44 by 87.249.175.45