

VSCP Specification 1.10.4

Copyright © 2000-2014 Åke Hedman, Grodans Paradis AB, <akhe@grodansparadis.com>

March 31, 2014

Many people has helped to create and evolve this protocol: *Behzad Ardakani, Marcus Rejås, Charles Tewiah, Mark Marooth, Gediminas Simanskis, Henk Hofstra, Stefan Langer, Kurt Herremans, Jiri Kubias, Frank Sautter, Dinesh Guleria, Andreas Merkle* are valued contributors.

If you use VSCP components professionally please consider contributing resources to the project (<http://vscp.org/support.php>). Keeping this project going is a daily struggle and the time it requires get all things in place makes it hard to earn money from other work.

Abstract

Collected in a package described as **VSCP & friends** a complete solution framework for measurement and control, also popular known as m2m (machine to machine) and IoT (Internet of Things), is described in this paper. VSCP (Very Simple Control Protocol) defines the protocol and the framework, “Friends” is the many software tools, examples and api’s the are freely available. The protocol was placed in the public domain in year 2000 when the project first was started and is therefore free to use and implement for everyone commercially or otherwise. VSCP solves the uniform discovery and uniform configurations problems for small devices. It also uses a very well specified event format and supports global unique identifiers for nodes making a node identifiable wherever it is installed in the world. It has a register model to give a flexible common interface to node configuration and a model for controlling node functionality. VSCP does not assume anything about the lower level system. It works with different transport mechanism such as Ethernet, TCP/IP, Wireless, Zigbee, Bluetooth, CAN, GPRS, RS-232, USB. Every control and measurement situation can be described and implemented using VSCP.

Contents

I Specification	27
1 Introduction	27
1.1 Open? What does that mean?	29
1.2 License	29
1.2.1 License options	29
1.2.2 Open License	29
1.2.3 Commercial License	30
1.2.4 Questions and answers	31
1.2.5 Full GNU GPL License	31
1.3 Where is the project located?	42
1.4 When was the project started?	43
1.5 Why another protocol?	43
1.6 Why is the VSCP protocol needed?	43
1.7 How does it work?	45
1.7.1 Events	45
1.7.2 Registers	45
1.7.3 MDF - Module Description File	45
1.7.4 DM - Decision Matrix	46
1.7.5 Measurements	46
1.7.6 Putting it all together	46
1.7.7 The Friends	47
1.7.8 Summary	48
2 Physical Level/Lower level protocols	48
2.1 VSCP over CAN	48
2.1.1 Format of the 29 bit CAN identifier in VSCP	50
2.1.2 RJ-XX pin-out	51
2.1.3 Using alternative bit-rates	51
2.2 VSCP Level I over RS-232	51
2.2.1 General frame format	52
2.2.2 Command frame	53
2.2.3 Error frame	54
2.2.4 Command Reply	55
2.2.5 ACK frame	55
2.2.6 NACK frame	56
2.2.7 Special characters	56
2.2.8 Sending CAN frames with the protocol	57
2.2.9 Command codes	57
2.2.10 Response codes	57
2.2.11 Error codes	57
2.3 VSCP Level I over RS-485/RS-422	57
2.3.1 Addressing	57

2.3.2	Packet format	58
2.4	VSCP over Ethernet (raw Ethernet)	59
2.4.1	Powering Ethernet nodes	61
2.5	VSCP over TCP/IP	61
2.6	VSCP over UDP	62
2.6.1	VSCP TCP Interface	63
2.7	VSCP over IEEE 802.15.4 protocol (MiWi)	64
2.7.1	What is MiWi	64
2.7.2	802.15.4 payload usage	64
3	VSCP Level I Specifics	65
3.1	Level I Node types	65
3.1.1	Dynamic nodes	65
3.1.2	Hard coded nodes	65
3.2	Address or “nickname” assignment for Level I nodes	65
3.2.1	Node segment initialization. Dynamic nodes	65
3.2.2	Node segment initialization. Hard coded nodes.	67
3.2.3	Node segment initialization.. Silent dynamic nodes.	67
3.2.4	Examples	69
4	Level II Specifics	71
4.1	Level II events	71
4.2	VSCP LEVEL II UDP datagram offsets	71
4.3	XML Representation	72
5	Globally Unique Identifiers	72
5.1	Predefined VSCP GUID’s	73
5.2	Shorthand GUID’s	75
6	Register Abstraction Model	75
6.1	Level I - Register Abstraction Model	75
6.1.1	User ID	76
6.1.2	Manufacturer device ID/Manufacturer sub device ID	76
6.1.3	Page select	76
6.1.4	Standard device families and types	76
6.2	Level II - Register Abstraction Model	76
7	Decision Matrix	79
7.1	Event handling and Rules	79
7.2	The model	79
7.2.1	Event	79
7.2.2	Decision	79
7.2.3	Action	80
7.3	Decision matrix for Level I nodes.	80
7.3.1	oaddr	80
7.3.2	flag	80

7.3.3	class-mask / class-filter	81
7.3.4	type-mask / type-filter	81
7.3.5	action	81
7.3.6	action-parameter	82
7.3.7	General	82
7.4	Decision matrix for Level II nodes.	82
7.4.1	Mask (32-bit)	83
7.4.2	Filter (32-bit)	83
7.4.3	Control(32-bit)	84
7.4.4	Action (16-bit)	85
7.4.5	Action Parameters	85
8	Data coding	85
8.1	Definitions for bits in byte 0	85
8.1.1	Bits 5,6,7	86
8.1.2	Bits 3,4	87
8.1.3	Bits 0,1,2	88
9	Module Description File	88
9.1	Real life file examples	88
9.2	XML Format Specification	88
9.3	Creating a new MDF file	97
9.3.1	Relations between registers and abstractions	98
10	VSCP boot loader algorithm	99
10.1	Microchip custom algorithm for PIC18 devices, Boot loader code = 0x01	100
10.1.1	Changes from Appnote 247	101
10.1.2	Loading code to a node.	102
10.1.3	Filter and Mask settings	102
10.1.4	PUT_BASE_INFO	103
10.1.5	PUT_DATA	104
10.1.6	GET_BASE_INFO	105
10.1.7	GET_DATA	105
10.1.8	References	107
11	VSCP Multicast	107
12	Level I Events	107
12.1	Class=0 (0x00) VSCP Protocol Functionality	107
12.1.1	Type = 0 (0x00) Undefined.	108
12.1.2	Type = 1 (0x01) Segment Controller Heartbeat.	108
12.1.3	Type = 2 (0x02) New node on line / Probe.	109
12.1.4	Type = 3 (0x03) Probe ACK.	109
12.1.5	Type = 4 (0x04) Reserved for future use.	109
12.1.6	Type = 5 (0x05) Reserved for future use.	110

12.1.7	Type = 6 (0x06) Set nickname-ID for node.	110
12.1.8	Type = 7 (0x07) nickname-ID accepted.	110
12.1.9	Type = 8 (0x08) Drop nickname-ID / Reset Device.	110
12.1.10	Type = 9 (0x09) Read register.	111
12.1.11	Type=10 (0xA) Read/Write response.	112
12.1.12	Type = 11 (0x0B) Write register.	112
12.1.13	Type = 12 (0x0C) Enter boot loader mode.	113
12.1.14	Type = 13 (0x0D) ACK boot loader mode.	114
12.1.15	Type = 14 (0x0E) NACK boot loader mode.	114
12.1.16	Type = 15 (0x0F) Start block data transfer.	114
12.1.17	Type = 16 (0x10) Block data.	115
12.1.18	Type = 17 (0x11) ACK data block.	116
12.1.19	Type = 18 (0x12) NACK data block.	116
12.1.20	Type = 19 (0x13) Program data block .	117
12.1.21	Type = 20 (0x14) ACK program data block .	117
12.1.22	Type = 21 (0x15) NACK program data block .	118
12.1.23	Type = 22 (0x16) Activate new image .	118
12.1.24	Type = 23 (0x17) GUID drop nickname-ID / reset device.	119
12.1.25	Type = 24 (0x18) Page read .	120
12.1.26	Type = 25 (0x19) Page write .	120
12.1.27	Type = 26 (0x1A) Read/Write page response .	121
12.1.28	Type = 27 (0x1B) High end server probe .	121
12.1.29	Type = 28 (0x1C) High end server response .	122
12.1.30	Type = 29 (0x1D) Increment register .	123
12.1.31	Type = 30 (0x1E) Decrement register .	123
12.1.32	Type = 31 (0x1F) Who is there? .	124
12.1.33	Type = 32 (0x20) Who is there response .	124
12.1.34	Type = 33 (0x21) Get decision matrix info .	124
12.1.35	Type = 34 (0x22) Decision matrix info response .	124
12.1.36	Type = 35 (0x23) Get embedded MDF.	125
12.1.37	Type = 36 (0x24) Embedded MDF response.	125
12.1.38	Type = 37 (0x25) Extended page read register.	126
12.1.39	Type = 38 (0x26) Extended page write register.	126
12.1.40	Type = 39 (0x27) Extended page read/write response .	127
12.1.41	Type = 40 (0x28) Get event interest .	128
12.1.42	Type = 41 (0x29) Get event interest response .	128
12.1.43	Type = 48 (0x30) Activate new image ACK.	129
12.1.44	Type = 49 (0x31) Activate new image NACK.	129
12.1.45	Type = 50 (0x32) Start block data transfer ACK.	129
12.1.46	Type = 51 (0x33) Start block data transfer NACK.	129
12.2	Class=1 (0x01) Alarm .	130
12.2.1	Type = 0 (0x00) Undefined .	130
12.2.2	Type = 1 (0x01) Warning .	130
12.2.3	Type = 2 (0x02) Alarm occurred.	130
12.2.4	Type = 3 (0x03) Alarm sound on/off.	130
12.2.5	Type = 4 (0x04) Alarm light on/off.	131

12.2.6	Type = 5 (0x05) Power on/off	131
12.2.7	Type = 6 (0x06) Emergency Stop	131
12.2.8	Type = 7 (0x07) Emergency Pause	131
12.2.9	Type = 8 (0x08) Emergency Reset	132
12.2.10	Type = 9 (0x09) Emergency Resume	132
12.3	Class=2 (0x02) Security	133
12.3.1	Type = 0 (0x00) undefined	133
12.3.2	Type = 1 (0x01) Motion Detect	133
12.3.3	Type = 2 (0x02) Glass break	133
12.3.4	Type = 3 (0x03) Beam break	133
12.3.5	Type = 4 (0x04) Sensor tamper	134
12.3.6	Type = 5 (0x05) Shock sensor	134
12.3.7	Type = 6 (0x06) Smoke sensor	134
12.3.8	Type = 7 (0x07) Heat sensor	134
12.3.9	Type = 8 (0x08) Panic switch	135
12.3.10	Type = 9 (0x09) Door Contact	135
12.3.11	Type = 10 (0x0A) Window Contact	135
12.3.12	Type = 11 (0x0B) CO Sensor	135
12.3.13	Type = 12 (0x0C) Frost detected	136
12.3.14	Type = 13 (0x0D) Flame detected	136
12.3.15	Type = 14 (0x0E) Oxygen Low	136
12.3.16	Type = 15 (0x0F) Weight detected.	136
12.3.17	Type = 16 (0x10) Water detected.	137
12.3.18	Type = 17 (0x11) Condensation detected.	137
12.3.19	Type = 18 (0x12) Noise (sound) detected.	137
12.3.20	Type = 19 (0x13) Harmful sound levels detected.	137
12.4	Class=10 (0x0A) Measurement	138
12.4.1	Type = 0 (0x00) Undefined	138
12.4.2	Type = 1 (0x01) Count	138
12.4.3	Type = 2 (0x02) Length/Distance	138
12.4.4	Type = 3 (0x03) Mass	138
12.4.5	Type = 4 (0x04) Time	139
12.4.6	Type = 5 (0x05) Electric Current	139
12.4.7	Type = 6 (0x06) Temperature	139
12.4.8	Type = 7 (0x07) Amount of substance	139
12.4.9	Type = 8 (0x08) Luminous Intensity (Intensity of light) .	139
12.4.10	Type = 9 (0x09) Frequency	140
12.4.11	Type = 10 (0x0A) Radioactivity and other random events	140
12.4.12	Type = 11 (0x0B) Force	140
12.4.13	Type = 12 (0x0C) Pressure	140
12.4.14	Type = 13 (0x0D) Energy	140
12.4.15	Type = 14 (0x0E) Power	141
12.4.16	Type = 15 (0x0F) Electrical Charge	141
12.4.17	Type = 16 (0x10) Electrical Potential (Voltage)	141
12.4.18	Type = 17 (0x11) Electrical Capacitance	141
12.4.19	Type = 18 (0x012) Electrical Resistance	141

12.4.20	Type = 19 (0x13) Electrical Conductance	142
12.4.21	Type = 20 (0x14) Magnetic Field Strength	142
12.4.22	Type = 21 (0x15) Magnetic Flux	142
12.4.23	Type = 22 (0x16) Magnetic Flux Density	142
12.4.24	Type = 23 (0x17) Inductance	142
12.4.25	Type = 24 (0x18) Luminous Flux	143
12.4.26	Type = 25 (0x19) Illuminance	143
12.4.27	Type = 26 (0x1A) Radiation dose	143
12.4.28	Type = 27 (0x1B) Catalytic activity	143
12.4.29	Type = 28 (0x1C) Volume	143
12.4.30	Type = 29 (0x1D) Sound intensity	144
12.4.31	Type = 30 (0x1E) Angle	144
12.4.32	Type = 31 (0x1F) Position	144
12.4.33	Type = 32 (0x20) Speed	144
12.4.34	Type = 33 (0x21) Acceleration	144
12.4.35	Type = 34 (0x22) Tension	145
12.4.36	Type = 35 (0x23) Damp/moist (Hygrometer reading) .	145
12.4.37	Type = 36 (0x24) Flow	145
12.4.38	Type = 37 (0x25) Thermal resistance	145
12.4.39	Type = 38 (0x26) Refractive power	145
12.4.40	Type = 39 (0x27) Dynamic viscosity	146
12.4.41	Type = 40 (0x28) Sound impedance	146
12.4.42	Type = 41 (0x29) Sound resistance	146
12.4.43	Type = 42 (0x2A) Electric elastance	146
12.4.44	Type = 43 (0x2B) Luminous energy	146
12.4.45	Type = 44 (0x2C) Luminance	147
12.4.46	Type = 45 (0x2D) Chemical concentration	147
12.4.47	Type = 46 (0x2E) Reserved	147
12.4.48	Type = 47 (0x2F) Dose equivalent	147
12.4.49	Type = 48 (0x30) Reserved	147
12.4.50	Type = 49 (0x31) Dew Point	147
12.4.51	Type = 50 (0x32) Relative Level	148
12.4.52	Type = 51 (0x33) Altitude.	148
12.4.53	Type = 52 (0x34) Area	148
12.4.54	Type = 53 (0x35) Radiant intensity	148
12.4.55	Type = 54 (0x36) Radiance	148
12.4.56	Type = 55 (0x37) Irradiance, Exitance, Radiosity . . .	149
12.4.57	Type = 56 (0x38) Spectral radiance	149
12.4.58	Type = 57 (0x39) Spectral irradiance	149
12.5	Class=15 (0x0f) Data	150
12.5.1	Type = 0 (0x00) Undefined	150
12.5.2	Type = 1 (0x01) I/O – value	150
12.5.3	Type = 2 (0x02) A/D value	150
12.5.4	Type = 3 (0x03) D/A value	150
12.5.5	Type = 4 (0x04) Relative strength	150
12.5.6	Type = 5 (0x05) Signal Level	151

12.5.7	Type = 6 (0x06) Signal Quality	151
12.6	Class=20 (0x14) Information	152
12.6.1	Type = 1 (0x01) Button	152
12.6.2	Type = 2 (0x02) Mouse	152
12.6.3	Type = 3 (0x03) On	152
12.6.4	Type = 4 (0x04) Off	153
12.6.5	Type = 5 (0x05) Alive	153
12.6.6	Type = 6 (0x06) Terminating	153
12.6.7	Type = 7 (0x07) Opened	153
12.6.8	Type = 8 (0x08) Closed	153
12.6.9	Type = 9 (0x09) Node Heartbeat	154
12.6.10	Type = 10 (0x0A) Below limit	154
12.6.11	Type = 11 (0x0B) Above limit	154
12.6.12	Type = 12 (0x0C) Pulse	154
12.6.13	Type = 13 (0x0D) Error	155
12.6.14	Type = 14 (0x0E) Resumed	155
12.6.15	Type = 15 (0x0F) Paused	155
12.6.16	Type = 16 (0x10) Sleeping	155
12.6.17	Type = 17 (0x11) Good morning	155
12.6.18	Type = 18 (0x12) Good day	156
12.6.19	Type = 19 (0x13) Good afternoon	156
12.6.20	Type = 20 (0x14) Good evening	156
12.6.21	Type = 21 (0x15) Good night	156
12.6.22	Type = 22 (0x16) See you soon	156
12.6.23	Type = 23 (0x17) Goodbye	157
12.6.24	Type = 24 (0x18) Stop	157
12.6.25	Type = 25 (0x19) Start	157
12.6.26	Type = 26 (0x1A) ResetCompleted	157
12.6.27	Type = 27 (0x1B) Interrupted	157
12.6.28	Type = 28 (0x1C) PreparingToSleep	158
12.6.29	Type = 29 (0x1D) WokenUp	158
12.6.30	Type = 30 (0x1E) Dusk	158
12.6.31	Type = 31 (0x1F) Dawn	158
12.6.32	Type = 32 (0x20) Active	158
12.6.33	Type = 33 (0x21) Inactive	159
12.6.34	Type = 34 (0x22) Busy	159
12.6.35	Type = 35 (0x23) Idle	159
12.6.36	Type = 36 (0x24) Stream Data.	159
12.6.37	Type = 37 (0x25) Token Activity	159
12.6.38	Type = 38 (0x26) Stream Data with zone.	161
12.6.39	Type = 39 (0x27) Confirm.	161
12.6.40	Type = 40 (0x28) Level Changed.	161
12.6.41	Type = 41 (0x29) Warning	162
12.6.42	Type = 42 (0x2A) State	162
12.6.43	Type = 43 (0x2B) Action Trigger	162
12.6.44	Type = 44 (0x2C) Sunrise	162

12.6.45	Type = 45 (0x2D) Sunset	162
12.6.46	Type = 46 (0x2E) Start of record	163
12.6.47	Type = 47 (0x2F) End of record	163
12.6.48	Type = 48 (0x30) Pre-set active	163
12.6.49	Type = 49 (0x31) Detect	163
12.6.50	Type = 50 (0x32) Overflow	164
12.7	Class=30 (0x1E) Control	165
12.7.1	Type = 0 (0x00) Undefined	165
12.7.2	Type = 1 (0x01) Mute on/off	165
12.7.3	Type = 2 (0x02) (All) Lamp(s) on/off	165
12.7.4	Type = 3 (0x03) Open	165
12.7.5	Type = 4 (0x04) Close	166
12.7.6	Type = 5 (0x05) TurnOn	166
12.7.7	Type = 6 (0x06) TurnOff	166
12.7.8	Type = 7 (0x07) Start	166
12.7.9	Type = 8 (0x08) Stop	166
12.7.10	Type = 9 (0x09) Reset	167
12.7.11	Type = 10 (0x0A) Interrupt	167
12.7.12	Type = 11 (0x0B) Sleep	167
12.7.13	Type = 12 (0x0C) Wakeup	167
12.7.14	Type = 13 (0x0D) Resume	167
12.7.15	Type = 14 (0x0E) Pause	168
12.7.16	Type = 15 (0x0F) Activate	168
12.7.17	Type = 16 (0x10) Deactivate	168
12.7.18	Type = 17 (0x11) Reserved for future use	168
12.7.19	Type = 18 (0x12) Reserved for future use	168
12.7.20	Type = 19 (0x13) Reserved for future use	168
12.7.21	Type = 20 (0x14) Dim lamp(s)	168
12.7.22	Type = 21 (0x15) Change Channel	169
12.7.23	Type = 22 (0x16) Change Level	169
12.7.24	Type = 23 (0x17) Relative Change Level	169
12.7.25	Type = 24 (0x18) Measurement Request	169
12.7.26	Type = 25 (0x19) Stream Data	169
12.7.27	Type = 26 (0x1A) Sync	170
12.7.28	Type = 27 (0x1B) Zoned Stream Data	170
12.7.29	Type = 28 (0x1C) Set Pre-set	170
12.7.30	Type = 29 (0x1D) Toggle state	170
12.7.31	Type = 30 (0x1E) Timed pulse on.	171
12.7.32	Type = 31 (0x1F) Timed pulse off.	171
12.7.33	Type = 32 (0x20) Set country/language.	172
12.8	Class=40 (0x28) Multimedia	173
12.8.1	Type=1 (0x1) Playback	173
12.8.2	Type=2 (0x2) NavigatorKey English	174
12.8.3	Type=3 (0x3) Adjust Contrast	174
12.8.4	Type=4 (0x4) Adjust Focus	175
12.8.5	Type=5 (0x5) Adjust Tint	175

12.8.6	Type=6 (0x6) Adjust Color Balance	175
12.8.7	Type=7 (0x7) Adjust Brightness	175
12.8.8	Type=8 (0x8) Adjust Hue	176
12.8.9	Type=9 (0x9) Adjust Bass	176
12.8.10	Type=10 (0xA) Adjust Treble	176
12.8.11	Type=11 (0xB) Adjust Master Volume	176
12.8.12	Type=12 (0xC) Adjust Front Volume	177
12.8.13	Type=13 (0xD) Adjust Center Volume	177
12.8.14	Type=14 (0xE) Adjust Rear Volume	177
12.8.15	Type=15 (0xF) Adjust Side Volume	178
12.8.16	Type=16 to 19 Reserved	178
12.8.17	Type=20 (0x14) Select Disk	178
12.8.18	Type=21 (0x15) Select Track	178
12.8.19	Type=22 (0x16) Select Album/Play list	179
12.8.20	Type=23 (0x17) Select Channel	179
12.8.21	Type=24 (0x18) Select Page	179
12.8.22	Type=25 (0x19) Select Chapter	179
12.8.23	Type=26 (0x1A) Select Screen Format	180
12.8.24	Type=27 (0x1B) Select Input Source	180
12.8.25	Type=28 (0x1C) Select Output	181
12.8.26	Type=29 (0x1D) Record	182
12.8.27	Type=30 (0x1E) Set Recording Volume	182
12.8.28	Type=40 (0x28) Tivo Function	182
12.8.29	Type=50 (0x32) Get Current Title	183
12.8.30	Type=51 (0x33) Set media position in milliseconds	183
12.8.31	Type=52 (0x34) Get media information	183
12.8.32	Type=53 (0x35) Remove Item from Album	184
12.8.33	Type=54 (0x36) Remove all Items from Album	184
12.8.34	Type=55 (0x37) Save Album/Play list	184
12.8.35	Type=60 (0x3C) Multimedia Control	184
12.8.36	Type=61 (0x3D) Multimedia Control reasons	185
12.9	Class=50 (0x32) Alert On LAN	186
12.9.1	Type = 0 (0x00)	186
12.9.2	Type = 1 (0x01) System unplugged from power source	186
12.9.3	Type = 2 (0x02) System unplugged from network	186
12.9.4	Type = 3 (0x03) Chassis intrusion	186
12.9.5	Type = 4 (0x04) Processor removal	186
12.9.6	Type = 5 (0x05) System environmental errors	187
12.9.7	Type = 6 (0x06) High temperature	187
12.9.8	Type = 7 (0x07) Fan speed problem	187
12.9.9	Type = 8 (0x08) Voltage fluctuations	187
12.9.10	Type = 9 (0x09) Operating system errors	187
12.9.11	Type = 10 (0x0A) System power-on errors	188
12.9.12	Type = 11 (0x0B) System is hung	188
12.9.13	Type = 12 (0x0C) Component failure	188

12.9.14 Type = 13 (0x0D) Remote system reboot upon report of a critical failure	188
12.9.15 Type = 15 (0x0F) Update BIOS image	189
12.9.16 Type = 16 (0x10) Update Perform other diagnostic pro- cedures	189
12.10 Class=60 (0x3C) Double precision floating point measurement .	190
12.11 Class=65 (0x41) Measurement with zone	190
12.12 Class=70 (0x46) Single precision floating point measurement .	191
12.13 Class=85 (0x55) Set value with zone	191
12.14 Class=100 (0x64) Phone	192
12.14.1 Type = 0 (0x00) Undefined.	192
12.14.2 Type = 1 (0x00) Incoming call.	192
12.14.3 Type = 2 (0x02) Outgoing call.	193
12.14.4 Type = 3 (0x03) Ring.	193
12.14.5 Type = 4 (0x04) Answer.	193
12.14.6 Type = 5 (0x05) Hangup.	193
12.14.7 Type = 6 (0x06) Giveup.	193
12.14.8 Type = 7 (0x07) Transfer.	194
12.14.9 Type = 8 (0x08) Database Info.	194
12.15 Class=102 (0x66) Display	195
12.15.1 Type = 0 (0x00) Undefined.	196
12.15.2 Type = 1 (0x01) - Clear Display	196
12.15.3 Type = 2 (0x02) - Position cursor	196
12.15.4 Type = 3 (0x03) - Write Display	196
12.15.5 Type = 4 (0x04) - Write Display buffer	197
12.15.6 Type = 5 (0x05) - Show Display Buffer	197
12.15.7 Type = 6 (0x06) - Set Display Buffer Parameter	197
12.15.8 Type = 32 (0x20) - Show Text	198
12.15.9 Type = 48 (0x30) - Set LED	198
12.15.10 Type = 49 (0x31) - Set RGB Color	199
12.16 Class=110 (0x6E) IR Remote I/f	200
12.16.1 Type = 0 (0x00)	200
12.16.2 Type = 1 (0x01) RC5 Send/Receive.	200
12.16.3 Type = 3 (0x02) SONY 12-bit Send/Receive.	200
12.16.4 Type = 32 (0x20) LIRC (Linux Infrared Remote Control). .	200
12.16.5 Type = 48 (0x30) VSCP Abstract Remote Format.	201
12.16.6 Type = 49 (0x31) MAPito Remote Format.	201
12.17 Class=200 (0xC8) 1-Wire protocol i/f	202
12.17.1 Type = 0 (0x00) Undefined.	202
12.17.2 Type = 1 (0x01) New ID	202
12.17.3 Type = 2 (0x02) Convert.	202
12.17.4 Type = 3 (0x03) Read ROM.	202
12.17.5 Type = 4 (0x04) Match ROM.	202
12.17.6 Type = 5 (0x05) Skip ROM.	202
12.17.7 Type = 6 (0x06) Search ROM.	202
12.17.8 Type = 7 (0x07) Conditional Search ROM.	203

12.17.9 Type = 8 (0x08) Program.	203
12.17.10 Type = 9 (0x09) Overdrive skip ROM.	203
12.17.11 Type = 10 (0x0A) Overdrive Match ROM.	203
12.17.12 Type = 11 (0x0B) Read Memory.	203
12.17.13 Type = 12 (0x0C) Write Memory.	203
12.18 Class=201 (0xC9) X10 protocol i/f	204
12.18.1 Type = 0 (0x00) Undefined.	204
12.18.2 Type = 1 (0x01) X10 Standard Message Receive.	204
12.18.3 Type = 2 (0x02) X10 Extended message Receive.	205
12.18.4 Type = 3 (0x03) X10 Standard Message Send.	205
12.18.5 Type = 4 (0x04) X10 Extended Message Send.	206
12.18.6 Type = 5 (0x05) Simple x10 message	206
12.19 Class=202 (0xCA) LON Works protocol i/f	207
12.19.1 Type = 0 (0x00) Undefined.	207
12.20 Class=203 (0xCB) KNX/EIB protocol i/f	208
12.20.1 Type = 0 (0x00) Undefined.	208
12.21 Class=204 (0xCC) S.N.A.P. protocol i/f	209
12.21.1 Type = 0 (0x00) Undefined.	209
12.22 Class=205 (0xCD) CBUS protocol i/f	210
12.22.1 Type = 0 (0x00) Undefined.	210
12.23 Class=206 (0xCE) Position (GPS)	211
12.23.1 Type = 0 (0x00) Undefined.	211
12.23.2 Type = 1 (0x01) Position.	211
12.23.3 Type = 2 (0x02) Satellites.	211
12.24 Class=212 (0xD4) Wireless	212
12.24.1 Type = 0 (0x00) Undefined.	212
12.24.2 Type = 1 (0x01) GSM Cell.	212
12.25 Class=509 (0x1FD) Logging i/f	213
12.25.1 Type = 0 (0x00) Undefined.	213
12.25.2 Type = 1 (0x01) Log event.	213
12.25.3 Type = 2(0x01) Log Start.	213
12.25.4 Type = 3 (0x03) Log Stop.	213
12.25.5 Type = 4 (0x04) Log Level.	213
12.26 Class=510 (0x1FE) Laboratory use	214
12.26.1 Type = 0 (0x00) Undefined.	214
12.27 Class=511 (0x1FF) Local use	215
13 Level II Events	216
13.1 Class=512 (0x200) ... 1023 (0x3FF) - Level II Mirror Level I events	216
13.2 Class=1024 (0x400) - Level II Protocol Functionality	218
13.2.1 Type = 0 (0x0000) Undefined.	218
13.2.2 Type = 1 (0x0001) ReadRegister	218
13.2.3 Type = 2 (0x0002) WriteRegister	218
13.2.4 Type = 3 (0x0003) ReadWriteResponse	219
13.2.5 Class=1025 (0x401) Level II Control	219

13.2.6	Type = 0 (0x0000) Undefined.	219
13.3	Class=1026 (0x402) Level II Information	220
13.3.1	Type = 0 (0x0000) Undefined.	220
13.3.2	Type = 1 (0x0001) Token Activity	220
13.4	Class=1027 (0x404) Level II Mirror Level II Text to speech	222
13.4.1	Type = 0 (0x0000) Undefined.	222
13.4.2	Type = 1 (0x0001) Talk	222
13.5	Class=1029 (0x405) Level II Custom	223
13.5.1	Type = 0 (0x0000) Undefined.	223
13.6	Class=1030 (0x406) Level II Display	224
13.7	Class=1040 (0x410) Measurement string	225
13.8	Class=65535 (0xFFFF) VSCP Daemon internal events	226
13.8.1	Type = 0 (0x0000) Undefined.	226
13.8.2	Type = 1 (0x0001) Loop	226
13.8.3	Type = 3 (0x0003) Pause	226
13.8.4	Type = 4 (0x0004) Activate	226
13.8.5	Type = 5 (0x0005) Second	226
13.8.6	Type = 6 (0x0006) Minute	226
13.8.7	Type = 7 (0x0007) Hour	226
13.8.8	Type = 8 (0x0008) Noon	226
13.8.9	Type = 9 (0x0009) Midnight	226
13.8.10	Type = 11 (0x000B) Week	227
13.8.11	Type = 12 (0x000C) Month	227
13.8.12	Type = 13 (0x000D) Quarter	227
13.8.13	Type = 14 (0x000E) Year	227
13.8.14	Type = 15 (0x000F) random-minute	227
13.8.15	Type = 16 (0x0010) random-hour	227
13.8.16	Type = 17 (0x0011) random-day	227
13.8.17	Type = 18 (0x0012) random-week	227
13.8.18	Type = 19 (0x0013) random-month	227
13.8.19	Type = 20 (0x0014) random-year	227
13.8.20	Type = 21 (0x0015) Dusk	227
13.8.21	Type = 22 (0x0016) Dawn	228
13.8.22	Type = 23 (0x0017) Starting up	228
13.8.23	Type = 24 (0x0018) Shutting down	228
13.8.24	Type = 25 (0x0019) Timer started	228
13.8.25	Type = 26 (0x001A) Timer paused	228
13.8.26	Type = 27 (0x001B) Timer resumed	229
13.8.27	Type = 28 (0x001C) Timer stopped	229
13.8.28	Type = 29 (0x001D) Timer Elapsed	230

II Software 230

14 The VSCP Daemon	230
14.1 Setting up the system	231
14.1.1 VSCPD for Windows - run as service	231
14.2 Windows setup & configuration	238
14.3 Linux setup & configuration	239
14.4 Configuration file walk through	240
14.4.1 General settings	240
14.4.2 Remote user settings	242
14.4.3 automation	243
14.4.4 VSCP level I driver (formerly known as CANAL driver) .	244
14.4.5 VSCP level II driver (former known as VSCP driver) .	245
14.4.6 interface	246
14.5 VSCP Daemon Control Interface	246
14.5.1 GUID assigned to the interface	246
14.5.2 Server discovery	247
14.5.3 Secure the TCP link	248
14.5.4 Username/password pairs for TCP/IP drivers	248
14.6 VSCP TCP/IP Protocol Description	249
14.6.1 Port	249
14.6.2 Command and response format	249
14.6.3 Link Commands	249
14.6.4 Available commands	250
14.6.5 NOOP - No operation.	250
14.6.6 QUIT - Close the connection.	250
14.6.7 USER - Username for login.	251
14.6.8 PASS - Password for login.	251
14.6.9 RESTART	251
14.6.10 SHUTDOWN	251
14.6.11 DRIVER	251
14.6.12 FILE	252
14.6.13 UDP	253
14.6.14 REMOTE	253
14.6.15 INTERFACE	254
14.6.16 DM	255
14.6.17 VARIABLE	257
14.6.18 SEND - Send an event.	263
14.6.19 RETR - Retrieve one or several event(s).	264
14.6.20 RCVLOOP - Send events to client as soon as they arrive.	265
14.6.21 CDTA - Check if there are events to retrieve.	265
14.6.22 CLRA - Clear all events in in-queue	265
14.6.23 STAT - Get statistics information.	265
14.6.24 INFO - Get status information.	266
14.6.25 CHID - Get channel ID.	266
14.6.26 SGID - Set GUID for channel.	266
14.6.27 Ggid - Get GUID for channel.	266
14.6.28 VERS - Get VSCP daemon version.	266

14.6.29	SFLT - Set incoming event filter.	266
14.6.30	SMSK - Set incoming event mask.	267
14.7	Decision Matrix	268
14.7.1	Level I	268
14.7.2	Level II	268
14.7.3	Row format	268
14.7.4	The decision matrix file format	270
14.7.5	Scheduler	271
14.7.6	Actions	272
14.7.7	Internal DM Events	281
14.7.8	Timing parameter data format	282
14.7.9	External standard functionality	283
14.7.10	String substitution keywords	283
14.7.11	Examples	283
14.8	Drivers	284
14.8.1	Level I (CANAL) drivers	284
14.8.2	Level II drivers	284
14.9	HTML5 websocket interface	285
14.9.1	Subprotocols	285
14.9.2	Packets and there formats	289
14.9.3	Packet prefixes	289
14.9.4	Commands	289
14.9.5	Send events	291
14.9.6	Error messages	291
15	VSCP Works	292
15.1	Introduction	292
15.2	The VSCP Client Window	292
15.2.1	Receive event list	296
15.2.2	Transmission object list	297
15.3	The Node configuration Window	299
15.4	The Bootloader Window	301
15.5	The Simulated Node Window	301
15.6	Shortcuts	301
15.7	FAQ	301
15.8	Technical Information	301
15.8.1	===== VSCP Works configuration file format =====	301
15.8.2	VSCP Works receive/transmission data file format	303
15.8.3	VSCP Works Transmission set file format	304
16	vscpcmd	304
16.1	Command line switches	304
16.1.1	-e/-event	304
16.1.2	-m/-measurement	305
16.1.3	-y/-value	305
16.1.4	-q/-host	305

16.1.5 -u/-user	305
16.1.6 -p/-password	305
16.1.7 -n/-count	305
16.1.8 -z/-zone	305
16.1.9 -s/-subzone	306
16.1.10 -t/-test	306
16.1.11 -v/-verbose	306
16.1.12 -h/-help	306
16.1.13 Examples	306
III HTML5 user interface components & tools	307
17 HTML5 websocket widgets	307
17.1 vscpws_stateButton	307
17.1.1 Parameters	312
17.1.2 Methods	326
17.1.3 Using the vscpws_stateButton widget	330
17.2 vscpws_simpleText	330
17.2.1 Function arguments	330
17.2.2 Methods	332
17.2.3 Using the widget	333
17.3 vscpws_thermometerCelsius	341
17.3.1 Function arguments	341
17.3.2 Methods	346
17.3.3 Using the widget	347
17.4 vscpws_speedometerCelsius	348
17.4.1 Function arguments	348
17.4.2 Methods	351
17.4.3 Using the widget	352
IV Source code	353
18 How the source tree is organized.	353
18.1 The software repository tree	353
18.2 The firmware repository tree	353
V Software api	354
19 VSCP Level II driver interface.	354
19.1 long open(char *username, char *password, char *prefix, char *parameters)	354
19.2 long close(void)	355
19.3 long change (char *variable, char *variablevalue)	355

20 vscphelper.dll/dl(so)	355
20.1 Description of the exported API	355
20.1.1 void vscp_setInterfaceTcp(const char *pHost, const short port, const char *pUsername, const char *pPassword) . . .	355
20.1.2 void vscp_setInterfaceDll(const char *pName, const char *pPath, const char *pParameters, unsigned long flags, unsigned long filter, unsigned long mask)	355
20.1.3 long vscp_doCmdOpen(const char *pInterface, unsigned long flags)	356
20.1.4 int vscp_doCmdClose(void)	356
20.1.5 int vscp_doCmdNoop(void)	356
20.1.6 unsigned long vscp_doCmdGetLevel(void)	356
20.1.7 int vscp_doCmdSendCanal(canalMsg *pMsg)	357
20.1.8 int vscp_doCmdSendEvent(const vscpEvent *pEvent)	357
20.1.9 int vscp_doCmdSendEventEx(const vscpEventEx *pEvent)	357
20.1.10 int vscp_doCmdReceiveCanal(canalMsg *pMsg)	357
20.1.11 int vscp_doCmdReceiveEvent(vscpEvent *pEvent)	357
20.1.12 int vscp_doCmdReceiveEventEx(vscpEventEx *pEvent)	357
20.1.13 int WINAPI EXPORT vscp_doCmdDataAvailable(void)	358
20.1.14 int vscp_doCmdStatistics(canalStatistics *pStatistics)	358
20.1.15 int vscp_doCmdFilter(unsigned long filter)	358
20.1.16 int vscp_doCmdMask(unsigned long mask)	358
20.1.17 int vscp_doCmdVscpFilter(const vscpEventFilter *pFilter)	358
20.1.18 int vscp_doCmdBaudrate(unsigned long baudrate)	358
20.1.19 unsigned long vscp_doCmdVersion(void)	359
20.1.20 unsigned long vscp_doCmdDLLVersion(void)	359
20.1.21 const char * vscp_doCmdVendorString(void)	359
20.1.22 const char * vscp_doCmdGetDriverInfo(void)	359
20.1.23 int vscp_getDeviceType(void)	359
20.1.24 bool vscp_isOpen(void)	359
20.1.25 int vscp_doCmdShutDown(void)	359
20.1.26 VscpTcpIf * vscp_getTcpIpInterface(void)	359
20.1.27 unsigned long vscp_readStringValue(const char * pStringValue)	360
20.1.28 unsigned char vscp_getVscpPriority(const vscpEvent *pEvent)	360
20.1.29 void vscp_setVscpPriority(vscpEvent *pEvent, unsigned char priority)	360
20.1.30 unsigned char vscp_getVSCPheadFromCANid(const unsigned long id)	360
20.1.31 unsigned short vscp_getVSCPclassFromCANid(const unsigned long id)	360
20.1.32 unsigned short vscp_getVSCPtypeFromCANid(const unsigned long id)	360

20.1.33	unsigned short vscp_getVSCPnicknameFromCANid(const unsigned long id)	361
20.1.34	unsigned long vscp_getCANidFromVSCPdata(const un- signed char priority, const unsigned short vscp_class, const unsigned short vscp_type)	361
20.1.35	unsigned long vscp_getCANidFromVSCPevent(const vscpEvent *pEvent)	361
20.1.36	short vscp_calcCRC(vscpEvent *pEvent, short bSet) . .	361
20.1.37	bool vscp_getGuidFromString(vscpEvent *pEvent, const char * pGUID)	361
20.1.38	bool vscp_getGuidFromArray(uint8_t *pGUID, const char * pStr)	362
20.1.39	bool vscp_writeGuidToString(const vscpEvent *pEvent, char * pStr)	362
20.1.40	bool vscp_writeGuidToString4Rows(const vscpEvent *pEvent, wxString& strGUID)	362
20.1.41	bool vscp_writeGuidArrayToString(const unsigned char * pGUID, wxString& strGUID)	362
20.1.42	bool vscp_isGUIDEmpty(unsigned char *pGUID) . . .	362
20.1.43	bool vscp_isSameGUID(const unsigned char *pGUID1, const unsigned char *pGUID2)	362
20.1.44	bool vscp_convertVSCPtoEx(vscpEventEx *pEventEx, const vscpEvent *pEvent)	363
20.1.45	bool vscp_convertVSCPfromEx(vscpEvent *pEvent, const vscpEventEx *pEventEx)	363
20.1.46	void vscp_deleteVSCPevent(vscpEvent *pEvent) . . .	363
20.1.47	void vscp_deleteVSCPeventEx(vscpEventEx *pEventEx)	363
20.1.48	void vscp_clearVSCPFilter(vscpEventFilter *pFilter) .	363
20.1.49	bool vscp_readFilterFromString(vscpEventFilter *pFil- ter, wxString& strFilter)	363
20.1.50	bool readMaskFromString(vscpEventFilter *pFilter, wxString& strMask)	364
20.1.51	bool vscp_doLevel2Filter(const vscpEvent *pEvent, const vscpEventFilter *pFilter)	364
20.1.52	bool vscp_convertCanalToEvent(vscpEvent *pvscpEvent, const canalMsg *pcanalMsg, unsigned char *pGUID, bool bCAN)	364
20.1.53	bool vscp_convertEventToCanal(canalMsg *pcanalMsg, const vscpEvent *pvscpEvent)	364
20.1.54	bool vscp_convertEventExToCanal(canalMsg *pcanalMsg, const vscpEventEx *pvscpEventEx)	364
20.1.55	unsigned long vscp_getTimeStamp(void)	364
20.2	Variable handling	364
20.2.1	bool vscp_copyVSCPEvent(vscpEvent *pEventTo, const vscpEvent *pEventFrom)	365

20.2.2	bool vscp_writeVscpDataToString(const vscpEvent *pEvent, wxString& str, bool bUseHtmlBreak)	365
20.2.3	bool vscp_getVscpDataFromString(vscpEvent *pEvent, const wxString& str)	365
20.2.4	bool vscp_writeVscpEventToString(vscpEvent *pEvent, char *p)	365
20.2.5	bool vscp_getVscpEventFromString(vscpEvent *pEvent, const char *p)	365
20.2.6	bool vscp_getVariableString(const char *pName, char *pValue)	365
20.2.7	bool vscp_setVariableString(const char *pName, char *pValue)	366
20.2.8	bool vscp_getVariableBool(const char *pName, bool *bValue)	366
20.2.9	bool vscp_setVariableBool(const char *pName, bool bValue)	366
20.2.10	bool vscp_getVariableInt(const char *pName, int *value)	366
20.2.11	bool vscp_setVariableInt(const char *pName, int value)	366
20.2.12	bool vscp_getVariableLong(const char *pName, long *value)	367
20.2.13	bool vscp_setVariableLong(const char *pName, long value)	367
20.2.14	bool vscp_getVariableDouble(const char *pName, dou- ble *value)	367
20.2.15	bool vscp_setVariableDouble(const char *pName, double value)	367
20.2.16	bool vscp_getVariableMeasurement(const char *pName, char *pValue)	367
20.2.17	bool vscp_setVariableMeasurement(const char *pName, char *pValue)	368
20.2.18	bool vscp_getVariableEvent(const char *pName, vscpEvent *pEvent)	368
20.2.19	bool vscp_setVariableEvent(const char *pName, vscpEvent *pEvent)	368
20.2.20	bool vscp_getVariableEventEx(const char *pName, vscpEven- tEx *pEvent)	368
20.2.21	bool vscp_setVariableEventEx(const char *pName, vscpEven- tEx *pEvent)	368
20.2.22	bool vscp_getVariableGUID(const char *pName, cguid& GUID)	369
20.2.23	bool vscp_setVariableGUID(const char *pName, cguid& GUID)	369
20.2.24	bool vscp_getVariableVSCPdata(const char *pName, uint16 _t *psizeData, uint8 _t *pData)	369

20.2.25	bool vscp_setVariableVSCPdata(const char *pName, uint16_t sizeData, uint8_t *pData)	369
20.2.26	bool vscp_getVariableVSCPclass(const char *pName, uint16_t *vscp_class)	370
20.2.27	bool vscp_setVariableVSCPclass(const char *pName, uint16_t vscp_class)	370
20.2.28	bool vscp_getVariableVSCPtype(const char *pName, uint8_t *vscp_type)	370
20.2.29	bool vscp_setVariableVSCPtype(const char *pName, uint8_t vscp_type)	370
20.3	Examples	370
21	Canalsuperwrapper	371
22	dllwrapper	381
23	vscptcpipif	386
VI	Firmware	402
24	How to port VSCP to new platform	402
25	Firmware common code documentation	405
25.1	vscp_firmware.h	405
25.2	vscp_firmware.c	414
VII	CANAL - CAN Abstraction Layer	428
26	Introduction to CANAL? What is it?	428
27	CANAL-API Specification	429
27.1	long CanalOpen(const char *pConfigStr, unsigned long flags)	429
27.1.1	Params	429
27.1.2	Returns	429
27.2	int CanalClose(long handle)	429
27.2.1	Params	429
27.3	unsigned long CanalGetLevel(long handle)	429
27.3.1	Params	429
27.4	int CanalSend(long handle, const PCANALMSG pCanMsg)	430
27.4.1	Params	430
27.4.2	Returns	430
27.5	int CanalBlockingSend(long handle, const PCANALMSG pCanMsg, unsigned long timeout)	430
27.5.1	Params	430
27.5.2	Returns	430

27.6	int CanalReceive(long handle, PCANALMSG pCanMsg)	431
27.6.1	Params	431
27.6.2	Returns	431
27.7	int CanalBlockingReceive(long handle, PCANALMSG pCan- Msg, unsigned long timeout)	431
27.7.1	Params	431
27.7.2	Returns	431
27.8	int CanalDataAvailable(long handle)	431
27.8.1	Params	431
27.8.2	Returns	431
27.9	int CanalGetStatus(long handle, PCANALSTATUS pCanStatus)	432
27.9.1	Params	432
27.9.2	Returns	432
27.10	int CanalGetStatistics (long handle, PCANALSTATISTICS pCanal- Statistics)	432
27.10.1	Params	432
27.10.2	Returns	432
27.11	int CanalSetFilter (long handle, unsigned long filter)	432
27.11.1	Params	432
27.11.2	Returns	433
27.12	int CanalSetMask (long handle, unsigned long mask)	433
27.12.1	Params	433
27.12.2	Returns	433
27.13	int CanalSetBaudrate (long handle, unsigned long baudrate) . .	433
27.13.1	Params	433
27.13.2	Returns	433
27.14	unsigned long CanalGetVersion (void)	433
27.14.1	Returns	434
27.15	unsigned long CanalGetDllVersion (void)	434
27.15.1	Returns	434
27.16	const char * CanalGetVendorString (void)	434
27.16.1	Returns	434
27.17	const char * CanalGetDriverInfo(void)	434
27.17.1	Returns	436
27.17.2	Params	436
27.17.3	Returns	436
27.18	CANAL - Data Structures	436
27.18.1	CANMSG	436
27.18.2	CANALSTATUS	437
27.18.3	PCANALSTATISTICS	437
27.18.4	Error codes	438
27.19	CANAL Specification History	438

VIII Level I (CANAL) drivers	442
27.20 Drivers for the Windows and Linux platform	442
27.20.1 APOX Driver	442
27.20.2 Lawicel CANUSB Driver	443
27.20.3 Lawicel CAN232 Driver	445
27.20.4 CCS CAN Driver	447
27.20.5 Generic Serial/RS-232 interface Driver	448
27.20.6 Tellstick Driver	448
27.20.7 USB2CAN CAN Driver	451
27.20.8 CAN4VSCP Driver	454
27.20.9 IXATT VCI interface Driver	457
27.20.10 PEAK CAN Adapter Driver	460
27.20.11 Vector CAN Driver	463
27.20.12 Zanthic CAN Driver	465
27.20.13 LIRC Driver	465
27.20.14 Logger Driver	466
27.20.15 TCP Driver	467
27.20.16 UDP Driver	468
27.20.17 AP Driver	469
28 Creating a Level I (CANAL) driver for new hardware	469
IX Level II Drivers	470
29 VSCP driver API	471
29.1 VSCPOpen	471
29.2 VSCPClose	471
29.3 VSCPBlockingSend	472
29.4 VSCPBlockingReceive	472
29.5 VSCPGetLevel	472
29.6 VSCPGet WebPageTemplate	472
29.7 VSCPGet WebPageInfo	473
29.8 VSCPWebPageupdate	473
29.9 VSCPGetDllVersion	473
29.10 VSCPGetVendorString	473
29.11 VSCPGetDriverInfo	473
30 Level II drivers for the Windows and Linux platform	476
30.1 Logger driver	476
30.2 Raw Ethernet driver	477
30.3 Lm-sensors driver	479
30.4 Socketcan driver	483
30.5 TCP driver	484
30.6 MQTT driver	485
30.7 Bluetooth proximity driver	488

X	Cookbook & HOWTO's	489
31	VSCP	489
31.1	How to make a VSCP system with operation confirm	489
31.2	How to receive VSCP events using C	490
31.3	How to send an event from C	496
31.4	How do I send an event from Perl	498
31.4.1	VSCP TCP Connection	498
31.5	How do I send an event from PHP	499
31.5.1	VSCP Broadcast	499
31.5.2	VSCP TCP Connection	500
31.6	How do I send an event from Python	501
31.6.1	VSCP TCP Connection	501
XI	FAQ	503
32	VSCP FAQ	503
32.1	General	503
32.1.1	Q: Where does CANAL end and VSCP start?	503
32.1.2	Q: Does it cost money to use VSCP?	503
32.1.3	Q: Can I make commercial applications from the code? .	503
32.1.4	Q: I need a vendor ID. How do I get it?	503
32.1.5	Q: I can't find anything about routing between segments in the specifications.	504
32.1.6	Q: Ouch!!! I try to understand all this but its just to much. How do I start?	504
32.1.7	Q: Zone/sub-zone. What are they used for?	505
32.1.8	Q: I don't understand the event naming convention. Please explain.	505
32.1.9	Q: Must my device understand all of them events?	505
32.2	VSCP over CAN	506
32.2.1	Q: What transfer speed does VSCP use?	506
32.2.2	Q: How long cable can be used?	506
32.2.3	Q: Can VSCP and CANopen be used on the same bus? .	506
32.3	VSCP and GUID	506
32.3.1	Q: What is the GUID?	506
32.3.2	Q: Do i have to pay for a vendor ID?	506
32.3.3	Q: From where can I get my own VSCP GUID?	506
32.3.4	Q: Do I have to be a big company to get my own VSCP GUID?	506
32.3.5	Q: Our big big big company need more then just 4294967295 unique GUID's. Can we get more?	507
32.3.6	Q: I already have a MAC address or some other vendor ID. Can I use that as the basis for a GUID?	507

32.3.7 Q: How is the GUID from a device connected to the daemon organized?	507
32.4 VSCP Level I	507
32.4.1 Q: I want to use the same nickname for a faulty module that is replaced. How do I do that?	507
32.4.2 Q: How is modules designed that contains combined functionality?	508
32.4.3 Q: My application needs more than 128 registers. How do I solve that?	509
32.5 VSCP Level II	509
32.5.1 Q: How is data laid out in Level II over UDP?	509
32.5.2 Q: How are filters/masks used?	509
32.6 The VSCP daemon	509
32.6.1 Q: Windows: The VSCP daemon is started but nothing works.	509
32.6.2 Q: How are events routed by the daemon?	510
32.6.3 Q: I get error about a missing shared library when I try to start vscpd under Linux/Unix.	510
32.6.4 Q: I use the logger driver under Unix and get two events in the log for every sent.	510
33 CANAL FAQ	511
33.1 General Drivers	511
33.1.1 How to get going with driver x on windows.	511
XII Appendix	513

List of Figures

1	Node discovery	67
---	----------------	----

List of Tables

1	Ethernet frame content	60
2	Payload usage	64
3	VSCP mandatory registers, part 1	77
4	VSCP mandatory registers, part2	78
5	Level I matrix row	80
6	Flag bit description	81
7	class/type filter	81
8	Level II decision matrix format	83
9	Message prefixes	289
10	Websocket commands	290
11	CANAL Levels	430
12	CANAL error codes	439
13	CANAL message ID flags	440
14	CANAL status flags	440
15	CAN4VSCP protocol flags	456
16	CAN4VSCP Error packet code	456
17	CAN4VSCP command response codes	457

Part I

Specification

This specification is a reference document. With +500000 characters written by someone that is not native in English it may be a pain to read for many. If you just want to use VSCP & Friends skip to section 2. Go back to the specification when you are in doubt about usage or just need a deeper understanding of what's going on. VSCP is designed to make complex systems simple to use and to necessarily easy to build such complex systems. We expect a greater technical knowledge and at least the ability to read from people that build and maintain such systems.

Another resource for information is the VSCP wiki (<http://www.vscp.org/wiki/doku.php>) it holds a lot of useful information and howto's.

1 Introduction

VSCP is an open source standard protocol for m2m, IoT and other remote control and measurement applications. It enables simple, low-cost devices to be networked together with high-end computers and/or to work as an autonomous system, whatever the communication media is.

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, IoT, M2M. More and more RF solutions are now available: Bluetooth, Zigbee, Z-wave, Wifi, etc. Other systems use a dedicated bus based on RS-485, 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 are part of the alliance and pay your yearly fees, or similar. There also is small companies that have their own proprietary and completely closed protocols.

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.
- Discovery and identification. Installed devices should be possible to discover and be identified in an uniform way.
- Uniform device configuration. Devices should be able to be configured in a uniform way.

- Autonomous/distributed device functionality.
- Uniform way to update/maintain device firmware.

Some features:

- Free and open for commercial and other use.
- Have two levels. Level I and Level II where level I is designed with CAN as the least common denominator. Can be used for TCP, UDP, RF, Mains communication, etc etc.
- Has globally unique IDs for each node.
- Has a mechanism to automatically assign a unique ID to a newly installed node and inform other nodes and possible hosts that a new node is available and ready.
- Use “registers” as a uniform way to configure nodes.
- Can use a “decision matrix” to program nodes with dynamic functionality.
- Has a common specification language “MDF” that describe a module in a uniform way that can be used by set up software and such.
- Has software and drivers for Windows and Linux. More added all the time.

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 micro-controllers) 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.
- is easy to configure.

1.1 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.

1.2 License

Alternative licenses for VSCP & Friends may be arranged by contacting Grodans Paradis AB at info@grodansparadis.com, <http://www.grodansparadis.com>

1.2.1 License options

VSCP & Friends is licensed under three different licenses.

- Commercial users can buy a license from Grodans Paradis AB.
- Non commercial users can use the GPL version.
- Both commercial and non-commercial users can use the LGPL code such as libraries and firmware code.

1.2.2 Open License

As of February 2006, VSCP & Friends is released under a modified version of the well known GNU General Public License (GPL), now making it an official GPL-compatible Free Software License. An exception clause has been added to the VSCP & Friends license which limits the circumstances in which the license applies to other code when used in conjunction with VSCP & Friends. The exception clause is as follows:

As a special exception, if other files instantiate templates or use macros or inline functions from this file, or you compile this file and link it with other works to produce a work based on this file, this file does not by itself cause the resulting work to be covered by the GNU General Public License. However the source code for this file must still be made available in accordance with section (3) of the GNU General Public License.

This exception does not invalidate any other reasons why a work based on this file might be covered by the GNU General Public License.

The goal of the license is to serve the VSCP & Friends user community as a whole. It allows all VSCP & Friends users to develop products without paying anything, no matter how many developers are working on the product or how many units will be shipped. The license also guarantees that the VSCP & Friends source code will always be freely available. This applies not only to the core VSCP & Friends code itself but also to any changes that anybody makes to the core. In particular, it should prevent any company or individual contributing code to the system and then later claiming that all VSCP & Friends users are now guilty of copyright or patent infringements and have to pay royalties. It should also prevent any company from making some small improvements, calling the result a completely new system, and releasing this under a new and less generous license.

The license does not require users to release the source code of any applications that are developed with VSCP & Friends. However, if anybody makes any changes to code covered by the VSCP & Friends license, or writes new files derived in any way from VSCP & Friends code, then we believe that the entire user community should have the opportunity to benefit from this. The license stipulates that these changes must be made available in source code form to all recipients of binaries based on the modified code, either by including the sources along with the binaries you deliver (or with any device containing such binaries) or with a written offer to supply the source code to the general public for three years. It is perhaps most practical for VSCP & Friends developers to make the source code available online and inform those who are receiving binaries containing VSCP & Friends code, and probably also the VSCP & Friends maintainers, about the location of the code. See the full text of the GPL for the most authoritative definition of the obligations.

Although it is not strictly necessary to contribute the modified code back to the VSCP & Friends open source project, we are always pleased to receive code contributions and hope that developers will also be keen to give back in return for what they received from the VSCP & Friends project completely free of charge. The VSCP & Friends maintainers are responsible for deciding whether such contributions should be applied to the public repository. In addition, a copyright assignment is required for any significant changes to the core VSCP & Friends packages.

The result is a royalty-free system with minimal obligations on the part of application developers. This has resulted in the rapid uptake of VSCP & Friends. At the same time, VSCP & Friends is fully open source with all the benefits that implies in terms of quality and innovation. We believe that this is a winning combination.

1.2.3 Commercial License

You have the right to incorporate any code of VSCP & Friends in your own application and resell it without any need to re-share any code.

The commercial licenses is available from Grodans Paradis AB, info@grodansparadis.com.

1.2.4 Questions and answers

The following queries provide some clarification as to the implications of the VSCP & Friends license. They do not constitute part of the legal meaning of the license.

- Q. What is the effect of the VSCP & Friends license?
- A. In the simplest terms, when you distribute anything containing VSCP & Friends code, you must make the source code to VSCP & Friends available under the terms of the GPL.
- Q. What if I make changes to VSCP & Friends or write new code based on VSCP & Friends code?
- A. Then you must make those changes available as well.
- Q. Do I have to distribute the source code to my application? Isn't the GPL "viral"?
- A. You do not have to distribute any code under the terms of the GPL other than VSCP & Friends code or code derived from VSCP & Friends. For example, if you write a port based on copying an existing VSCP & Friends in any way, you must make the source code available with the binary. However you would not need to make available any other code, such as the code of a wholly separate application linked with VSCP & Friends.

1.2.5 Full GNU GPL License

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This file is part of the VSCP (<http://can.sourceforge.net>) Copyright (C) 2000-2013 Ake Hedman, Grodans Paradis AB, <akhe@grodansparadis.com>

GNU GENERAL PUBLIC LICENSE

Version 3, 29 June 2007

Copyright © 2007 Free Software Foundation, Inc. <<http://fsf.org/>>

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Preamble The GNU General Public License is a free, copyleft license for software and other kinds of works.

The licenses for most software and other practical works are designed to take away your freedom to share and change the works. By contrast, the GNU General Public License is intended to guarantee your freedom to share and

change all versions of a program—to make sure it remains free software for all its users. We, the Free Software Foundation, use the GNU General Public License for most of our software; it applies also to any other work released this way by its authors. You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for them if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs, and that you know you can do these things.

To protect your rights, we need to prevent others from denying you these rights or asking you to surrender the rights. Therefore, you have certain responsibilities if you distribute copies of the software, or if you modify it: responsibilities to respect the freedom of others.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must pass on to the recipients the same freedoms that you received. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

Developers that use the GNU GPL protect your rights with two steps: (1) assert copyright on the software, and (2) offer you this License giving you legal permission to copy, distribute and/or modify it.

For the developer's and author's protection, the GPL clearly explains that there is no warranty for this free software. For both user's and author's sake, the GPL requires that modified versions be marked as changed, so that their problems will not be attributed erroneously to authors of previous versions.

Some devices are designed to deny users access to install or run modified versions of the software inside them, although the manufacturer can do so. This is fundamentally incompatible with the aim of protecting user's freedom to change the software. The systematic pattern of such abuse occurs in the area of products for individuals to use, which is precisely where it is most unacceptable. Therefore, we have designed this version of the GPL to prohibit the practice for those products. If such problems arise substantially in other domains, we stand ready to extend this provision to those domains in future versions of the GPL, as needed to protect the freedom of users.

Finally, every program is threatened constantly by software patents. States should not allow patents to restrict development and use of software on general-purpose computers, but in those that do, we wish to avoid the special danger that patents applied to a free program could make it effectively proprietary. To prevent this, the GPL assures that patents cannot be used to render the program non-free.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS

0. Definitions. “This License” refers to version 3 of the GNU General Public License.

“Copyright” also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

“The Program” refers to any copyrightable work licensed under this License. Each licensee is addressed as “you”. “Licensees” and “recipients” may be individuals or organizations.

To “modify” a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a “modified version” of the earlier work or a work “based on” the earlier work.

A “covered work” means either the unmodified Program or a work based on the Program.

To “propagate” a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

To “convey” a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

An interactive user interface displays “Appropriate Legal Notices” to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.

1. Source Code. The “source code” for a work means the preferred form of the work for making modifications to it. “Object code” means any non-source form of a work.

A “Standard Interface” means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

The “System Libraries” of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A “Major Component”, in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used

to produce the work, or an object code interpreter used to run it.

The “Corresponding Source” for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work’s System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source.

The Corresponding Source for a work in source code form is that same work.

2. Basic Permissions. All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.

Conveying under any other circumstances is permitted solely under the conditions stated below. Sub licensing is not allowed; section 10 makes it unnecessary.

3. Protecting User’s Legal Rights From Anti-Circumvention Law. No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.

When you convey a covered work, you waive any legal power to forbid circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as

a means of enforcing, against the work's users, your or third parties legal rights to forbid circumvention of technological measures.

4. Conveying Verbatim Copies. You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee.

5. Conveying Modified Source Versions. You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

- a) The work must carry prominent notices stating that you modified it, and giving a relevant date.
- b) The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to "keep intact all notices".
- c) You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it.
- d) If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.

A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an "aggregate" if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation's users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.

6. Conveying Non-Source Forms. You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

- a) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange.
- b) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge.
- c) Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and non commercially, and only if you received the object code with such an offer, in accord with subsection 6b.
- d) Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements.
- e) Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no charge under subsection 6d.

A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A “User Product” is either (1) a “consumer product”, which means any tangible personal property which is normally used for personal, family, or household

purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, “normally used” refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

“Installation Information” for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM).

The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying.

7. Additional Terms. “Additional permissions” are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

When you convey a copy of a covered work, you may at your option remove

any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission.

Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms:

- a) Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or
- b) Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or
- c) Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or
- d) Limiting the use for publicity purposes of names of licensors or authors of the material; or
- e) Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or
- f) Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.

All other non-permissive additional terms are considered “further restrictions” within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits re licensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such re licensing or conveying.

If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way.

8. Termination. You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10.

9. Acceptance Not Required for Having Copies. You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.

10. Automatic Licensing of Downstream Recipients. Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

An “entity transaction” is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party’s predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this

License, and you may not initiate litigation (including a cross-claim or counter-claim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it.

11. Patents. A “contributor” is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor’s “contributor version”.

A contributor’s “essential patent claims” are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, “control” includes the right to grant patent sub licenses in a manner consistent with the requirements of this License.

Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor’s essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

In the following three paragraphs, a “patent license” is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To “grant” such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.

If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. “Knowingly relying” means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient’s use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is “discriminatory” if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a

third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law.

12. No Surrender of Others Freedom. If conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program.

13. Use with the GNU Affero General Public License. Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU Affero General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the special requirements of the GNU Affero General Public License, section 13, concerning interaction through a network will apply to the combination as such.

14. Revised Versions of this License. The Free Software Foundation may publish revised and/or new versions of the GNU General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies that a certain numbered version of the GNU General Public License “or any later version” applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU General Public License, you may choose any version ever published by the Free Software Foundation.

If the Program specifies that a proxy can decide which future versions of the GNU General Public License can be used, that proxy’s public statement of

acceptance of a version permanently authorizes you to choose that version for the Program.

Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version.

15. Disclaimer of Warranty. THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. Limitation of Liability. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

17. Interpretation of Sections 15 and 16. If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

END OF TERMS AND CONDITIONS

1.3 Where is the project located?

Current information about VSCP (Very Simple Control Protocol) can be found at:

<http://www.vscp.org>

There are three mailing lists available on Sourceforge that are about CANAL, VSCP and OHAS topics.

- To subscribe to the CAN/CANAL list go to http://sourceforge.net/mail/?group_id=53560
- To subscribe to the VSCP list go to http://sourceforge.net/mail/?group_id=224170

1.4 When was the project started?

Many of the thoughts behind this protocol come from work done by Ake Hedman as early as 1986 but the 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>) was registered at Sourceforge. EDA, is an acronym for *Event-Decision-Action* and is still preserved in the decision matrix of VSCP.

1.5 Why another protocol?

VSCP is designed to be used where other solutions are too expensive to implement. This can typically be in code overhead where most other protocols use more resources (flash/ram) on a micro-controller than the actual “application” adding a lot of cost to the project. VSCP can work with a typical lowest cost dumb nodes like the Microchip MCP250xx series, to a 1K-2K flash micro controller module up to a full implementation with all features in about 5K flash. VSCP is both free and open. Everyone can join the project and help to add features and functionality to the protocol. There is free firmware and driver code available for most micro-controllers which uses many communication techniques. Everyone is free to use this code and there is no requirement to share any new code you develop, although most choose to, in order to improve the code base.

1.6 Why is the VSCP protocol needed?

Most people don’t remember the world looked like for PC developers before Windows were introduced. This was a time when the developer needed to ship a big bunch of floppy disks with his/hers application. One big pile of floppies 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 and 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 its own format and does its specific work using its own protocol and return a

CLASS1.INFORMATION, Type=3, ON *event*

When its job is done as a confirmation for the rest of the system.

An application 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 events.

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 are 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.

1.7 How does it work?

1.7.1 Events

VSCP is an event based system. Nodes generate events and nodes react on events. Normally events are not addressed but instead are broadcast on the bus. Its up to the receiving end to decide if its interested in the event or not. All events have an originating address for the node they are sent from. This is a GUID consisting of 16 bytes but a shorter, typically, one byte nickname-ID is used on most system. It is always possible to deduce the full GUID from the nickname. Events are divided into groups. First there is Level I and Level II events. Level I events are limited to a maximum of eight bytes of data while Level II events can have up to 488 bytes of data. The low maximum data count for Level I comes from that CAN has been used as the least common denominator. This does not limit VSCP Level I to be used only over CAN. Both Level I and Level II events are divided into a class and a type. The class defines a group of events of a specific type. Typical examples are classes for measurements, information and control. As mentioned above events are not addressed. This is not entirely true as one class in each level (CLASS1.PROTOCOL and CLASS2.PROTOCOL) have events that are addressed. These classes define protocol functionality that all nodes have to implement. Typical examples found there are event types for boot loader, register access and status information. Most of the events can use a zone/sub-zone to identify the group of nodes it is relevant to.

1.7.2 Registers

Each node has a specified number of byte wide registers defined. This is the second interface to the black box the node represents. The register space of a node is 256 bytes divided in two halves. The lower part (0x00-0x7F) is application specific. The upper part content is mandatory and specified by VSCP. This space hold, among many other things, the firmware and hardware version of the node. The GUID, user module id, alarm status and similar registers. In the top of the mandatory register space is a 32 byte string stored, the MDF, which contains an URL that points to a location where to find an XML file that describe this module its registers and its functionality. The lower part of the register space is used to define control and status registers and more complex structures. The area is mapped and 65535 pages can be used if needed allowing for very complex scenarios if needed.

1.7.3 MDF - Module Description File

The MDF file describe the module at a high level. It provides information about the manufacturer of the module, the events that can be expected from it and the definition of its register content. The MDF looks at the module from a high level perspective and can see floating point values, bit arrays, option tables in register space and thus give application software a way to present registers in a

much more user friendly way. There is two way to obtain the MDF after getting it from the module by reading the registers holding it . As VSCP is designed for low end nodes it is normally to much for such a node to have the MDF stored internally. In this case the string points to a web server from which the XML can be downloaded and processed. In a more capable module with more resources the MDF is stored internally in which case the read string has zero length and in this case the MDF data can be fetched from the module itself. The node constructor decide which method he/she wants to use. The MDF also point out where drivers for different environments and systems for the module and user manuals etc can be found.

1.7.4 DM - Decision Matrix

All nodes can optionally implement a decision matrix. This matrix is used to define one or a group of events that should trigger a predefined functionality at the module. In VSCP this is called the Event-Decision-Action mechanism from the predecessor of the project which was called EDA. A typical examples can be to trigger a relay. The module got one action that set the relay and another action that reset it. Typically this could be implemented hard coded so that the relay would be set when an ON-event was received and has been reset when an OFF-event was received. With the use of the DM it is easy to use any event to trigger the action not just the ON/OFF events. Physically the DM is located in register space just as any other parameters of the module so register access functions are used to changed it.

1.7.5 Measurements

A special class is used for measurements and events for all SI units and derived units is defined. The class will grow over time when new types are needed. This means that for example a temperature measurement is normally sent as a Kelvin temperature. Celsius and Fahrenheit is also possible in this case and similar alternatives is available for other types but the SI unit is always the default. How the measurement is presented in the frame is also well specified. Bit field, string, integer, normalized integer (decimal) and floating point values are possible. The normalized integer is especially well suited for a low end system to send decimal data. As the event and its content is well specified it is very easy to interpret the data on the receiving end where it should be processed, stored in a database, logged or displayed.

1.7.6 Putting it all together

As VSCP is event based, one often need to think a bit differently when constructing a system. A typical example is a tank with a level sensor and a pump which together construct a self contained system. A traditional system system would have used a master to control the pump and the sensor. In VSCP we allow for this also but try to distribute as much of the intelligence as possible to the nodes themselves. So what we do is to tell the level sensor to send

level information with a predefined interval. Several sensors can be added for redundancy. We tell the pump that it should start to fill up the tank when the tank reach a low level and stop at a high level. This might be dangerous as the sensor may stop working, the cable get cut or whatever. In this case the we also program the pump goes to its safe state == pump off and alarming this state to the rest of the system when it does not receive sensor measurements any more.. This “safe state“ is often possible to find for most control situations. As the transport mechanism is unknown to the application, timing must be very lose for VSCP. We cannot be certain when an event arrives or if it arrives. Many transport mechanisms, such as CAN and TCP/IP, makes delivery more certain while other solutions like UDP, RF and PLC can be problematic. It is therefore very important that a sent event always get a confirmation event back. For some nodes this might not be important. A temperature node or the level sensor node above just send there measurements and don’t care if someone uses them or not. This thinking is central. The node that originate an event should – if possible - know as little as possible about how the event it sends out will be used. This make the system very flexible.

1.7.7 The Friends

A software package called **VSCP & Friends** is available to support VSCP users. This package can be downloaded and used for free and contain a lot of application and code. Everything is available both for Windows and Linux based systems.

First and possibly most important it contain the VSCP daemon. This is a server software that makes it possible to control several VSCP segments over the Internet.

The server expose a secure Internet interface and makes it possible to add drivers for segments of nodes or special equipment. A driver can communicate with the server using the CANAL interface for a Level I driver and the TCP/IP interface for a Level II driver.

CANAL stands for CAN Abstraction Layer and was initially developed as a software layer between application software and CAN equipment. For the VSCP it is not limited to interface CAN equipment but CAN drivers for most CAN adapters such as PEAK, IXXAT, PORT etc is included in the package as many CAN users use the system.

A lot of code is available that confirms to the CANAL interface. This same code can be used to communicate directly to a device that use the CANAL driver as can be used to talk to the daemon. This makes, among many other advantages, it easy to build simulated systems that use the same code base as the resulting system.

The daemon can do a lot more such as remotely replace/install drivers, remotely handle users and permissions, set up and handle an internal decision matrix to set up a self contained control system etc.

VSCP Works is a client application that is included in the package. This application can be used to send/receive VSCP events to/from every segment/de-

vice that export a CANAL interface and also talk to a remote VSCP daemon. VSCP Works can be used to investigate register space on any VSCP node and will also have support for remote firmware upgrade.

Server and client application is released under the GPL license. Libraries and classes is released under a modified LGPL license that make sure they can be used in proprietary projects.

Maybe the most important aspect of the VSCP & Friends package is that it can be used as an abstraction for interfacing other technologies and protocols. One just need to build a driver that translate the systems functionality to/from VSCP events and then install this driver for the VSCP daemon and then remote control functionality, software interface etc is directly available. This way one application interface can be constructed to control several technologies using different protocols. One can compare this to the situation for printers before windows was announced. Each application needed to distribute a stack of disks with printer drivers. Windows ended this by introducing the printer API abstraction for printers. VSCP & Friends does the same for SECO (SEensor/COntrol) devices .

A web interface named OHAS is under development that will make it possible to build dynamic control interfaces with drag and drop technology.

1.7.8 Summary

VSCP makes it easy and very cost effective to build systems with distributed intelligence. The protocol is placed in the Public Domain so it is therefore free for anyone to use and modify to their own needs. VSCP can be used all the way from the application down to the device but it can also be used as an abstraction to other technologies so one application can be written that transparently use several different technologies..

More information and downloads is available at [HTTP://www.vscp.org](http://www.vscp.org)

2 Physical Level/Lower level protocols

When the protocol was designed its 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.

2.1 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 bit rates can be used too. See chapter about using alternative bit rates below]].

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 24 meters 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)
1Mbps	25	2	10
800 Kbps	50	3	15
500 Kbps	100	6	30
250 Kbps	250	12	60
125 Kbps	500	24	120
50 Kbps	1000	60	300
20 Kbps	2500	150	750
10 Kbps	5000	300	1500

CAN bit rate data

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

VSCP requires Extended Data Frames with a 29-bit ID.

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%.

2.1.1 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	Hard-coded	If this bit is set the nickname-ID of the device is hard-coded
24	Class	The class identifies the event 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 event within the set event 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 (hard-coded bit set) or an address retrieved through the nickname discovery process. 0x00 is reserved for a segment master node. 0xFF is reserved for no assigned nickname.
6	Originating-Address	
5	Originating-Address	
4	Originating-Address	
3	Originating-Address	
2	Originating-Address	
1	Originating-Address	
0	Originating-Address	

Format of the 29 bit CAN identifier in VSCP

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.

2.1.2 RJ-XX pin-out

Pin	Use	RJ-11	RJ-12	RJ-45	Patch Cable wire color T568B
1	+9-28V DC			RJ-45	Orange/White
2 1	+9-28V DC		RJ-12	RJ-45	Orange
3 2 1	+9-28V DC	RJ-11	RJ-12	RJ-45	Green/White
4 3 2	CANH	RJ-11	RJ-12	RJ-45	Blue
5 4 3	CANL	RJ-11	RJ-12	RJ-45	Blue/White
6 5 4	GND	RJ-11	RJ-12	RJ-45	Green
7 6	GND		RJ-12	RJ-45	Brown/White
8	GND			RJ-45	Brown

RJ-11/12/45 pin-out

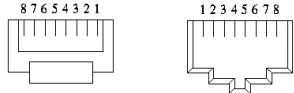


Figure 1:
End view of RJ45 Plug Figure 2:
Looking into an RJ45 Jack

RJ-45 pin out

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> which is opposite numbered VSCP but they are actually the same. We use the numbering that look into the female connector and count from the left. They use the Arabic style and count from the right.

Always try to use a pair of wires for CANH/CANL for best noise immunity. If the EIA/TIA 56B standard is used this condition will be satisfied. This is good as most Ethernet networks already is wired this way.

2.1.3 Using alternative bit-rates

Don't! The bit-rate should be fixed at 125kbps.

2.2 VSCP Level I over RS-232

◆Preliminary Information. May change in future specifications.

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 implementer.

2.2.1 General frame format

The checksum is calculated from, and including, flags up to the last data-byte.

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 frames are sent after each other and they will get different counter numbers. The NACKs/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.

As noted there can be two types of ACKs/NACKs 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.

Byte 0 - [DLE]

Byte 1 - [STX]

Byte 2 - 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.

Byte 3 - Flags + bit nine of class

- Bit 0-4 - Number of data-bytes
- Bit 5 - Bit nine of Class.
- Bit 6-7 - Reserved and should be set to 0.

Byte 4 - Channel

- Sequence number

Byte 5 - Class

- Lower 8 bits of class.

Byte 6 - Frame type

Byte 7 - Data byte(s) 0 - 15

- There can be 0 to 16 data-bytes.

Byte len-2 - Frame check sum.

- XOR check sum is used.

Byte len-1 - [DLE]

Byte len-0 - [ETX]

2.2.2 Command frame

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.

[DLE]

[STX]

- 255 = Command.

Flags

- bit 7-5 reserved.
- bit 0-4 - Number of data-bytes.

Channel

Sequence number

Command MSB

Command LSB

Optional command data byte(s) 0-15

Frame checksum

[DLE]

[ETX]

2.2.3 Error frame

There can be 0 to 16 argument bytes.

[DLE]

[STX]

- 253 = error.

Flags

- bit 7-5 reserved.
- bit 0-4 - Number of data-bytes.

Channel

Sequence number

Error code MSB

Error code LSB

Optional error data byte(s) 0-15

Frame checksum

[DLE]

[ETX]

2.2.4 Command Reply

There can be 0 to 16 argument bytes.

[DLE]

[STX]

- 254 = response.

Flags

- bit 7-5 reserved.
- bit 0-4 - Number of data-bytes.

Channel

Sequence number

Command reply code MSB

Command reply code LSB

Optional error data byte(s) 0-15

Frame checksum

[DLE]

[ETX]

2.2.5 ACK frame

Used as a positive response.

[DLE]

[STX]

251 = ACK.

Channel

Sequence number

Frame checksum

[DLE]

[ETX]

2.2.6 NACK frame

Used as a negative response.

[DLE]

[STX]

252 = NACK.

Channel

Sequence number

Frame checksum

[DLE]

[ETX]

2.2.7 Special characters

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

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

2.2.8 Sending CAN frames with the protocol

For generic CAN frames the operation byte is set to 5 The first four data-bytes form the CAN ID.

The four last data-bytes can be a time-stamp (big endian).

- Bit 31 (offset 0) of the ID is Extended frame.
- Bit 30 is remote frame.
- Bit 29 is error frame.

2.2.9 Command codes

Command codes and data format is decided by the implementer.

2.2.10 Response codes

Response codes and data format is decided by the implementer.

2.2.11 Error codes

Error codes and data format is decided by the implementer.

2.3 VSCP Level I 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 Baudrate)

Format: 8N1 for packages. 9N1 for address.

2.3.1 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 hard-coded within devices and each device should have its own unique address. Nine bit addressing is described here <http://electronicdesign.com/embedded/use-pcs-uart-9-bit-protocols>

2.3.2 Packet format

If a node has an event to send it sends it directly after the poll event. The node should respond within one maximum packet time ($14 * 8 * 104\mu\text{S} = 10 \text{ ms}$). 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 events is standard VSCP Level I events in every other aspect.

A master on a RS-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.

Nine bit destination address

Originating address (master=0)

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 data-bytes
- bit 4 - Bit nine of Class.
- bit 5-7 - reserved and should be set to 0.

VSCP Class (Lower 8 bits).

VSCP 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). The DOW checksum is used. (http://www.vscp.org/wiki/doku.php?id=vscp_specification_crc_polyomns.)

2.4 VSCP over Ethernet (raw Ethernet)

VSCP can use Ethernet directly making it possible to construct very low end nodes. There is no need for nickname schema's etc as a VSCP GUID can be constructed from an Ethernet MAC. The payload is a standard VSCP level II frame with some specific Ethernet stuff in it

The VSCP GUID for Ethernet is defined as

FF FF FF FF FF FF FE YY YY YY YY YY YY XX XX

where *FF FF FF FF FF FF FE* say this is an Ethernet GUID and *YY YY YY YY YY YY* is the MAC address and *XX XX* is the ID for a specific VSCP device. Each node can thus appear as 65535 - 2 nodes which is perfect for a gateway or similar device that have VSCP devices connected on one or many other buses such as CAN, RF and the like. Two IDs are reserved. 0x0000 which is the Ethernet node itself and 0xFFFF which is all nodes on the interface. If a node consist just of itself then it can ignore the VSCP sub address but it is recommended that it use 0x0000 for its transmitted frames.

The header (32-bits) is defined as follows

CONTENT	SIZE	DESCRIPTION
Priority (Bit 31,30,29)	3 bits	Standard VSCP priority.
Cryptographic algorithm (Bit 28, 27, 26, 25)	4 bits	Cryptographic algorithm.
Reserved	25 bits	

Cryptographic encoding is done after type up to and including the last data byte thus the VSCP head is never encoded.

Content	Size	Description
Preamble	62 bits	Standard Ethernet preamble. (Part of Ethernet)
SOF (Start of Frame)	2 bits	Standard Ethernet SOF. (Part of Ethernet)
Destination MAC address	6 bytes	Always set to <i>FF:FF:FF:FF:FF:FF</i> (Part of Ethernet)
Source MAC address	6 bytes	Source address. Part of GUID see information below. (Part of Ethernet)
Ethernet Type	2 bytes	Frame type: Always set to <i>0x257E</i> (<i>decimal 9598</i>) (Part of Ethernet)
Version	1 byte	Currently zero for this version. May change in the future.
VSCP head	4 bytes	Header
VSCP sub source address	2 bytes	Identifies one subunit of an Ethernet device. Last two bytes of the GUID.
Timestamp	4 bytes	Timestamp in microseconds. Module relative value.
obid	4 bytes	For use by modules and drivers.
VSCP class	2 bytes	16-bit VSCP class
VSCP type	2 bytes	16-bit VSCP type
DataSize	2 bytes	Size of data part. Needed because Ethernet frames with have a minimum length of 64 bytes.
Payload	0- 487 bytes	The VSCP data part. As usual 512-25 bytes.
CRC	4-bytes	32 bit CRC checksum (Part of Ethernet)

Table 1: Ethernet frame content

2.4.1 Powering Ethernet nodes

The unused pairs can be used to power nodes and that way making them lower cost. In a perfect world a standard compliant Power over Ethernet schema could have been used. Regrettably the technique is too expensive. At least at the moment. Instead a +9-48V DC injector is used. If in doubt use +48V DC. The IEEE 802.3af standard POE pin out is as follows:

Pin	Description
1	
2	
3	
4	+9V - +48V
5	+9V - +48V
6	
7	GND
8	GND

Maximum power usage depends on the cabling but for AWG-24 it is 0.5A for each pair but never more than 13 watts. There is a description of how different manufacturers implement this here http://pinouts.ru/Net/poe_pinout.shtml. A calculator for losses can be found here [Power calculator](#).

2.5 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 of VSCP over TCP/IP. One UDP version which is broad-casted on a segment and the other which is more secure and use the TCP low level protocol.

2.6 VSCP over UDP

Default Port: 9598¹

Format The packet format is

Byte 0 - HEAD

Byte 1 - CLASS MSB

Byte 2 - CLASS LSB

Byte 3 - TYPE MSB

Byte 4 - TYPE LSB

Byte 5 - ORIGINATING ADDRESS MSB

...

Byte 20 - ORIGINATING ADDRESS LSB

Byte 21 - DATA SIZE MSB

Byte 22 - DATA SIZE LSB

... data ... limited to max 512-25 = 487 bytes

Byte len -2 - CRC MSB (Calculated on HEAD + CLASS + TYPE + ADDRESS + SIZE + DATA...)

Byte len -1 - CRC LSB The number above is the offset in the package. Len is the total datagram size.

¹The port was 1681 up to release 0.1.6

HEAD The HEAD is defined as

- bit 7,6,5 - Priority
- bit 4 - Hard-coded
- bit 3,2,1,0 reserved (**Set to zero!**)

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.

As indicated, the Class is 16-bits allowing for 65535 classes. Class 0000000x xxxxxxxx is reserved for the Level I classes. A low-end device should ignore all packages with a class > 511.

A packet traveling from a Level I device out to the Level II world should have an address translation done by the master so that a full address will be visible on the level II segment. A packet traveling from a Level II segment to a Level I segment must have a class with a value that is less than 512 in order for it to be recognized. If it has it is aimed for the Level I segment. Classes 512-1023 are reserved for packets that should stay in the Level II segment but in all other aspects (the lower nine bits + type) are defined in the same manner as for the low-end net.

The Level II 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.

2.6.1 VSCP TCP Interface

The VSCP TCP interface is a telnet type interface.

For more information see 14.6 on page 249

Byte	Description	
0	Number of VSCP events in frame (1-x)	
1 head:	bit	Description
	bit 7,6,5	Priority
	bit 4	bit 9 of class
	bit 3,2,1,0	reserved (Set to zero!)
2	class	
3	type	
4	Start of data. Max eight byte. Min 0	
4 + data-count + 1	head for next event if any	

Table 2: Payload usage

2.7 VSCP over IEEE 802.15.4 protocol (MiWi)

◆Preliminary Information. May change in future specifications.

2.7.1 What is MiWi

MiWi is Microchips solution to get a simpler protocol then Zigbee according to IEEE 802.15.4. MiWi documents can be found here. Special requirements for VSCP over MiWi

There are no special requirements other then that nodes on a MiWi net should preferably implement CLASS1.PROTOCOL, Type=40 and CLASS1.PROTOCOL, Type=41 to loosen the burden of the PAN coordinator to resend all events to all node. For example a temperature node is normally not interested in anything else then CLASS1.PROTOCOL and it is a waste of bandwidth to send other events to it if the PAN know it just will through them away.

The PAN must have buffers and intelligence to be able to handle event filtering for all connected nodes. A standard node however does not need to have this extra burden.

There is no requirement for a node other then the PAN and coordinators to be active all the time. However all nodes must at least send a one minute heartbeat. CLASS1.INFORMATION, Type=9. After a node sent an heartbeat the node should be awake for a minimum of five seconds (!: Time value may change) and receive any collected events intended for it.

Just as most other VSCP devices, VSCP MiWi devices need a n initialization sequence. In this case the intention of the sequence is to hook together a node and a PAN. This is typically done by activating a discover mode on the PAN and then press a button on the node that should be connected to this PAN.

2.7.2 802.15.4 payload usage

The 802.15.4 payload is max 122 bytes. This payload can carry one or more VSCP events. The first byte tell how many events there is in the frame.

3 VSCP Level I Specifics

3.1 Level I Node types

On each segment there can be two kind of nodes. Dynamic and hard-coded.

3.1.1 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 events 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 event 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>

3.1.2 Hard coded nodes

VSCP hard-coded nodes have a nickname that is set in hardware and cant be changed.

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.

3.2 Address or “nickname” assignment for Level I nodes

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

A VSCP Level 1 segment can have a maximum of 254 nodes + an additional 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.

3.2.1 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 event (CLASS1.PROTOCOL, TYPE=2,12.1.3) to address 0 (the address reserved for the master of a segment) using 0xFF as its own address. The priority of this event is set to 0x07. The master (if available) now has a chance to assign a nickname to the node ((CLASS1.PROTOCOL, TYPE=2, 12.1.7). Five seconds may be a good interval for which this assignment should happen.

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 (CLASS1.PROTOCOL, TYPE=2, 12.1.4), 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.

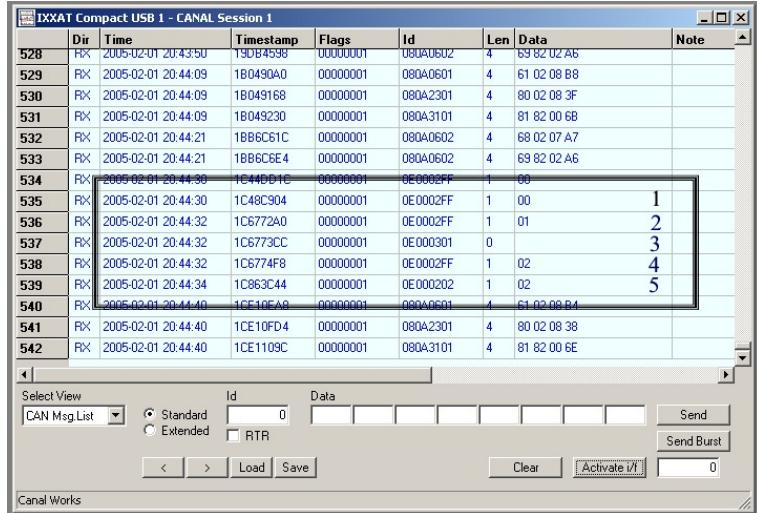


Figure 1: Node discovery

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 preassigned IDs to the nodes. This is just done by connecting each of the nodes to a PC or similar and assigning IDs 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.

3.2.2 Node segment initialization. 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.

3.2.3 Node segment initialization.. 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 list box 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 IDs 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 the 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)

3.2.4 Examples

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 non-volatile memory. By being connected to the segment the installer makes note of the IDs. It is of course also possible to set the nicknames to some desired value instead.

Additionally, VSCP aware relays are installed and also initialized to handle the actual switching of lighting. Again each in 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 event 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 event add information on which zone it controls and the same change is done in the decision matrix we get something 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 set up.

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....

```

// This structure is for VSCP Level II type def struct _vscpMsg
{
    _u16 crc;           // CRC checksum
    _u8 *pdata;         // Pointer to data. Max 487 (512- 25) bytes

    // The 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 priority,
                        // Priority 0-7 where 0 is highest.
                        // bit 4 = hardcoded, true for a hardcoded device.
                        // bit 3 = don't 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;

```

4 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 than 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.

4.1 Level II events

The header for a level II event is defined as

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

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

4.2 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 time-stamp 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 event from a node on a Level I that is transferred 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 implementer 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.

4.3 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 of sending node -->
  <guid>ff:ee:dd:cc:bb:aa:99:88:77:66:55:44:33:22:11:00</guid>
  <timestamp>YYYY-MM-DD HH:MM:SS | millieconds since epoch</timestamp>
  <data>
    Comma separated list with event data.
    Hex values (preceded with '0x') and decimal values allowed.
  </data>
</event>

```

5 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 IDs.

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>

5.1 Predefined VSCP GUID's

It is possible to create your own GUID without requesting a series and still get a valid VSCP GUID.

Assigned Global Unique IDs	IDs Series Reserved to/for
FF FF FF FF FF FF FF FF FF YY YY YY YY YY YY YY YY	Dallas Semiconductor GUID's. This is the 1-wire/iButton 64-bit ID. The device code is in the MSB byte and CRC in the LSB byte.
FF FF FF FF FF FF FF FE YY YY YY YY YY YY XX XX	Ethernet Device GUID's. The holder of the address can freely use the two least significant bytes of the GUID. MAC address in MSB - LSB order. Also called MAC-48 or EUI-48 by IEEE
FF FF FF FF FF FF FF FD YY YY YY XX XX XX XX XX	Internet version 4 GUID's. This is a 32-bit ID so the holder of the address can freely use the four least significant bytes of the GUID. IP V4 address in MSB - LSB order.
FF FF FF FF FF FF FF FC XX XX XX XX XX XX XX XX	Private. Use for in-house local use. The GUID should never appear outside your local segments.
FF FF FF FF FF FF FF FB YY YY YY XX XX XX XX XX	ISO ID. This is a three byte ID so the holder of the ISO ID can freely use the five least significant bytes of the GUID.
FF FF FF FF FF FF FF FA YY YY YY YY XX XX XX XX	CiA (CAN in Automation) vendor ID. This is a 32-bit ID so the holder of the vendor ID can freely use the four least significant bytes of the GUID.
FF FF FF FF FF FF FF F9 YY YY YY XX XX XX XX XX	ZigBee 802.15.4 OID. This is a 24-bit ID so the holder of the OID can freely use the five least significant bytes of the GUID.
FF FF FF FF FF FF FF F8 YY YY YY YY YY YY XX XX	Bluetooth MAC. This is a 48-bit ID so the holder of the OID can freely use the two least significant bytes of the GUID.
FF FF FF FF FF FF FF F7 YY YY YY YY YY YY YY YY	IEEE EUI-64. This is a 64-bit ID. The upper three bytes are purchased from IEEE by the company that releases the product. The lower five bytes are assigned by the device and must be unique.
FF FF FF FF FF FF FF F6 00 YY YY YY YY YY YY YY	Reserved for RAMTRON MRAM (and compatible), seven byte IDs.
FF FF FF FF FF FF FF F6 01 YY YY YY YY YY YY YY - FF FF FF FF FF FF FF F6 FF YY YY YY YY YY YY YY	Reserved other future seven bit ID (memory devices) that may have ID, such as PRAM etc.
FF FF FF FF FF FF FF F5 00 00 00 00 - FF FF FF FF FF FF FF FF FF FF FF FF	Reserved
00 00 00 00 00 00 00 00 00 00 00 00 xx xx xx xx	Lab usage. You can use this range for your own development or for in-house local use. The GUID should never appear outside your local segments.
FE YY YY YY YY YY YY YY YY YY YY YY YY YY YY YY	Reserved for a generated 128 bit GUID where the most significant byte is replaced by FE Only use for Level II and on internal net. http://hegel.ittc.ku.edu/topics/internet/internet-drafts/draft-l/draft-leach-uuids-guids-01.txt

5.2 Shorthand GUID's

Note that there is a convenient shorthand notation :: used for IPV6 that can be used as a place holder for zeros. For example

::1 really means 00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:01

and

FF:21::22:32

is the same as

FF:21:00:00:00:00:00:00:00:00:00:00:00:22:32

we use *: in the same way for FFs so that

*:1 really means

FF:FF:FF:FF:FF:FF:FF:FF:FF:FF:FF:FF:FF:FF:FF:01

These short cut notations makes it much easier to write GUID's.

6 Register Abstraction Model

Functionality of a device in VSCP is exposed to the world through 8-bit registers. Much like a IC circuit expose information to the world in electronic systems. This is useful only for the lowest level devices. Higher level devices and users usually get the information in the registers presented as higher level value. This low level information is something the VSCP systems can hide with the help of the MDF information (see 9). This is this why we talk about an abstraction model.

6.1 Level I - 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 0x00 – 0x7F are application specific. Registers between 0x80 – 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 (0x83) the node behavior can be controlled. The start up control is two bits that can have two values, binary 10 and binary 01. All other values are invalid and are translated to binary 10. If set to binary 10 it will prevent the initialization routine of the node to start before the initialization button on the node has been pressed.

Bit 5 of the node control flags write protects all user registers if cleared (== 1 is write enabled). The page select registers (0x92/0x93) can be used if

more application specific registers are needed. The page is selected by writing a 16-bit page in these positions. This gives a theoretical user registry space of 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.

0x98 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 than eight data bytes if the buffer is larger than eight and the implementer wishes to do so. This even if the event is a Level I event.

The VSCP registers are defined as follows:

Unimplemented registers should return zero.

6.1.1 User ID

This is a register space that can be used to store installation values as for example a location code of where the unit is installed.

6.1.2 Manufacturer device ID/Manufacturer sub device ID

The manufacturer of the device can store whatever they like in these registers. Typical use is for hardware version. Firmware version. Batch numbers. Serial numbers and such.

6.1.3 Page select

The page select registers can be used to switch pages for the lower 128 byte. It's important that an application that uses the page functionality switch back to page 0x0000 when it's ready. Applications can use the page registers as a mutex which is signaled when not zero.

6.1.4 Standard device families and types

Added in version 1.9.0 of the specification

Some devices have a need for a common register structure standard. This is defined by two 32-bit values in the standard register space. The *family value* defines the group of devices the module belongs to and the *type value* describes the specific device type in that family it belongs to. *Families and device types are still to be defined.*

6.2 Level II - Register Abstraction Model

The level II abstraction model is the same as Level I but covers a much wider address space.

Address	Access Mode	Description																		
0x00 – 0x7F	—	Device specific. <i>Unimplemented registers should return zero.</i>																		
128/0x80	Read Only	Alarm status register content (!= 0 indicates alarm). Condition is reset by a read operation. The bits represent different alarm conditions.																		
129/0x81	Read Only	VSCP Major version number this device is constructed for.																		
130/0x82	Read Only	VSCP Minor version number this device is constructed for.																		
131/0x83	Read/Write	Node control flags <table border="1"> <thead> <tr> <th>Bit</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>7</td> <td>Start-up control</td> </tr> <tr> <td>6</td> <td>Start-up control</td> </tr> <tr> <td>5</td> <td>r/w control of registers below 0x80. (1 means write enabled)</td> </tr> <tr> <td>4</td> <td>Reserved</td> </tr> <tr> <td>3</td> <td>Reserved</td> </tr> <tr> <td>2</td> <td>Reserved</td> </tr> <tr> <td>1</td> <td>Reserved</td> </tr> <tr> <td>0</td> <td>Reserved</td> </tr> </tbody> </table>	Bit	Description	7	Start-up control	6	Start-up control	5	r/w control of registers below 0x80. (1 means write enabled)	4	Reserved	3	Reserved	2	Reserved	1	Reserved	0	Reserved
Bit	Description																			
7	Start-up control																			
6	Start-up control																			
5	r/w control of registers below 0x80. (1 means write enabled)																			
4	Reserved																			
3	Reserved																			
2	Reserved																			
1	Reserved																			
0	Reserved																			
132/0x84	Read/Write	User ID 0 – Client settable node-ID byte 0.																		
133/0x85	Read/Write	User ID 1 – Client settable node-ID byte 1.																		
134/0x86	Read/Write	User ID 2 – Client settable node-ID byte 2.																		
135/0x87	Read/Write	User ID 3 – Client settable node-ID byte 3.																		
136/0x88	Read/Write	User ID 4 – Client settable node-ID byte 4.																		
137/0x89	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.																		
144/0x90	Read only	Manufacturer sub device ID byte 3.																		
145/0x91	Read only	Nickname-ID for node if assigned or 0xFF if no nickname-ID assigned.																		
146/0x92	Read/Write	Page select register MSB																		
147/0x93	Read/Write	Page Select register LSB																		
148/0x94	Read Only	Firmware major version number.																		
149/0x95	Read Only	Firmware minor version number.																		
150/0x96	Read Only	Firmware sub minor version number.																		
151/0x97		Boot loader algorithm used. 0xFF for no boot loader support. Codes for algorithms are specified here <code>vscp_event_class_000</code> for Type = 12																		
152/0x98	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 events 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.																		

Table 3: VSCP mandatory registers, part 1

Address	Access Mode	Description
153/0x99	Read Only	<p>Number of register pages used. If not implemented one page is assumed. Set to zero if your device have more then 255 pages.</p> <p><i>Deprecated: Use the MDF instead as the central place for information about actual number of pages.</i></p>
154/0x9A	Read Only	<p>Standard device family code (MSB)</p> <p>Devices can belong to a common register structure standard. For such devices this describes the family coded as a 32-bit integer. Set all bytes to zero if not used. Also 0xff is reserved and should be interpreted as zero was read.</p> <p><i>Added in version 1.9.0 of the specification</i></p>
155/0x9B	Read Only	<p>Standard device family code</p> <p><i>Added in version 1.9.0 of the specification</i></p>
156/0x9C	Read Only	<p>Standard device family code</p> <p><i>Added in version 1.9.0 of the specification</i></p>
157/0x9D	Read Only	<p>Standard device family code (LSB)</p> <p><i>Added in version 1.9.0 of the specification</i></p>
158/0x9E	Read Only	<p>Standard device type (MSB)</p> <p>This is part of the code that specifies a device that adopts to a common register standard. This is the type code represented by a 32-bit integer and defines the type belonging to a specific standard.</p> <p><i>Added in version 1.9.0 of the specification</i></p>
159/0x9F	Read Only	<p>Standard device type</p> <p><i>Added in version 1.9.0 of the specification</i></p>
160/0xA0	Read Only	<p>Standard device type</p> <p><i>Added in version 1.9.0 of the specification</i></p>
161/0xA1	Read Only	<p>Standard device type (LSB)</p> <p><i>Added in version 1.9.0 of the specification</i></p>
162/0xA2	Write Only	<p>Standard configuration should be restored for a unit if first 0x55 and then 0xAA is written to this location and is done so within one second.</p> <p><i>Added in version 1.10.0 of the specification</i></p>
163/0xA3-207/0xCF	—	Reserved for future use. Return zero.
208/0xD0-223/0xDF	Read Only	<p>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).</p>
224/0xE0-255/0xFF	Read Only	<p>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 0xFF. 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.</p>

Table 4: VSCP mandatory registers, part2

The registers are laid out in an 32-bit address space with the standard Level I register map on the top (0xFFFFFFF80 - 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.

7 Decision Matrix

7.1 Event handling and Rules

Every event 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 events. A level II node can handle both level I and level II events. In fact when we define rules we define them at the highest level.

7.2 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.

7.2.1 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.

7.2.2 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.

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-parameter

Table 5: Level I matrix row

7.2.3 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.

7.3 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

7.3.1 oaddr

oaddr is the originating address. We are only interested in events from the node given if it is used. For example we may be interested in events from 0x00 a 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.

7.3.2 flag

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

The zone and use sub-zone bits can be activated to have a check on the zone/sub-zone information of an event. That is the zone/sub-zone of the machine must match the one of the event to trigger the DM row. Bits 0/1 are the ninth bit of the filter/mask not enable bits.

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

Table 6: Flag bit description

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

Table 7: class/type filter

7.3.3 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

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 that are of interest.

- 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.

7.3.4 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.

7.3.5 action

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

7.3.6 action-parameter

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

7.3.7 General

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 (< 0x80) 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 0x7e the decision matrix is indexed. This means that 0x7e is the index and 0x7f is the data. Read a byte from the matrix by first setting the index to the position you want to read and reading the byte in 0x7f. Set index to the position you want to write and write data into 0x7f.

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

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.

7.4 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:

where the action-parameter has device specific length.

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

Table 8: Level II decision matrix format

7.4.1 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 events 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.

7.4.2 Filter (32-bit)

This is a bit mask, which tells the value for bits that is of interest. 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.

7.4.3 Control(32-bit)

bit #	Description
31	Enabled if set to 1. If disabled the decision matrix element is ignored.
30	Reserved.
29	Marks end of matrix. No more valid entries 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 sub-zone of this module to trigger DM entry. Byte 2 of data.
2	Reserved.
1	Reserved.
0	Reserved.

Level II decision matrix control word

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

7.4.4 Action (16-bit)

This is the what should be carried out to make this action happen. This is a 16-bit cod that is application specific. 0x0000 is the only code that is predefined as no operation.

7.4.5 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 0x80 - dm-row-size (a DM row aligned to the upper register border) it is implied that the register position just below, 0x80 -dm-row-size - 1, contains a register that is an index to the row that has its first byte at 0x80 - 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 0x80 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.

8 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.

8.1 Definitions for bits in byte 0

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

8.1.1 Bits 5,6,7

Bits	Description
5,6,7	Represent one of several numerical representations in which the data that follows can be represented as.

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 numerical string (possibly with language extensions (8-bit)). Max seven characters that together represent a number. Example “-123”, “1.3456”, “0.00001” etc

011b Integer Format(0x60) Data is coded as a signed 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 8-bit 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(0x80) Data is coded as a normalized integer. In this case the format byte is followed by the normalizer byte.

The normalizer byte is the exponent of the following integer and coded as ‘sign (bit 7) and magnitude (bits 0-6)’, representing an exponent in the range—($2^6 - 1$) to $2^6 - 1$. Thus bits 0-6 describe how many places the decimal point has to be shifted and bit 7 the direction of the shift (0 = right; 1 = left).

The actual integer (mantissa) 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 given as a two’s complement number.

0x02 0x1B 0x22
 0x1B22 = 6946 decimal
 0x02 has bit 7 cleared meaning the decimal point has to be shifted 2 steps to the left which is the same as multiplying the value by $10^2 = 100$

0x85 0x8D
 0x8D = -115 decimal
 0x85 has bit 7 set meaning the decimal point has to be shifted 5 steps to the left which is the same as dividing the value by $10^5 = 100000$

0x81 0x01 0x07
 0x0107 = 263 decimal
 0x81 has bit 7 set meaning the decimal point has to be shifted 1 step to the left which is the same as dividing the value by $10^1 = 10$

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.htm> and further info at 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.

8.1.2 Bits 3,4

Bits	Description
3,4	Indicates how the data should be interpreted. Typically this is a unit like Centigrade, Fahrenheit or Kelvin for a temperature value.

00b Standard format. All other codes in this field are event class/type specific.

8.1.3 Bits 0,1,2

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

9 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

9.1 Real life file examples

examples

- Kelvin NTC10K - http://www.eurosource.se/ntc10KA_2.xml
- Paris relay board - http://www.eurosource.se/paris_010.xml

9.2 XML Format Specification

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
“\n” can be used for a new line in text.

```
<?xml version = "1.0" encoding = "UTF-8" ?>  
<!-- Version 0.0.6 2012-10-01  
"string" -- Text string  
"bitfield" -- a field of bits. Width tells how many bits the field consist  
of. When read from a device the number of bits will always be  
in even octets with unused bits set to zero. Bitfield is  
taken from MSB part throue LSB and continues that way on  
next octet in the series.
```

"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 number. 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 number. 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
 "float" — Data is coded as a IEEE-754 1985 floating point value
 That is a total of 32-bits. The most significant byte is stored first.
 "double" — IEEE-754, 64 Bits, double precision.
 That is a total of 64-bits. The most significant byte is stored first.
 "date" — Must be passed in the format dd-mm-yyyy and mapped to "yy yy mm dd" that is four bytes, MSB->LSB
 "time" — Must be passed in the format hh:mm:ss where hh is 24 hour clock and mapped to "hh mm ss" MSB->LSB that is four bytes.
 "guid" — Holds the 16 bytes of a GUID. Stored on the form 11:22:33:... MSB->LSB

synonyms

"char"	— Is the same as "int8_t".
"byte"	— Is the same as "uint8_t".
"short"	— Is the same as "int16_t".
"integer"	— Is the same as "int16_t".
"long"	— Is the same as "int32_t".

index storage

This type of storage takes up two bytes in register space. The first byte is the index into the second.

index8_int16_t
 index8_uint16_t
 index8_int32_t
 index8_uint32_t
 index8_int64_t
 index8_uint64_t
 index8_float
 index8_double
 index8_date
 index8_time
 index8_guid
 index8_string — String stored as [width, string]. Width tells how long the strings are.
 If any of them are shorter then this value it should be zero terminated.

—>

<!— General section —>
<vscp>

<module> <!— one file can contain one or several modules —>

<name>aaaaaaaa</name>
 <model>bbbbbb</model>
 <version>cccccc</version>
 <description lang = "en">yyyyyyyyyyyyyyyyyyyyyyyy</description>

```

<!-- Site with info about the product -->
<infourl>http://www.somewhere.com</infourl>

<!-- Max package size a node can receive -->
<buffersize></buffersize>

<manufacturer>
    <name>tttttttttttttt</name>
    <address>
        <street>tttttttttt</street>
        <town>11111111</town>
        <city>London</city>
        <postcode>HH1234</postcode>
        <!-- Use region or state -->
        <state></state>
        <region></region>
        <country>tttt</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 lang="en">Main web address</description>
    </web>
</manufacturer>

<!-- Available Drivers
    [id] is a manufacturer defined ID for the driver
    [type] is "canal" or "vscp"
    [os] is "linux-generic", "mac-generic", "windows-generic",
          "windows-nt", "windows-xp", "windows-vista"
    [osver] is os version
-->
<driver id="xxx", type="yyy" os="zzz" osver="123">
    <description lang="en">Main email address</description>
    <location>Where the driver can be fetched from
        (one or many)</location>
</driver>

<!-- Picture of device -->
<picture path="url where picture can be found"
    format="bmp | jpg | png | ..... "
    height="height for picture in pixels"
    width="width of picture in pixels"
    size="size of picture file in bytes" >
    <description lang="en">Description of picture</description>
</picture>

<!-- Firmware for the device (can be several) -->
<firmware path="url where firmware can be found"

```

```

        format="intelhex8|intelhex16"
        size="Optional size in bytes for firmware file (not image)"
        date="ISO date year-month-day when released."
        version_major="x"
        version_minor="y"
        version_subminor="z">
        <description lang="en" >Firmware description</description>
    </firmware>

    <!-- Full documentation for the device (can be several) -->
    <manual path="url where manual can be found"
            lang="Two digit iso code for language"
            format="txt | rtf | doc | pdf | html">
        <description lang="en" >Firmware description</description>
    </manual>
<!--
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 is
a boolean the system knows it occupies one bit. Other types may
need "width" to be defined also. The variable 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
languages 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 (big-endian). The variable
can be read and written.
-->
<abstraction id="alsoaname" type="short" default="182"
              page = "0" offset = "15" indexed="false" >
    <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 the string is stored in
page=0 at offset=20-21. Read write access is possible
-->

<abstraction id="adescrivename" type="string" width="12" default=""
              page = "0" offset = "20" indexed="false" >
    <name lang="en">tttt</name>
    <description lang="en">yyy</description>

```

```

<help type="text/url" lang="en">ttt</help>
<access>rw</access>
</abstraction>
<!-- This example shows a value list stored in an integer. This is
     typically presented to the user as a list-box or a combo-box
     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">ttt</name>
    <description lang="en">yyy</description>
    <help type="text/url" lang="en">ttt</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>

<abstraction id="Calibrations" type="index8_int16_t"
    page = "0" offset = "116" size="6" >
    <name lang="en">Indexed array of values</name>
    <description lang="en">
        This is an array of six 16-bit integers. Register 116 is the index
        into the array which is at 117. So setting register 116 to 0 will
        put the MSB of the first integer into register 117 and so on.
    </description>
    <access>rw</access>
</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">abcdefgihh</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. Here 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>
<!-- example for bit groups -->
<bit pos="4" width="4">
  <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
itself 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 >

<!-- Decision matrix -->

```

```

<dmatrix>
  <help type="text/url" lang="en">tttt </help>
  <!-- Can currently be 1 or 2 -->
  <level>1</level>
  <!-- Where is matrix located -->
  <start page="0" offset="1" indexed="true|false"/>
  <!-- If the matrix is placed at location 126 (Level I) it will
      automatically be set to indexed -->
  <!-- # of rows in matrix -->
  <rowcnt>10</rowcnt>
  <!-- Size in bytes for one row - only for level II.
      Always 8 for Level I. Defaults to 8. -->
  <rowsize>8</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 (or receive) -->
<!-- Normally you only describe events the module is capable to send out -->
<!-- here. In this case direction="out" which is the default and -->
<!-- what is used if direction is not given. Sometimes some events can -->
<!-- have special meaning to the module. A typical is the CONTROL.SYNC event -->
<!-- if a module understands this the event can be described here with -->
<!-- direction="in". -->

<events>
  <event class="0" type="10" direction="in | out" >
    <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">yyyy </description>
    <help type="text/url" lang="en">tttt </help>
    <!-- Optional: What priority it will be sent as -->
    <priority>3</priority>
    <!-- Information about event 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 --&gt;
<!-- Description/specification for alarm bits --&gt;

&lt;alarm&gt;
    &lt;bit pos="1"&gt;
        &lt;name lang="en"&gt;tttt &lt;/name&gt;
        &lt;description lang="en"&gt;yyy&lt;/description&gt;
        &lt;help type="text/url" lang="en"&gt;tttt &lt;/help&gt;
    &lt;/bit&gt;
&lt;/alarm&gt;

<!-- bootloader information --&gt;

&lt;boot&gt;    &lt;!-- bootloader algorithm that can be used on this module --&gt;
    &lt;algorithm&gt;1&lt;/algorithm&gt;
    &lt;!-- Size of boot block/sector --&gt;
    &lt;blocksize&gt;20&lt;/blocksize&gt;
    &lt;!-- Number of available boot blocks/sectors --&gt;
    &lt;blockcount&gt;66&lt;/blockcount&gt;
&lt;/boot&gt;

&lt;/module&gt;
&lt;/vscp&gt;
</pre>

```

9.3 Creating a new MDF file

1. Add contact and company information.
2. Include name and a description about the module and pointers to full descriptions if available.
3. Add descriptions to registers if the module have extra registers besides the standard ones.
4. Add information about any boot loader supported.
5. Add abstractions for complex registers. A complex register can be two registers that together form a 16-bit number.
6. If the module can issue alarms add alarm section.
7. If the modules got a decision matrix add information about it.
8. If the module send out events or can send out events add information about them.

9.3.1 Relations between registers and abstractions

Registers are always eight bits. This is the only way a VSCP unit can be interfaced to. Everything it exports of its black box functionality must be broken down into eight bit registers. The common denominator between devices in short.

For user applications this may be inconvenient as a higher level application wants to look at parameters in a smarter way. A string should be a string instead of a sequence of registers. An integer a numerical value instead of two consecutive registers with a high and a low part. To overcome this abstractions can be defined in the MDF file that tells which registers make up a string and which register makes up an integer. This can then be presented to the user and the software can handle the actual register abstraction model.

Lets look at an example. The reference model for the Ethernet based Nova module have a protection timer. On the unit this timer takes up two consecutive registers for each output channel it protects. The first byte is as it should be the most significant byte of the timer and the second byte is the least significant byte. Thus the actual timer value is byte0 * 256 + byte1. In the MDF file for Nova this is written as

```
<reg page="0" offset="26" >
    <name lang="en">Output protection timer 0 MSB</name>
    <description lang="en">
        This is the most significant byte for the output protection time
        An output will be inactivated if not written to before this time
        has elapsed.
        Set to zero to disable (default). The max time is 65535 seconds
        which is about 18 hours.
        The registers can be as an example be used as a security feature
        to ensure that an output
        is deactivated after a preset time even if the controlling device
        failed to deactivate the relay.
    </description>
    <access>rw</access>
</reg>

<reg page="0" offset="27" >
    <name lang="en">Output protection timer 0 LSB</name>
    <description lang="en">
        This is the least significant byte for the output protection time
        An output will be inactivated if not written to before this time
        has elapsed.
        Set to zero to disable (default). The max time is 65535 seconds
        which is about 18 hours.
        The registers can be as an example be used as a security feature
        to ensure that an output
```

```

        is deactivated after a preset time even if the controlling device
        failed to deactivate the relay.
    </description>
    <access>rw</access>
</reg>
```

As seen the register at position 26 and 27 is used. Both on page 0. A user that gets this information presented for him/her needs to do some calculations to actually set the value. To make it possible to preset this to a user in a more user friendly way and abstraction is defined.

```

<abstraction id="Protectionontimer0" type="uint16_t" default="0" page = "0" offset="26">
    <name lang="en">Output protection timer 0</name>
    <description lang="en">
        This is the least significant byte for the output protection timer.
        An output will be inactivated if not written to before this time has elapsed.\n
        Set to zero to disable (default). The max time is 65535 seconds which is about 18 hours.
        The registers can be as an example be used as a security feature to ensure that an output
        is deactivated after a preset time even if the controlling device failed to deactivate the relay.
    </description>
    <help type="url" lang="en">
        http://www.vscp.org/wiki/doku.php/modules/nova#output_protection
    </help>
    <access>rw</access>
</abstraction>
```

Now the two registers instead is presented as an unsigned 16 bit integer in a way a user expect it to be. He/she just set the value in seconds for the protection timer and the control system knowing that an unsigned integer needs two bytes can write or read the value from the register pair 26/27.

Also here an URL pointing to formatted help information is set. This URL could have been set for the registers as well of course.

User software first try to present information to users using the definitions in the abstraction section and only use registers if no abstraction covers that register.

10 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 event 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 event (a NACK boot loader mode event is also possible). Block size and number of blocks are sent as arguments in the acknowledge event.
3. The master sends a start block data transfer event 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 events until the complete block is transferred.
5. When all data for a block is received by a node it sends a confirm block event to acknowledge the reception of a complete block.
6. The master now sends a program block event 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 event.
8. The next block is handled or the node is taken out of the boot loader mode by sending a drop nickname/reset device event.
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 boot-loader 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.

10.1 Microchip custom algorithm for PIC18 devices, Boot loader code = 0x01

rev 0.3: 2004-09-22 The CAN boot loader is a modified version of a boot loader that Microchip describes in there application note AN247. Source for the

boot loader is available at `/firmware/pic/bootloader/Microchip/firmware` in the source tree.

10.1.1 Changes from Appnote 247

This boot loader is based on the Microchip CAN boot loader for 18F devices that is described in their application note 247 (see references at the end of this document for a pointer to original code and excellent documentation). This work remains much the same but some changes have been made to it to make it suitable for VSCP nodes.

1. Message format has been changed to suit VSCP protocol.
2. Messages from the node uses the node-ID in the origin field.
3. The node sends an initial message at start-up.
4. The boot flag is in EEPROM byte 0 as specified in the VSCP specification.

The `vscpboot` application is written in C++ using the wxWidgets library making it usable on multiple platforms (Windows, Unix and in the future Macintosh) The bootloader firmware Entering bootloader mode on a VSCP node

To enter boot loader mode a `class=0, Type=12, Enter boot loader mode` should be sent to the node. The following data should be supplied

- Byte 0: The nickname-ID for the node.
- Byte 1: 0x01 – Bootloader for Microchip devices type 1 (this bootloader).
- Byte 2: GUID byte 0 for node (register 0xDF).
- Byte 3: GUID byte 3 for node (register 0xDC).
- Byte 4: GUID byte 5 for node (register 0xDA).
- Byte 5: GUID byte 7 for node (register 0xD7).
- Byte 6: Content of node register 0x92, Page Select Register MSB
- Byte 7: Content of node register 0x93, Page Select Register LSB

If the supplied data is correct and this bootloader type is supported on the device, a message `class=0, type=13, ACK Boot Loader Mode` will be sent from the node.

If something is wrong a `class=0, type=14, NACK Boot Loader Mode` will be sent.

To indicate that the node is in boot loader mode. It will send a message `0x15nn` where nn is its node-ID when it has entered the boot loader. The application should wait for this initial message before it starts its work.

Register 0x97 contains info (the code) about which bootloader algorithm a node support as documented in the VSCP specification for class=0, type=12..

10.1.2 Loading code to a node.

1. Send a PUT_BASE_INFO command. The address pointer should be set. The control byte is typically set to 0x1D (MODE_WRT_UNLOCK, MODE_AUTO_ERASE, MODE_AUTO_INC, MODE_ACK). If CMD_RESET is set the checksum will be tested. This is usually how the first command to the node is formatted.
2. Write data to the device by sending multiple PUT_DATA commands to it. Each byte sent to the node must be summed in order to cater for the checksum.
3. Check that the written data is OK. This can be done by reading the data back or by checking the checksum. If a PUT_BASE_INFO command is sent with CMD_CHK_RUN and with CMD_DATA_LOW/CMD_DATA_HIGH set to (0x10000 - (checksum & 0xFFFF)) the node will check its own calculated checksum and start the application on the node after setting the EEPROM boot info to start-application state. If the checksum fails it will stay in bootloader mode.

The VSCP bootloader wizard handles this task.

10.1.3 Filter and Mask settings

If the filter allows messages for all nicknames to come in to the node it is possible to boot load many nodes at the same time. Each node will answer with its node-ID but will only react to packages for node 0xFF and its own ID. In this way it is possible to have part of the programming common for all nodes.

accept only class=0, type=16,17,18,19, origin=all

Mask

- EIDL = 0x00 All origins
- EIDH = 0xFC
- SIDL = 0xFF
- SIDH = 0x0f All priorities, either hard coded or not.

Filter

- EIDL = 0x00
- EIDH = 0x10
- SIDL = 0x08 only extended messages
- SIDH = 0x00

Responses are of type 20 and 21.

10.1.4 PUT_BASE_INFO

- Priority = 0
- Hard code = 0
- Class = 0
- Type = 16

Command ID is 0x000010nn where nn is the nickname for the node that initiated the boot load.

The PUT_BASE_INFO command sets address and flags for a device.

- Byte 0: ADDR_LOW - Low address bits 0-7
- Byte 1: ADDR_HIGH - High address bits 8-15
- Byte 2: ADDR_UPPER - Upper address bits 16 - 23
- Byte 3: reserved
- Byte 4: CONTROL - Control byte
- Byte 5: COMMAND - Command
- Byte 6: CMD_DATA_LOW - Command data 0-7
- Byte 7: CMD_DATA_HIGH - Command data 8-15

CONTROL is defined as follows

- Bit 0: MODE_WRT_UNLCK Set this to allow memory write and erase.
- Bit 1: MODE_ERASE_ONLY Set this to only erase program memory on a put command. Must be on a 64-bit boundary.
- Bit 2: MODE_AUTO_ERASE Set this to automatically erase program memory while writing data.
- Bit 3: MODE_AUTO_INC Set this to automatically increment the pointer after a write.
- Bit 4: MODE_ACK Set to get acknowledge.
- Bit 5: undefined.
- Bit 6: undefined.
- Bit 7: undefined.

COMMAND is defined as follows

- 0x00: CMD_NOP No operation – Do nothing
- 0x01: CMD_RESET Reset the device.
- 0x02: CMD_RST_CHKSM Reset the checksum counter and verify.
- 0x03: CMD_CHK_RUN Add checksum to CMD_DATA_LOW and CMD_DATA_HIGH, if verify and zero checksum then clear the last location of EEDATA.

Send this command first with the base address and MODE_WRT_UNLCK to be able to write to memory.

If MODE_AUTO_ERASE is set the memory pointer will auto increment after each write.

If MODE_ACK is set then ack messages will be sent from the node after the write. Response messages has the form 0x000014nn where nn is node nickname-ID.

To only erase but not write set MODE_ERASE_ONLY

if MODE_AUTO_INC is set, the memory pointer will increment automatically.

10.1.5 PUT_DATA

- Priority = 0
- Hardcode = 0
- Class = 0
- Type = 17

Command ID is 0x000011nn where nn is the nickname for the node that initiated the boot load.

This command writes data to memory.

The PUT_DATA command sets address and flags for a device.

- Byte 0: D0 - Data byte 0
- Byte 1: D1 - Data byte 1
- Byte 2: D2 - Data byte 2
- Byte 3: D3 - Data byte 3
- Byte 4: D4 - Data byte 4
- Byte 5: D5 - Data byte 5
- Byte 6: D6 - Data byte 6
- Byte 7: D7 - Data byte 7

If MODE_ACK is set then ack messages will be sent from the node after the write. Response messages has the form 0x000015nn where nn is node nickname-ID.

If MODE_AUTO_INC is set the memory pointer will increase automatically and one can issue multiple PUT_DATA after each other until the flash is written.

10.1.6 GET_BASE_INFO

- Priority = 0
- Hardcode = 0
- Class = 0
- Type = 18

Command ID is 0x000012nn where nn is the nickname for the node that initiated the boot load.

This command gets the base info from the node. There is no data for this command.

The response is:

- Byte 0: ADDR_LOW - Low address bits 0-7
- Byte 1: ADDR_HIGH - High address bits 8-15
- Byte 2: ADDR_UPPER - Upper address bits 16 - 23 Byte 3: reserved
- Byte 4: CONTROL - Control byte Byte 5: reserved
- Byte 6: reserved
- Byte 7: reserved

with ID 0x000010nn where nn is nickname.

10.1.7 GET_DATA

- Priority = 0
- Hardcode = 0
- Class = 0
- Type = 19

Command ID is 0x000013nn where nn is the nickname for the node that initiated the boot load.

This command gets data at the pointer from the node. There is no data for this command.

The response is:

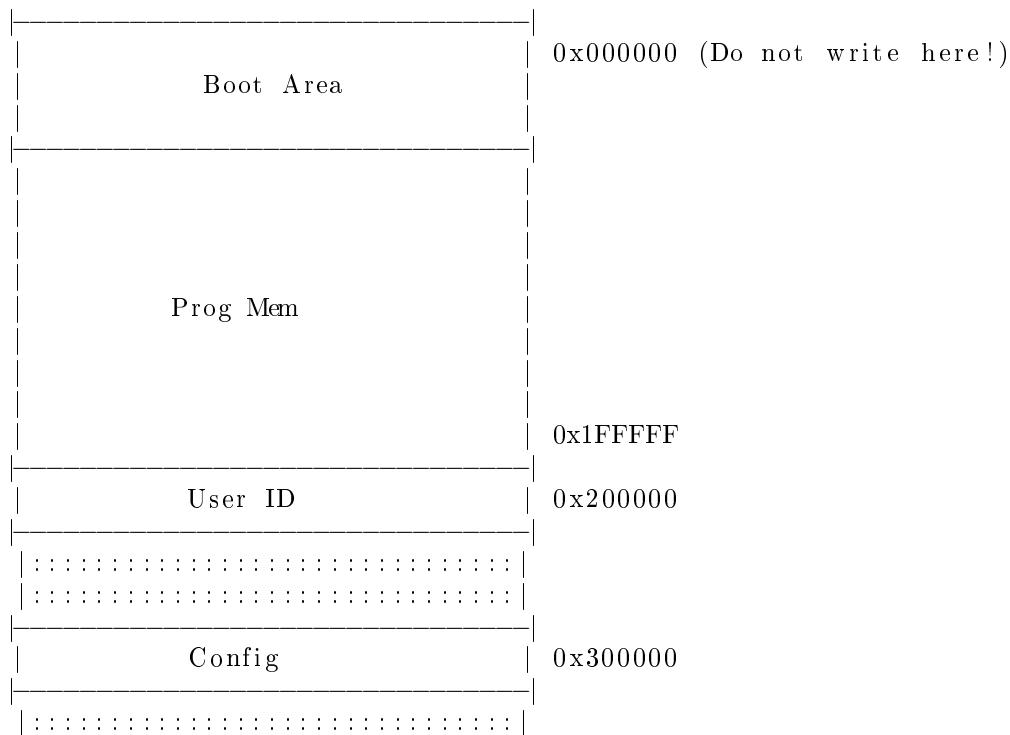
- Byte 0: D0 - Data byte 0
- Byte 1: D1 - Data byte 1
- Byte 2: D2 - Data byte 2
- Byte 3: D3 - Data byte 3
- Byte 4: D4 - Data byte 4
- Byte 5: D5 - Data byte 5
- Byte 6: D6 - Data byte 6
- Byte 7: D7 - Data byte 7

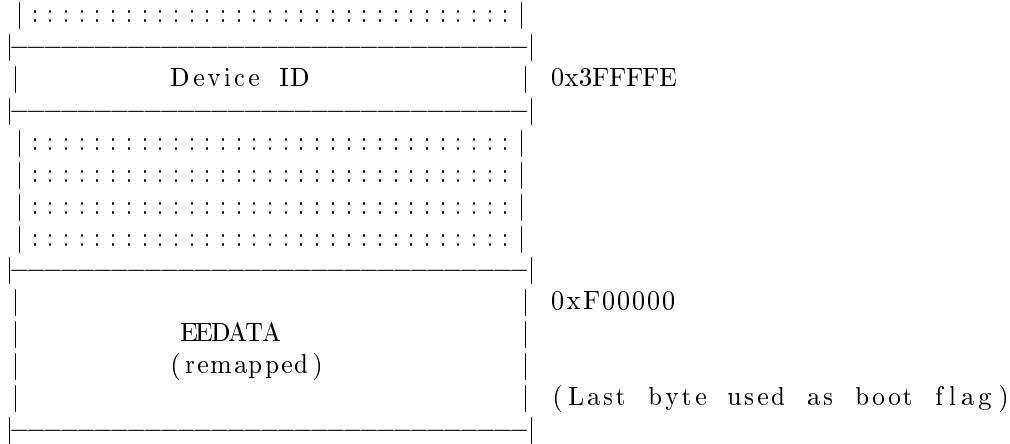
The ID has the form 0x000011nn where nn is node nickname-ID.

If MODE_AUTO_INC is set the memory pointer will increase automatically and one can issue multiple PUT_DATA after each other until the flash is written.

A node that implements the bootloader but does not want to share memory content can report all data as 0xFF.

Memory Organization





10.1.8 References

The Microchip application note AN 247 is the base work for this application. It and its accompanying source can be found at http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_

11 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> and <http://www.tldp.org/HOWTO/Multicast-HOWTO.html>

12 Level I Events

A class in Level I is described by a number between 0-511. Instead of writing a number the class can be described as CLASS1.XXXX indicate a specific (XXXX in this case) Level I class. Also the form CLASS1.XXXX=yy can be used where yy is the numerical form.

Class definitions can be found in the header file *vscp_class.h* which is located located in the *src/vscp/common* folder. In the same folder *vscp_type-h* can be found which contains defines for types. Also the *vscphelper.h/cpp* files contains stuff that are useful for handling classes/types as a programmer.

12.1 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. This means that for all Level II class=512 events the data described here is preceded by the 16-bit destination GUID (address of receiver) stored with MSB first followed by the data specified here. Nickname is present also if documented below but have no meaning and should be discarded.

12.1.1 Type = 0 (0x00) Undefined.

Undefined protocol function.

12.1.2 Type = 1 (0x01) Segment Controller Heartbeat.

Not mandatory. Implement in device if needed by application.

A segment controller sends this event 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 could respond with a CLASS1.INFORMATION, Type=9 event (HEARTBEAT) and should do so if it does not send out a regular heartbeat 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 (HEARTBEAT)

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 its 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 event is not available on all segments and is not mandatory.

Byte0 8-bit CRC of the segment controller GUID.

Byte1 MSB of time since epoch (optional).

Byte2 Time since epoch (optional).

Byte3 Time since epoch (optional).

Byte4 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 controller but use a nickname other than zero.

12.1.3 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 event before they go on line to do their “day to day” work.

Normally all nodes should save their assigned nickname-ID in non-volatile 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 on-line event 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.

If a node send this event with the unassigned ID 0xFF and byte 0 set to 0xFF it has given up the search for a free ID.

It is recommended that also level II nodes send this event when they come alive. In this case the 16-byte data is the GUID of the node.

Byte0 Target address - If specified this is a probe event 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.

12.1.4 Type = 3 (0x03) Probe ACK.

Mandatory. Must be implemented by all devices.

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

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

Reserved.

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

Reserved.

12.1.7 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. The node just uses the new nickname and don't start nickname discovery or similar.

Byte0 Old nickname for node.

Byte1 The new nickname for the node.

12.1.8 Type = 7 (0x07) nickname-ID accepted.

Mandatory. Must be implemented by all devices.

A node sends this event to confirm that it accepts its assigned nickname-ID. When sending this event the node uses its newly assigned nickname address.

12.1.9 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.

Byte0 The current nickname for the node.

Byte1 *Optional:* Flags.

- Bit 0: Reserved.
- Bit 1: Reserved.
- Bit 2: Reserved.
- Bit 3: Reserved.
- Bit 4: Reserved.
- Bit 5: Reset device. Keep nickname.
- Bit 6: Set persistent storage to default.

- Bit 7: Go idle. Do not start up again.

Byte2 *Optional:* Time the node should wait before it starts a nickname discovery or starts the device. The time is in seconds.

So if byte 1 and 2 is not in event restart device, set default parameters and do a nickname discovery. If byte 1 and 2 are present and bit 5 is set load defaults into device, restart but keep nickname. In all cases byte 2 delays before the node is restarted.

1. With just one byte as an argument. The node should do a standard node discovery in the same way as if the status button of the node is pressed. Preserve initiated data,
2. If byte 1 is present bit 5: Just restart. Don't change any data. not even nickname. bit 6: Restart write default to persistent storage. bit 7: die die my darling. If both bit 5 and 6 is set, do 5 and then 6 == 6 or do 6 and then 5 == 6
3. Byte 1 + byte 2 Wait this amount of seconds after the above operation has been carried out.

There is a variant of this where the GUID is used instead of the nickname to identify the device, Class = 0, Type = 23.

12.1.10 Type = 9 (0x09) Read register.

Mandatory. Must be implemented by all devices.

Read a register from a node.

Byte0 Node address.

Byte1 Register to read.

A read/write response event 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.

Byte0-15 GUID (MSB -> LSB).

Byte16 Reserved.

Byte17 Register to read.

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

Mandatory. Must be implemented by all devices.

Response for a read/write event. . Note that the data is returned for both a read and a write and can and probably should be checked for validity.

Byte0 Register read/written.

Byte1 Content of register.

12.1.12 Type = 11 (0x0B) Write register.

Mandatory. Must be implemented by all devices.

Write register content to a node.

Byte0 Node address.

Byte1 Register to write.

Byte2 Content for register.

A read/write response event 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.

Byte0-15 GUID (MSB -> LSB).

Byte16 Reserved.

Byte17 Register to write.

Byte18 Content of register.

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

Mandatory. Must be implemented by all devices.

Send NACK (Class=0,Type=14 if no boot-loader implemented)

This is the first event in the boot loader sequence. The node should stop all other activities when in boot loader mode. This also means that the node should not react on other events (commands) then the boot loader related.

Byte0 The nickname for the node.

Byte1 Code that select boot loader algorithm to use.

Byte2 GUID byte 0

Byte3 GUID byte 3

Byte4 GUID byte 5

Byte5 GUID byte 7

Byte6 Content of register 0x92, Page select MSB.

Byte7 Content of register 0x93, 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.

Byte0-15 GUID.

Byte16 Boot-loader algorithm code.

Boot-loader Codes

- 0x00 VSCP algorithm.
- 0x01 Microchip PIC algorithm
- 0x10 Atmel AVR algorithm 0
- 0x20 NXP ARM algorithm 0
- 0x30 ST ARM algorithm 0
- All other reserved.

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

Not mandatory. Only needed if a VSCP boot-loader algorithm is used.

This event 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.

Byte0 MSB of flash block size.

Byte1 Flash block size.

Byte2 Flash block size.

Byte3 LSB of flash block size.

Byte4 MSB of number of block s available.

Byte5 Number of block s available.

Byte6 Number of block s available.

Byte7 LSB of number of blocks available.

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

Mandatory. Should be implemented by all devices.

The node was unable to enter boot loader mode. The reason is given by a user specified error code byte. This event has no meaning for any node that is not in boot mode and should be disregarded.

Byte0 Optional user defined error code.

12.1.16 Type = 15 (0x0F) Start block data transfer.

Not mandatory. Only needed if a VSCP boot-loader algorithm is used.

Begin transfer of data for a block of memory. This event has no meaning for any node that is not in boot mode and should be disregarded.

Byte0 MSB of block number.

Byte1 Block number.

Byte2 Block number.

Byte3 LSB of block number.

Byte4 (optional) Type of Memory we want to write

0	or byte absent = PROGRAM Flash (status quo for old nodes)
1	DATA (EEPROM, MRAM, FRAM)
2	CONFIG (Fuses, CPU configuration)
3	RAM
4...255	Currently undefined - send a NACK as response

Byte5 (optional) Bank/Image to be written Used together with byte 4 to specify either separate Flash or EEPROM/MRAM spaces.

0	or absent = normally, means first memory from the view of the node creator, e.g. internal Flash, internal EEPROM etc. Useful for projects that have internal as well as external EEPROMs so the external one could be addressed with byte5=1. Also with byte4=0 and byte5=1 an SD-Card as well as a second firmware image inside the flash could be addressed.
---	--

Response can be Class=0, Type=50 (Start block data transfer ACK) or Class=0, Type=51(Start block data transfer NACK).

12.1.17 Type = 16 (0x10) Block data.

Not mandatory. Only needed if a VSCP boot-loader algorithm is used.

Data for a block of memory. This event has no meaning for any node that is not in boot mode and should be disregarded.

Byte0 Data.

Byte1 Data.

Byte2 Data.

Byte3 Data.

Byte4 Data.

Byte5 Data.

Byte6 Data.

Byte7 Data.

12.1.18 Type = 17 (0x11) ACK data block.

Not mandatory. Only needed if a VSCP boot-loader algorithm is used.

Confirm the reception of a complete data block. This event has no meaning for any node that is not in boot mode and should be disregarded.

Byte0 MSB of 16-bit CRC for block.

Byte1 LSB for 16-bit CRC for block.

Byte2 MSB of write pointer.

Byte3 write pointer.

Byte4 write pointer.

Byte5 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.

12.1.19 Type = 18 (0x12) NACK data block.

Not mandatory. Only needed if a VSCP boot-loader algorithm is used.

NACK the reception of data block. This event has no meaning for any node that is not in boot mode and should be disregarded.

Byte0 User defined error code.

Byte1 MSB of write pointer.

Byte2 write pointer.

Byte3 write pointer.

Byte4 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.

12.1.20 Type = 19 (0x13) Program data block

Not mandatory. Only needed if a VSCP boot-loader algorithm is used.

Request from a node to program a data block that has been uploaded and confirmed. This event has no meaning for any node that is not in boot mode and should be disregarded.

Byte0 MSB of block number.

Byte1 Block number.

Byte2 Block number.

Byte3 LSB of block number.

12.1.21 Type = 20 (0x14) ACK program data block

Not mandatory. Only needed if a VSCP boot-loader algorithm is used.

A node confirms the successful programming of a block. This event has no meaning for any node that is not in boot mode and should be disregarded.

Byte0 MSB of block number.

Byte1 Block number.

Byte2 Block number.

Byte3 LSB of block number.

12.1.22 Type = 21 (0x15) NACK program data block

Not mandatory. Only needed if a VSCP boot-loader algorithm is used.

A node failed to program a data block. This event has no meaning for any node that is not in boot mode and should be disregarded.

Byte0 User defined error code.

Byte1 MSB of block number.

Byte2 Block number.

Byte3 Block number.

Byte4 LSB of block number.

12.1.23 Type = 22 (0x16) Activate new image

Not mandatory. Only needed if a VSCP boot-loader algorithm is used.

This command is sent as the last command during the boot-loader sequence. It resets the device and starts it up using the newly loaded code. The 16-bit CRC for the entire program block is sent as an argument. This must be correct for the reset/activation to be performed. NACK boot loader mode will be sent if the CRC is not correct and the node will not leave boot loader mode.

Byte0 16 bit CRC of full flash data block, MSB

Byte1 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. This event has no meaning for any node that is not in boot mode and should be disregarded. Response can be Class=0, type=48 (Activate new image ACK) or Class=0, Type=49(Activate new image NACK).

12.1.24 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 Class=0, Type=8 but here the full GUID is used instead of the nickname to identify the node that should drop its current nickname and enter the node-name discovery procedure.

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

Byte0 index.

Byte1 GUID byte.

Byte2 GUID byte.

Byte3 GUID byte.

Byte4 GUID byte.

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

Index = Byte 0	Byte 1	Byte 2	Byte 3	Byte 4
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 than one node should be initialized. This event can be used to assign nickname-IDs 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.

12.1.25 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. 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 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.

Byte0 Node-ID which registers should be read.

Byte1 Index into page.

Byte2 Number of bytes to read.

Response is

Class=0, Type=26 (0x1A) Read page response.

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.

Byte0-15 GUID.

Byte16 Index into page.

Byte17 Number of bytes to read.

12.1.26 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. One data-space positions is reserved to select a base into this storage. See Page read for a full description.

It is only possible to write one 6-byte chunk 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.

Byte0 Node-ID

Byte1 Register start.

Byte2-7 Data.

Response is

Class=0, Type = 26 (0x1A) Read Page Response.

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.

Byte0-15 GUID.

Byte16 Base index.

Byte18-... Data.

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

12.1.27 Type = 26 (0x1A) Read/Write page response

Mandatory. Should be implemented by all devices.

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

Byte0 Sequence number.

Byte1-7 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.

Byte0-15 GUID.

Byte16 Sequence number.

Byte17-... Data.

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

12.1.28 Type = 27 (0x1B) High end server probe

Should be implemented by all devices that work over 802.15.4/Ethernet/Internet or other high end protocols.

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

Byte0 Code for node type.

Byte1-7 Meaning depends on code in byte 0.

Code node type byte

Code Description 0 this is a TCP/IP node, byte 1-4 is the V4 IP address of the node. Byte 5 - if available - tells the capabilities of this node.

Capabilities byte (byte 5)

Code Description Bit 0 Node will log in to server using text mode. Bit 1 Node will log in to server using binary mode. Bit 2 Node expect server to connect using text mode. Bit 3 Node expect server to connect using binary mode. Bit 4-7 Reserved. Bit 0 + Bit 1 can be set at the same time indicating that the node can handle both modes. Bit 2 + bit 3 can be set at the same time indicating that the node can handle both modes.

The VSCP daemon documentation have a description on how server discovery works 14.5.2

12.1.29 Type = 28 (0x1C) High end server response

Should be implemented by all devices that work on 802.15.4/Ethernet/Internet.

Byte0 Code for server capabilities MSB

Byte1 Code for server capabilities LSB

Byte2 Server IP address MSB - or other relevant data as of server capabilities (Network byte order)

Byte3 Server IP address - or other relevant data as of server capabilities (Network byte order)

Byte4 Server IP address - or other relevant data as of server capabilities (Network byte order)

Byte5 Server IP address LSB - or other relevant data as of server capabilities (Network byte order)

Byte6 Server Port MSB - or other relevant data as of server capabilities Byte 7 Server Port LSB - or other relevant data as of server capabilities

Description of bit-usage

- Bit 15 - VSCP TCP server - data is V4 IP address and port
- Bit 14 - Reserved
- Bit 13 - Reserved
- Bit 12 - Reserved
- Bit 11 - VSCP UDP server - data is V4 IP address and port

- Bit 10 - Reserved
- Bit 9 - Reserved
- Bit 8 - VSCP TCP SSL secure server link - data is V4 IP address and port
- Bit 7 - Reserved
- Bit 6 - Reserved
- Bit 5 - Reserved
- Bit 4 - Reserved
- Bit 3 - Accepts two or more simultaneous connections.
- Bit 2 - Implements RCVLOOP command.
- Bit 1 - Text interface connection available.
- Bit 0 - Reserved.

A node that needs 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.

The VSCP daemon documentation have a description on how server discovery works 14.5.2

12.1.30 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

Byte0 Node-ID

Byte1 Register to increment.

Node should answer with Read/Write register response. Class=0, Type=10

12.1.31 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

Byte0 Node-ID

Byte1 Register to decrement.

Node should answer with Read/Write register response. Class=0, Type=10

12.1.32 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.

Byte0 Node-ID or 0xFF for all nodes.

Response is Class = 0, Type = 32.

12.1.33 Type = 32 (0x20) Who is there response

Mandatory. Must be implemented by all devices.

Response from node(s) looks like this

byte 0	byte 1	byte 2	byte 3	byte 4	byte 5	byte 6	byte 7
0	GUID15	GUID14	GUID13	GUID12	GUID11	GUID10	GUID9
1	GUID8	GUID7	GUID6	GUID5	GUID4	GUID3	GUID2
2	GUID1	GUID0	MDF0	MDF1	MDF2	MDF3	MDF4
3	MDF5	MDF6	MDF7	MD8	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

All seven frames should be sent also if the MDF URL is shorter than 32 characters,

12.1.34 Type = 33 (0x21) Get decision matrix info

Mandatory

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

Byte0 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.

Byte0-15 GUID.

A node that does not have a decision matrix should return zero rows.

12.1.35 Type = 34 (0x22) Decision matrix info response

Mandatory for nodes with 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 0x00 (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 but it's better if it returns a decision matrix size of zero.

Byte0 Matrix size (number of rows). Zero for a device with no decision matrix.

Byte1 Offset in register space.

Byte2 Optional page start MSB (Interpret as zero if not sent)

Byte3 Optional page start LSB (Interpret as zero if not sent)

Byte4 Optional page end MSB (Interpret as zero if not sent) Deprecated. Set to zero.

Byte5 Optional page end LSB (Interpret as zero if not sent) Deprecated. Set to zero.

Byte6 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. The decision matrix can also be stored indexed. In that case the first byte is the index and the second is the data. If the index is in location 0x7f then an indexed matrix is assumed.

Register position	Description
0x77	Index for row in decision matrix.
0x78-0x7F	Level I decision matrix row.

12.1.36 Type = 35 (0x23) Get embedded MDF.

Optional.

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

Byte0 Node-ID.

12.1.37 Type = 36 (0x24) Embedded MDF response.

Optional. See Type=35

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.

If an embedded MDF is not available a response on the form

byte 0 = 0 byte 1 = 0 byte 2 = 0

should be sent.

Byte0 High byte of MDF description index.

Byte1 Low byte of MDF description index.

Byte2-7 MDF data.

12.1.38 Type = 37 (0x25) Extended page read register.

Mandatory. Must be implemented by all devices.

Read a register from a node with page information.

Implementation must take care so all page register change done by these routines must restore the content of the page registers to their original content when they are done.

Byte0 Node address.

Byte1 MSB of page where the register is located.

Byte2 LSB of page where the register is located.

Byte3 Register to read (offset into page).

Byte4 Optional: Number of registers to read.

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

This means that a register (or a maximum of four 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.

Byte0-15 GUID.

Byte16 MSB of page where the register is located.

Byte17 LSB of page where the register is located.

Byte18 Register to read.

Byte19 Optional: bytes to read (1-255).

12.1.39 Type = 38 (0x26) Extended page write register.

Mandatory. Must be implemented by all devices.

Write register content to a node.

Implementation must take care so all page register change done by these routines must restore the content of the page registers to their original content when they are done.

Byte0 Node address.

Byte1 MSB of page where the register is located.

Byte2 LSB of page where the register is located.

Byte3 Register to write.

Byte4 Content for register.

Byte5,6,7 Optional extra data bytes to write.

A read/write response event 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.

Byte0-15 GUID.

Byte16 MSB of page where the register is located.

Byte17 LSB of page where the register is located.

Byte18 Register to write.

Byte19 Content of register. byte 20-buffer-size Optional extra data bytes to write.

12.1.40 Type = 39 (0x27) Extended page read/write response

Mandatory. Must be implemented by all devices.

This is the replay sent for events CLASS1.PROTOCOL, Type=40,41.

Byte0 Index (starts at zero).

Byte1 MSB of page where the register is located.

Byte2 LSB of page where the register is located.

Byte3 Register read/written.

Byte4 Content of register.

Byte5-7 Content of register if multi register read/write.

A multi register read/write can generate many events of this type. Index will then be increased by one for each event sent.

12.1.41 Type = 40 (0x28) Get event interest

Optional. Implemented if needed.

It is possible to ask a node which event(s) it is interested in with this event. If not implemented the node is supposed to be interested in all events.

All nodes are by default interested in CLASS1.PROTOCOL.

The event is intended for very low bandwidth nodes like low power wireless nodes where it saves a lot of bandwidth if only events that really concerns the node is sent to it.

12.1.42 Type = 41 (0x29) Get event interest response

Optional. Implemented if needed.

Response for event CLASS1.PROTOCOL, Type=40. The node report all events it is interested in.

Byte0 Index

Byte1 class bit 9

- bit 0 - Bit 9 for class 1
- bit 1 - Bit 9 for class 2
- bit 2 - Bit 9 for class 3
- bit 3 - All Type 1 is recognized (set type to zero).
- bit 4 - All Type 2 is recognized (set type to zero).
- bit 5 - All Type 3 is recognized (set type to zero).
- bit 6 - 0
- bit 7 - 0

Byte2 class 1

Byte3 type 1

Byte4 class 2

Byte5 type 2

Byte6 class 3

Byte7 type 3

A node that is interested in everything send just a CLASS1.PROTOCOL, Type=41 with no data.

Nodes that want to specify events of interest fill them in. If all types of a class should be recognized set the corresponding bit in byte 1 and the related type to zero.

A maximum of 255 frames (index = 0-254) may be sent.

Fill unused pairs with zero.

12.1.43 Type = 48 (0x30) Activate new image ACK.

Not mandatory. Only needed if a VSCP boot-loader algorithm is used.

Part of the VSCP boot-loader functionality. This is the positive response after a node received a CLASS1.PROTOCOL, Type=22, Activate new image. It is sent by the node before the new firmware is booted into.

12.1.44 Type = 49 (0x31) Activate new image NACK.

Not mandatory. Only needed if a VSCP boot-loader algorithm is used.

Part of the VSCP boot-loader functionality. This is the negative response after a node received a CLASS1.PROTOCOL, Type=22, Activate new image. It is sent by the node to inform it that it will (or can not) switch to the new firmware image.

12.1.45 Type = 50 (0x32) Start block data transfer ACK.

Not mandatory. Only needed if a VSCP boot loader algorithm is used.

Part of the VSCP boot-loader functionality. This is the positive response after a node received a CLASS1.PROTOCOL, Type=15, Start block data transfer. It is sent by the node as a validation that it can handle the block transfer.

12.1.46 Type = 51 (0x33) Start block data transfer NACK.

Not mandatory. Only needed if a VSCP boot-loader algorithm is used.

Part of the VSCP boot-loader functionality. This is the negative response after a node received a CLASS1.PROTOCOL, Type=15, Start block data transfer. It is sent by the node as an indication that it can NOT handle the block transfer.

12.2 Class=1 (0x01) Alarm

CLASS1.ALARM

Description

Alarm events that indicate that something not ordinary has occurred. Note that the priority bits can be used as a mean to level alarm for severity.

12.2.1 Type = 0 (0x00) Undefined

Undefined alarm.

12.2.2 Type = 1 (0x01) Warning .

Indicates a warning condition.

Byte0 User defined data.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 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.

12.2.3 Type = 2 (0x02) Alarm occurred.

Indicates an alarm condition.

Byte0 User defined data.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 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.

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

Alarm sound should be turned on or off.

Byte0 0=off. 1=on.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 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.

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

Alarm light should be turned on or off.

Byte0 User defined data.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 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.

12.2.6 Type = 5 (0x05) Power on/off

Power has been lost or is available again.

Byte0 User defined data.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 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.

12.2.7 Type = 6 (0x06) Emergency Stop

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

Byte0 User defined data.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 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.

12.2.8 Type = 7 (0x07) Emergency Pause

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

Byte0 User defined data.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 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.

12.2.9 Type = 8 (0x08) Emergency Reset

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

Byte0 User defined data.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 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.

12.2.10 Type = 9 (0x09) Emergency Resume

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

Byte0 User defined data.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 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.

12.3 Class=2 (0x02) Security

CLASS1.SECURITY

Description

Security related events for alarms and similar devices.

12.3.1 Type = 0 (0x00) undefined

Undefined security issue.

12.3.2 Type = 1 (0x01) Motion Detect

A motion has been detected.

Byte0 User defined data.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 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.

12.3.3 Type = 2 (0x02) Glass break

A glass break event has been detected.

Byte0 User defined data.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 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.

12.3.4 Type = 3 (0x03) Beam break

A beam break event has been detected.

Byte0 User defined data.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 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.

12.3.5 Type = 4 (0x04) Sensor tamper

A sensor tamper has been detected.

Byte0 User defined data.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 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.

12.3.6 Type = 5 (0x05) Shock sensor

A shock sensor event has been detected.

Byte0 User defined data.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 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.

12.3.7 Type = 6 (0x06) Smoke sensor

A smoke sensor event has been detected.

Byte0 User defined data.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 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.

12.3.8 Type = 7 (0x07) Heat sensor

A heat sensor event has been detected.

Byte0 User defined data.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 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.

12.3.9 Type = 8 (0x08) Panic switch

A panic switch event has been detected.

Byte0 User defined data.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 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.

12.3.10 Type = 9 (0x09) Door Contact

Indicates a door sensor reports that a door is open.

Byte0 User defined data.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 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.

12.3.11 Type = 10 (0x0A) Window Contact

Indicates a window sensor reports that a window is open.

Byte0 User defined data.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 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.

12.3.12 Type = 11 (0x0B) CO Sensor

CO sensor has detected CO at non secure level

Byte0 User defined data.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 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.

12.3.13 Type = 12 (0x0C) Frost detected

A frost sensor condition is detected

Byte0 User defined data.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 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.

12.3.14 Type = 13 (0x0D) Flame detected

Flame is detected.

Byte0 User defined data.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 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.

12.3.15 Type = 14 (0x0E) Oxygen Low

Low oxygen level detected.

Byte0 User defined data.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 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.

12.3.16 Type = 15 (0x0F) Weight detected.

Weight-detector triggered.

Byte0 User defined data.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 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.

12.3.17 Type = 16 (0x10) Water detected.

Water has been detected.

Byte0 User defined data.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 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.

12.3.18 Type = 17 (0x11) Condensation detected.

Condensation (humidity) detected.

Byte0 User defined data.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 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.

12.3.19 Type = 18 (0x12) Noise (sound) detected.

Noise (sound) has been detected.

Byte0 User defined data.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 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.

12.3.20 Type = 19 (0x13) Harmful sound levels detected.

Harmful sound levels detected.

Byte0 User defined data.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 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.

12.4 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 set to 000 and the first optional unit 001 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. See 8 for a full description on data coding used.

All events in this class are mirrored in Class=60 (0x3C) as floating point values with the default unit 12.10.

All events in this class are mirrored in Class=65 (0x41) as normalized integer values with index, zone, sub-zone (12.11) . Default unit is used.

12.4.1 Type = 0 (0x00) Undefined

Undefined measurement value.

12.4.2 Type = 1 (0x01) Count

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

Byte0 Data coding.

Byte1-7 Data with format defined by byte 0.

12.4.3 Type = 2 (0x02) Length/Distance

Default unit: Meter.

This is a measurement of a length or a distance.

Byte0 Data coding.

Byte1-7 Data with format defined by byte 0.

12.4.4 Type = 3 (0x03) Mass

Default unit: Kilogram.

This is a measurement of a mass.

Byte0 Data coding.

Byte1-7 Data with format defined by byte 0.

12.4.5 Type = 4 (0x04) Time

Default unit: Millisecond.

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

Byte0 Data coding.

Byte1-7 Data with format defined by byte 0.

12.4.6 Type = 5 (0x05) Electric Current

Default unit: Ampere.

This is a measurement of an electric current.

Byte0 Data coding.

Byte1-7 Data with format defined by byte 0.

12.4.7 Type = 6 (0x06) Temperature

Default unit: Kelvin.

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

This is a measurement of a temperature.

Byte0 Data coding.

Byte1-7 Data with format defined by byte 0.

12.4.8 Type = 7 (0x07) Amount of substance

Default unit: Mole.

This is a measurement of an amount of a substance.

Byte0 Data coding.

Byte1-7 Data with format defined by byte 0.

12.4.9 Type = 8 (0x08) Luminous Intensity (Intensity of light)

Default unit: Candela.

This is a measurement of luminous intensity.

Byte0 Data coding.

Byte1-7 Data with format defined by byte 0.

12.4.10 Type = 9 (0x09) Frequency

Default unit: Hertz.

This is a measurement of regular events during a second.

Byte0 Data coding.

Byte1-7 Data with format defined by byte 0.

12.4.11 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.

Byte0 Data coding.

Byte1-7 Data with format defined by byte 0.

12.4.12 Type = 11 (0x0B) Force

Default unit: Newton.

This is a measurement of force.

Byte0 Data coding.

Byte1-7 Data with format defined by byte 0.

12.4.13 Type = 12 (0x0C) Pressure

Default unit: Pascal.

Opt. unit: bar (1), psi (2)

This is a measurement of pressure.

Byte0 Data coding.

Byte1-7 Data with format defined by byte 0.

12.4.14 Type = 13 (0x0D) Energy

Default unit: Joule.

This is a measurement of energy.

Byte0 Data coding.

Byte1-7 Data with format defined by byte 0.

12.4.15 Type = 14 (0x0E) Power

Default unit: Watt.

This is a measurement of power.

Byte0 Data coding.

Byte1-7 Data with format defined by byte 0.

12.4.16 Type = 15 (0x0F) Electrical Charge

Default unit: Coulomb.

This is a measurement electrical charge.

Byte0 Data coding.

Byte1-7 Data with format defined by byte 0.

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

Default unit: Volt.

This is a measurement of electrical potential.

Byte0 Data coding.

Byte1-7 Data with format defined by byte 0.

12.4.18 Type = 17 (0x11) Electrical Capacitance

Default unit: Farad.

This is a measurement of electric capacitance.

Byte0 Data coding.

Byte1-7 Data with format defined by byte 0.

12.4.19 Type = 18 (0x012) Electrical Resistance

Default unit: Ohm.

This is a measurement of resistance.

Byte0 Data coding.

Byte1-7 Data with format defined by byte 0.

12.4.20 Type = 19 (0x13) Electrical Conductance

Default unit: Siemens.

This is a measurement of electrical conductance.

Byte0 Data coding.

Byte1-7 Data with format defined by byte 0.

12.4.21 Type = 20 (0x14) Magnetic Field Strength

Default unit: Ampere meters.

This is a measurement of magnetic field strength.

Byte0 Data coding.

Byte1-7 Data with format defined by byte 0.

12.4.22 Type = 21 (0x15) Magnetic Flux

Default unit: Weber.

This is a measurement of magnetic flux.

Byte0 Data coding.

Byte1-7 Data with format defined by byte 0.

12.4.23 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).

Byte0 Data coding.

Byte1-7 Data with format defined by byte 0.

12.4.24 Type = 23 (0x17) Inductance

Default unit: Henry.

This is a measurement of inductance.

Byte0 Data coding.

Byte1-7 Data with format defined by byte 0.

12.4.25 Type = 24 (0x18) Luminous Flux

Default unit: Lumen ($\text{lm} = \text{cd} * \text{sr}$)

This is a measurement of luminous Flux.

Byte0 Data coding.

Byte1-7 Data with format defined by byte 0.

12.4.26 Type = 25 (0x19) Illuminance

Default unit: Lux ($\text{lx} = \text{lm} / \text{m}^2$)

This is used to express both Illuminance (incidence of light) and Luminous Emittance (emission of light).

Byte0 Data coding.

Byte1-7 Data with format defined by byte 0.

12.4.27 Type = 26 (0x1A) Radiation dose

Default unit: Gray. Opt unit: Sievert.

This is a measurement of a radiation dose (Absorbed dose of ionizing radiation).

Byte0 Data coding.

Byte1-7 Data with format defined by byte 0.

12.4.28 Type = 27 (0x1B) Catalytic activity

Default unit: Katal.

This is a measurement of catalytic activity used in biochemistry.

Byte0 Data coding.

Byte1-7 Data with format defined by byte 0.

12.4.29 Type = 28 (0x1C) Volume

Default unit: cubic meter

Opt. Unit: Liter (dm^3) (1).

This is a measurement of volume.

Byte0 Data coding.

Byte1-7 Data with format defined by byte 0.

12.4.30 Type = 29 (0x1D) Sound intensity

Default unit: Bel.

Opt Unit: Neper (1), dB (2).

This is a measurement of sound intensity.

Byte0 Data coding.

Byte1-7 Data with format defined by byte 0.

12.4.31 Type = 30 (0x1E) Angle

Default unit: Radian (Plane angles).

Opt Unit: Degree (1)

Opt Unit: Arcminute (2)

Opt Unit: Arcseconds (3)

This is a measurement of an angle.

Byte0 Data coding.

Byte1-7 Data with format defined by byte 0.

12.4.32 Type = 31 (0x1F) Position

Default unit: Longitude. Opt. Unit: Latitude.

This is a measurement of a position.

Byte0 Data coding.

Byte1-7 Data with format defined by byte 0.

12.4.33 Type = 32 (0x20) Speed

Default unit: Meters per second.

This is a measurement of a speed.

Byte0 Data coding.

Byte1-7 Data with format defined by byte 0.

12.4.34 Type = 33 (0x21) Acceleration

Default unit: Meters per second/second.

This is a measurement of acceleration.

Byte0 Data coding.

Byte1-7 Data with format defined by byte 0.

12.4.35 Type = 34 (0x22) Tension

Default unit: N/m.

This is a measurement of tension.

Byte0 Data coding.

Byte1-7 Data with format defined by byte 0.

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

Default unit: Relative percentage 0-100%.

This is a measurement of relative moistness (Humidity).

Byte0 Data coding.

Byte1-7 Data with format defined by byte 0.

12.4.37 Type = 36 (0x24) Flow

Default unit: Cubic meters/second. Opt Unit: Liter/Second.

This is a measurement of flow.

Byte0 Data coding.

Byte1-7 Data with format defined by byte 0.

12.4.38 Type = 37 (0x25) Thermal resistance

Default unit: Thermal ohm K/W.

This is a measurement of thermal resistance.

Byte0 Data coding.

Byte1-7 Data with format defined by byte 0.

12.4.39 Type = 38 (0x26) Refractive power

Default unit: Dioptr (dpt) m-1.

This is a measurement of refractive power.

Byte0 Data coding.

Byte1-7 Data with format defined by byte 0.

12.4.40 Type = 39 (0x27) Dynamic viscosity

Default unit: Poiseuille (Pl) Pa . s.

This is a measurement of dynamic viscosity.

Byte0 Data coding.

Byte1-7 Data with format defined by byte 0.

12.4.41 Type = 40 (0x28) Sound impedance

Default unit: Rayal Pa . s/m.

This is a measurement of sound impedance.

Byte0 Data coding.

Byte1-7 Data with format defined by byte 0.

12.4.42 Type = 41 (0x29) Sound resistance

Default unit: Acoustic ohm Pa . s/ m³.

This is a measurement of refractive power.

Byte0 Data coding.

Byte1-7 Data with format defined by byte 0.

12.4.43 Type = 42 (0x2A) Electric elastance

Default unit: Darag F-1.

This is a measurement of electric elasticity.

Byte0 Data coding.

Byte1-7 Data with format defined by byte 0.

12.4.44 Type = 43 (0x2B) Luminous energy

Default unit: Talbot (tb = lm * s)

This is a measurement of luminous energy.

Byte0 Data coding.

Byte1-7 Data with format defined by byte 0.

12.4.45 Type = 44 (0x2C) Luminance

Default unit: Nit (nt = cd / m²)

This is a measurement of luminance.

Byte0 Data coding.

Byte1-7 Data with format defined by byte 0.

12.4.46 Type = 45 (0x2D) Chemical concentration

Default unit: Molal mol/kg.

This is a measurement of chemical concentration.

Byte0 Data coding.

Byte1-7 Data with format defined by byte 0.

12.4.47 Type = 46 (0x2E) Reserved

Reserved (previously was doublet of Type 26, don't use any longer!)

12.4.48 Type = 47 (0x2F) Dose equivalent

Default unit: Sievert J/Kg.

This is a measurement of dose equivalent.

Byte0 Data coding.

Byte1-7 Data with format defined by byte 0.

12.4.49 Type = 48 (0x30) Reserved

Reserved (was doublet of type 24, do not use any longer!)

12.4.50 Type = 49 (0x31) Dew Point

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

This is a measurement of the Dew Point.

Byte0 Data coding.

Byte1-7 Data with format defined by byte 0.

12.4.51 Type = 50 (0x32) Relative Level

Default unit: Relative value.

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

Byte0 Data coding.

Byte1-7 Data with format defined by byte 0.

12.4.52 Type = 51 (0x33) Altitude.

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

Altitude in meters.

Byte0 Data coding.

Byte1-7 Data with format defined by byte 0.

12.4.53 Type = 52 (0x34) Area

Default unit: square meter (m^2)

Area in square meter.

Byte0 Data coding.

Byte1-7 Data with format defined by byte 0.

12.4.54 Type = 53 (0x35) Radiant intensity

Default unit: watt per steradian (W / sr)

Radiated power per room angle.

Byte0 Data coding.

Byte1-7 Data with format defined by byte 0.

12.4.55 Type = 54 (0x36) Radiance

Default unit: att per steradian per square metre (W / (sr * m^2))

Byte0 Data coding.

Byte1-7 Data with format defined by byte 0.

12.4.56 Type = 55 (0x37) Irradiance, Exitance, Radiosity

Default unit: watt per square metre (W / m²)

Power emitted from or striking onto a surface or area.

Byte0 Data coding.

Byte1-7 Data with format defined by byte 0.

12.4.57 Type = 56 (0x38) Spectral radiance

Default unit: watt per steradian per square metre per nm (W·sr-1·m-2·nm-1)
Opt. Units: watt per steradian per metre³ (W·sr-1·m-3)
watt per steradian per square metre per hertz (W·sr-1·m-3)

Byte0 Data coding.

Byte1-7 Data with format defined by byte 0.

12.4.58 Type = 57 (0x39) Spectral irradiance

Default unit: watt per square metre per nm (W·m-2·nm-1) Opt.
Units: watt per metre³ (W·m-3) watt per square metre per hertz
(W·m-2·Hz-1)

Byte0 Data coding.

Byte1-7 Data with format defined by byte 0.

12.5 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.

12.5.1 Type = 0 (0x00) Undefined

General event.

12.5.2 Type = 1 (0x01) I/O – value

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

Byte0 Data coding.

Byte1-7 Data with format defined by byte 0.

12.5.3 Type = 2 (0x02) A/D value

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

Byte0 Data coding.

Byte1-7 Data with format defined by byte 0.

12.5.4 Type = 3 (0x03) D/A value

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

Byte0 Data coding.

Byte1-7 Data with format defined by byte 0.

12.5.5 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.

Byte0 Data coding.

Byte1-7 Data with format defined by byte 0.

12.5.6 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 a percentage. For a digital transmission Signal Level it can be used to give an indication of the analogue signal and CLASS1.DATA, Type = 6, 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)

Byte0 Data coding.

Byte1-7 Data with format defined by byte 0.

12.5.7 Type = 6 (0x06) Signal Quality

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)

Byte0 Data coding.

Byte1-7 Data with format defined by byte 0.

12.6 Class=20 (0x14) Information

CLASS1.INFORMATION

Description

Most of the events 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.

12.6.1 Type = 1 (0x01) Button

A button has been pressed/released.

Byte0 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.

Byte1 Zone for which event applies to (0-255). 255 is all zones

Byte2 Sub-zone for which event applies to (0-255). 255 is all sub-zones

Byte3 MSB of code for button.

Byte4 LSB of code for button.

Byte5 MSB of optional code-page.

Byte6 LSB of optional code-page.

12.6.2 Type = 2 (0x02) Mouse

A mouse movement has occurred.

Byte0 If zero absolute coordinates follow. If equal to one relative coordinates
follow.

Byte1 Zone for which event applies to (0-255). 255 is all zones

Byte2 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.

12.6.3 Type = 3 (0x03) On

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

Byte0 index. Often used as an index for channels within a module.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 Sub-zone for which event applies to (0-255). 255 is all sub-zones.

12.6.4 Type = 4 (0x04) Off

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

Byte0 Index.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 Sub-zone for which event applies to (0-255). 255 is all sub-zones.

12.6.5 Type = 5 (0x05) Alive

A node tells the world that it is alive.

Byte0 User specified.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 Sub-zone for which event applies to (0-255). 255 is all sub-zones.

12.6.6 Type = 6 (0x06) Terminating

A node tells the world that it is terminating.

Byte0 User specified.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 Sub-zone for which event applies to (0-255). 255 is all sub-zones.

12.6.7 Type = 7 (0x07) Opened

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

Byte0 User specified.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 Sub-zone for which event applies to (0-255). 255 is all sub-zones.

12.6.8 Type = 8 (0x08) Closed

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

Byte0 User specified.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 Sub-zone for which event applies to (0-255). 255 is all sub-zones.

12.6.9 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 it is recommended that all nodes send this event on regular intervals.

Byte0 User specified.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 Sub-zone for which event applies to (0-255). 255 is all sub-zones.

12.6.10 Type = 10 (0x0A) Below limit

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

Byte0 Reserved.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 Sub-zone for which event applies to (0-255). 255 is all sub-zones.

12.6.11 Type = 11 (0x0B) Above limit

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

Byte0 Reserved.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 Sub-zone for which event applies to (0-255). 255 is all sub-zones.

12.6.12 Type = 12 (0x0C) Pulse

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

Byte0 Reserved.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 Sub-zone for which event applies to (0-255). 255 is all sub-zones.

12.6.13 Type = 13 (0x0D) Error

A node indicates that an error occurred.

Byte0 User specified.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 Sub-zone for which event applies to (0-255). 255 is all sub-zones.

12.6.14 Type = 14 (0x0E) Resumed

A node indicates that it has resumed operation.

Byte0 Reserved.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 Sub-zone for which event applies to (0-255). 255 is all sub-zones.

12.6.15 Type = 15 (0x0F) Paused

A node indicates that it has paused.

Byte0 Reserved.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 Sub-zone for which event applies to (0-255). 255 is all sub-zones.

12.6.16 Type = 16 (0x10) Sleeping

A node indicates that it entered a sleeping state.

Byte0 Reserved.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 Sub-zone for which event applies to (0-255). 255 is all sub-zones.

12.6.17 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 its morning state.

Byte0 Reserved.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 Sub-zone for which event applies to (0-255). 255 is all sub-zones.

12.6.18 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 its day state.

Byte0 Reserved.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 Sub-zone for which event applies to (0-255). 255 is all sub-zones.

12.6.19 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 its afternoon state.

Byte0 Reserved.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 Sub-zone for which event applies to (0-255). 255 is all sub-zones.

12.6.20 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 its evening state.

Byte0 Reserved.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 Sub-zone for which event applies to (0-255). 255 is all sub-zones.

12.6.21 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 its night state.

Byte0 Reserved.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 Sub-zone for which event applies to (0-255). 255 is all sub-zones.

12.6.22 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.

Byte0 Reserved.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 Sub-zone for which event applies to (0-255). 255 is all sub-zones.

12.6.23 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 days work or similar situation. All alarm systems should be activated in this situation.

Byte0 Reserved.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 Sub-zone for which event applies to (0-255). 255 is all sub-zones.

12.6.24 Type = 24 (0x18) Stop

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

Byte0 Reserved.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 Sub-zone for which event applies to (0-255). 255 is all sub-zones.

12.6.25 Type = 25 (0x19) Start

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

Byte0 Reserved.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 Sub-zone for which event applies to (0-255). 255 is all sub-zones.

12.6.26 Type = 26 (0x1A) ResetCompleted

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

Byte0 Reserved.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 Sub-zone for which event applies to (0-255). 255 is all sub-zones.

12.6.27 Type = 27 (0x1B) Interrupted

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

Byte0 Reserved.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 Sub-zone for which event applies to (0-255). 255 is all sub-zones.

12.6.28 Type = 28 (0x1C) PreparingToSleep

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

Byte0 Reserved.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 Sub-zone for which event applies to (0-255). 255 is all sub-zones.

12.6.29 Type = 29 (0x1D) WokenUp

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

Byte0 Reserved.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 Sub-zone for which event applies to (0-255). 255 is all sub-zones.

12.6.30 Type = 30 (0x1E) Dusk

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

Byte0 Reserved.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 Sub-zone for which event applies to (0-255). 255 is all sub-zones.

12.6.31 Type = 31 (0x1F) Dawn

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

Byte0 Reserved.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 Sub-zone for which event applies to (0-255). 255 is all sub-zones.

12.6.32 Type = 32 (0x20) Active

A node indicates that its active.

Byte0 Reserved.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 Sub-zone for which event applies to (0-255). 255 is all sub-zones.

12.6.33 Type = 33 (0x21) Inactive

A node indicates that its inactive.

Byte0 Reserved.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 Sub-zone for which event applies to (0-255). 255 is all sub-zones.

12.6.34 Type = 34 (0x22) Busy

A node indicates that its busy.

Byte0 Reserved.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 Sub-zone for which event applies to (0-255). 255 is all sub-zones.

12.6.35 Type = 35 (0x23) Idle

A node indicates that its idle.

Byte0 Reserved.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 Sub-zone for which event applies to (0-255). 255 is all sub-zones.

12.6.36 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 some other sequential stream.

Byte0 Sequence number which is increased by one for every new event with stream data. The sequence number is increased for every frame sent going from 0 - 255 and back to 0 and so on if needed. A new stream starts with a sequence number of 0.

Byte1-7 Stream data.

12.6.37 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 deduced from the type.

Byte0 Bit 0,1 - Event code. Bit 2-7 - Token device type code.

Byte1 Zone.

Byte2 Sub-zone.

Byte3 Frame index (Increase by one for every frame sent out for one token activity. Start with zero).

Byte4-7 Token data.

Event codes

- 0 Touched and released.
- 1 Touched.
- 2 Released.
- 3 Reserved.

Token device type codes

- 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 Biometric device. 256-bits
- 17 Biometric 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>
- 20 GSM IMSI code (International Mobile Subscriber Identity) packed in 64 bits. <http://en.wikipedia.org/wiki/IMSI>
- 21 RFID Token. 40-bits
- 22 RFID Token. 32-bits

23	RFID Token. 24-bits
24	RFID Token. 16-bits
25	RFID Token. 8-bits
26-63	Reserved.

12.6.38 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 some other sequential stream.

Byte0 Zone.

Byte1 Sub-zone.

Byte2 Sequence number which is increased by one for every new event with stream data. The sequence number is increased for every frame sent going from 0 - 255 and back to 0 and so on if needed. A new stream starts with a sequence number of 0.

Byte3-7 Stream data.

12.6.39 Type = 39 (0x27) Confirm.

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

Byte0 Zone.

Byte1 Sub-zone.

Byte2 Sequence number byte 3 Class MSB.

Byte4 Class LSB.

Byte5 Type MSB.

Byte6 Type LSB.

12.6.40 Type = 40 (0x28) Level Changed.

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

Byte0 Relative or absolute level.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 Sub-zone for which event applies to (0-255). 255 is all sub-zones.

12.6.41 Type = 41 (0x29) Warning

A node indicates that a warning condition occurred.

Byte0 Relative or absolute level.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 Sub-zone for which event applies to (0-255). 255 is all sub-zones.

12.6.42 Type = 42 (0x2A) State

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

Byte0 User specified.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 Sub-zone for which event applies to (0-255). 255 is all sub-zones.

Byte3 State changed from.

Byte4 New State.

12.6.43 Type = 43 (0x2B) Action Trigger

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

Byte0 Action ID.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 Sub-zone for which event applies to (0-255). 255 is all sub-zones.

12.6.44 Type = 44 (0x2C) Sunrise

A node indicates that sunrise is detected/calculated.

Byte0 User specified.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 Sub-zone for which event applies to (0-255). 255 is all sub-zones.

12.6.45 Type = 45 (0x2D) Sunset

A node indicates that sunset is detected/calculated.

Byte0 User specified.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 Sub-zone for which event applies to (0-255). 255 is all sub-zones.

12.6.46 Type = 46 (0x2E) Start of record

This event is used to mark the start of a multi-frame 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.

Byte0 Index for record.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 Sub-zone for which event applies to (0-255). 255 is all sub-zones.

Byte3 Number of frames to follow or zero for not used.

12.6.47 Type = 47 (0x2F) End of record

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

Byte0 Index for record.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 Sub-zone for which event applies to (0-255). 255 is all sub-zones.

12.6.48 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.

Byte0 0

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 Sub-zone for which event applies to (0-255). 255 is all sub-zones.

Byte3 Code for pre-set that has been set.

12.6.49 Type = 49 (0x31) Detect

This event is used to tell the system that a detection of some kind has occurred.

Byte0 Index

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 Sub-zone for which event applies to (0-255). 255 is all sub-zones.

The first byte is used as an index if a module have several channels or detectors.

12.6.50 Type = 50 (0x32) Overflow

This event is used to tell the system that an overflow of some kind has occurred.

Byte0 Index

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 Sub-zone for which event applies to (0-255). 255 is all sub-zones.

The first byte is used as an index if a module have several channels or detectors.

12.7 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 events to specific devices. A device can belong to a zone which select limit events 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.

12.7.1 Type = 0 (0x00) Undefined

General control.

12.7.2 Type = 1 (0x01) Mute on/off

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

Byte0 If equal to zero no mute else mute.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 Sub-zone for which event applies to (0-255). 255 is all sub-zones.

12.7.3 Type = 2 (0x02) (All) Lamp(s) on/off

Turn on/off lamps on nodes in zone.

Byte0 If equal to zero off else on.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 Sub-zone for which event applies to (0-255). 255 is all sub-zones.

12.7.4 Type = 3 (0x03) Open

Perform open on all nodes in zone.

Byte0 Optional byte that have a meaning given by the issuer of the event.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 Sub-zone for which event applies to (0-255). 255 is all sub-zones.

12.7.5 Type = 4 (0x04) Close

Perform close on all nodes in zone.

Byte0 Optional byte that have a meaning given by the issuer of the event.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 Sub-zone for which event applies to (0-255). 255 is all sub-zones.

12.7.6 Type = 5 (0x05) TurnOn

Turn On a nodes in a zone/subzone.

Byte0 Optional byte that have a meaning given by the issuer of the event.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 Sub-zone for which event applies to (0-255). 255 is all sub-zones.

12.7.7 Type = 6 (0x06) TurnOff

Turn Off a nodes in a zone/subzone.

Byte0 Optional byte that have a meaning given by the issuer of the event.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 Sub-zone for which event applies to (0-255). 255 is all sub-zones.

12.7.8 Type = 7 (0x07) Start

Start all nodes in a zone.

Byte0 Optional byte that have a meaning given by the issuer of the event.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 Sub-zone for which event applies to (0-255). 255 is all sub-zones.

12.7.9 Type = 8 (0x08) Stop

Stop all nodes in zone.

Byte0 Optional byte that have a meaning given by the issuer of the event.

Byte1 Zone for which event applies to (0-255). 255 is all zones

Byte2 sub-zone for which event applies to (0-255). 255 is all sub-zones

12.7.10 Type = 9 (0x09) Reset

Perform Reset on all nodes in zone.

Byte0 Optional byte that have a meaning given by the issuer of the event.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 Sub-zone for which event applies to (0-255). 255 is all sub-zones.

12.7.11 Type = 10 (0x0A) Interrupt

Perform Interrupt on all nodes in zone.

Byte0 Interrupt level. (0 – 255 , zero is lowest interrupt level.).

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 Sub-zone for which event applies to (0-255). 255 is all sub-zones.

12.7.12 Type = 11 (0x0B) Sleep

Perform Sleep on all nodes in zone.

Byte0 Optional byte that have a meaning given by the issuer of the event.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 Sub-zone for which event applies to (0-255). 255 is all sub-zones.

12.7.13 Type = 12 (0x0C) Wakeup

Wakeup all nodes in zone.

Byte0 Optional byte that have a meaning given by the issuer of the event.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 Sub-zone for which event applies to (0-255). 255 is all sub-zones.

12.7.14 Type = 13 (0x0D) Resume

Resume all nodes in zone.

Byte0 Optional byte that have a meaning given by the issuer of the event.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 Sub-zone for which event applies to (0-255). 255 is all sub-zones.

12.7.15 Type = 14 (0x0E) Pause

Pause all nodes in zone.

Byte0 Optional byte that have a meaning given by the issuer of the event.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 Sub-zone for which event applies to (0-255). 255 is all sub-zones.

12.7.16 Type = 15 (0x0F) Activate

Activate all nodes in zone.

Byte0 Optional byte that have a meaning given by the issuer of the event.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 Sub-zone for which event applies to (0-255). 255 is all sub-zones.

12.7.17 Type = 16 (0x10) Deactivate

Deactivate all nodes in zone.

Byte0 Optional byte that have a meaning given by the issuer of the event.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 Sub-zone for which event applies to (0-255). 255 is all sub-zones.

12.7.18 Type = 17 (0x11) Reserved for future use

Reserved.

12.7.19 Type = 18 (0x12) Reserved for future use

Reserved.

12.7.20 Type = 19 (0x13) Reserved for future use

Reserved.

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

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

Byte0 Value (0 – 100) . 0 = off, 100 = full on. 254 dim down one step. 255 dim up one step.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 Sub-zone for which event applies to (0-255). 255 is all sub-zones.

12.7.22 Type = 21 (0x15) Change Channel

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

Byte0 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.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 Sub-zone for which event applies to (0-255). 255 is all sub-zones.

12.7.23 Type = 22 (0x16) Change Level

Change an absolute level.

Byte0 Absolute level.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 Sub-zone for which event applies to (0-255). 255 is all sub-zones.

12.7.24 Type = 23 (0x17) Relative Change Level

Byte0 Relative level.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 Sub-zone for which event applies to (0-255). 255 is all sub-zones.

12.7.25 Type = 24 (0x18) Measurement Request

Byte0 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.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 Sub-zone for which event applies to (0-255). 255 is all sub-zones.

12.7.26 Type = 25 (0x19) Stream Data

Byte0 Sequence number which is increase by one for each stream data event sent.

Byte1-7 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.

12.7.27 Type = 26 (0x1A) Sync

Synchronize events on a segment.

Byte0 Index for subunits within modules. 255 is all subunits.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 Sub-zone for which event applies to (0-255). 255 is all sub-zones.

12.7.28 Type = 27 (0x1B) Zoned Stream Data

Byte0 Sequence number which is increase by one for each stream data event sent.

Byte1 Zone which should received stream.

Byte2 SubZone which should received stream.

Byte3-7 Stream data.

12.7.29 Type = 28 (0x1C) 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/sub-zone.

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

Byte0 Code for pre-set to set.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 Sub-zone for which event applies to (0-255). 255 is all sub-zones.

12.7.30 Type = 29 (0x1D) Toggle state

Toggle the state of a node.

Note: This may be a bad design option as it often demands that the state should be known for the node on beforehand.

Byte0 Optional byte that have a meaning given by the issuer of the event.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 Sub-zone for which event applies to (0-255). 255 is all sub-zones.

Type = 29 (0x1E) Toggle state

12.7.31 Type = 30 (0x1E) Timed pulse on.

With this event it is possible to generate a timed pulse that is on for a specified time.

Byte0 Optional byte that have a meaning given by the issuer of the event.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 Sub-zone for which event applies to (0-255). 255 is all sub-zones.

Byte3 Control byte.

Byte4-7 Set time as a long with MSB in the first byte.

The control byte have the following bits defined

Bit	Description	
0-3	Value	Description
	0	Time specified in microseconds.
	1	Time specified in milliseconds.
	2	Time specified in seconds.
	3	Time specified in minutes.
	4	Time specified in hours.
	5	Time specified in days.
4	Reserved.	
5	Reserved.	
6	Send on event (Class=20 Type = 3 (0x03) On) when pulse goes on.	
7	Send off event (Class=20 Type = 4 (0x04) Off) when pulse goes off.	

12.7.32 Type = 31 (0x1F) Timed pulse off.

With this event it is possible to generate a timed pulse that is off for a specified time.

Byte0 Optional byte that have a meaning given by the issuer of the event.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 Sub-zone for which event applies to (0-255). 255 is all sub-zones.

Byte3 Control byte.

Byte4-7 Set time as a long with MSB in the first byte.

The control byte have the following bits defined

<i>Bit</i>	<i>Description</i>	
	<i>Value</i>	<i>Description</i>
0-3	0	Time specified in microseconds.
	1	Time specified in milliseconds.
	2	Time specified in seconds.
	3	Time specified in minutes.
	4	Time specified in hours.
	5	Time specified in days.
4	Reserved.	
5	Reserved.	
6	Send on event (Class=20 Type = 3 (0x03) On) when pulse goes on.	
7	Send off event (Class=20 Type = 4 (0x04) Off) when pulse goes off.	

12.7.33 Type = 32 (0x20) Set country/language.

Byte0 Country/Language code.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 Sub-zone for which event applies to (0-255). 255 is all sub-zones.

Byte3-7 Country/Language code specific

Language code	Description	Example
0	Custom coded system	Byte 3 = 0 English, Byte 3 = 1 German or similar.
1	ISO 639-1	<i>nl</i> for Dutch, <i>en</i> for English.
2	ISO 639-2/T	<i>nid</i> for Dutch, <i>eng</i> for English.
3	ISO 639-2/B	<i>dut</i> for Dutch, <i>eng</i> for English.
4	ISO 639-3	<i>nid</i> for Dutch, <i>eng</i> for English.
5	IETF (RFC-5646/4647)	en-US for American English. en-GB British.

ISO codes can be found here http://en.wikipedia.org/wiki/List_of_ISO_639-1_codes

12.8 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.

12.8.1 Type=1 (0x1) Playback

This is for controlling playback functionality

Byte0 Function

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 Sub-zone for which event applies to (0-255). 255 is all sub-zones.

Function Codes

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 controlled device as response to this event.

12.8.2 Type=2 (0x2) NavigatorKey English

This is typically for navigation functions or DVD controls

Byte0 Function

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 Sub-zone for which event applies to (0-255). 255 is all sub-zones.

Function Codes

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)

12.8.3 Type=3 (0x3) Adjust Contrast

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

Byte0 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 an extended event and that the specific contrast level is sent in byte 3 and after.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 Sub-zone for which event applies to (0-255). 255 is all sub-zones.

12.8.4 Type=4 (0x4) Adjust Focus

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

Byte0 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 an extended event and that the specific focus level is sent in byte 3 and after.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 Sub-zone for which event applies to (0-255). 255 is all sub-zones.

12.8.5 Type=5 (0x5) Adjust Tint

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

Byte0 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 an extended event and that the specific tint level is sent in byte 3 and after.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 Sub-zone for which event applies to (0-255). 255 is all sub-zones.

12.8.6 Type=6 (0x6) Adjust Color Balance

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

Byte0 Reserved.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 Sub-zone for which event applies to (0-255). 255 is all sub-zones.

12.8.7 Type=7 (0x7) Adjust Brightness

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

Byte0 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 an extended event and that the specific brightness level is sent in byte 3 and after.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 Sub-zone for which event applies to (0-255). 255 is all sub-zones.

12.8.8 Type=8 (0x8) Adjust Hue

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

Byte0 Reserved.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 Sub-zone for which event applies to (0-255). 255 is all sub-zones.

12.8.9 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.

Byte0 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 an extended event and that the specific bass level is sent in byte 3 and after.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 Sub-zone for which event applies to (0-255). 255 is all sub-zones.

12.8.10 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.

Byte0 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 an extended event and that the specific treble level is sent in byte 3 and after.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 Sub-zone for which event applies to (0-255). 255 is all sub-zones.

12.8.11 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.

Byte0 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 an extended event and that the specific volume level is sent in byte 3 and after.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 Sub-zone for which event applies to (0-255). 255 is all sub-zones.

12.8.12 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 center speaker.

Byte0 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 an extended event and that the specific volume level is sent in byte 3 and after.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 Sub-zone for which event applies to (0-255). 255 is all sub-zones.

12.8.13 Type=13 (0xD) Adjust Center Volume

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

Byte0 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 an extended event and that the specific volume level is sent in byte 3 and after.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 Sub-zone for which event applies to (0-255). 255 is all sub-zones.

12.8.14 Type=14 (0xE) Adjust Rear Volume

This is typically for adjusting the rear speaker volume level.

Byte0 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 an extended event and that the specific volume level is sent in byte 3 and after.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 Sub-zone for which event applies to (0-255). 255 is all sub-zones.

12.8.15 Type=15 (0xF) Adjust Side Volume

This is typically for adjusting the side speaker volume level.

Byte0 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 an extended event and that the specific volume level is sent in byte 3 and after.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 Sub-zone for which event applies to (0-255). 255 is all sub-zones.

12.8.16 Type=16 to 19 Reserved

These are reserved for other future speaker combinations

12.8.17 Type=20 (0x14) Select Disk

This is typically for selecting a disk for playback

Byte0 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 an extended event and that the disk number is sent in byte 3 and after.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 Sub-zone for which event applies to (0-255). 255 is all sub-zones.

12.8.18 Type=21 (0x15) Select Track

This is typically for selecting a track for playback

Byte0 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 an extended event and that the track number is sent in byte 3 and after.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 Sub-zone for which event applies to (0-255). 255 is all sub-zones.

12.8.19 Type=22 (0x16) Select Album/Play list

This is typically for selecting an album or play-list for playback

Byte0 A value between 0 and 127 indicates the album/play-list number. A value between 128 and 159 is change down by the specified number of albums/play-lists. 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 an extended event and that the album number is sent in byte 3 and after.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 sub-zone for which event applies to (0-255). 255 is all sub-zones.

12.8.20 Type=23 (0x17) Select Channel

This is typically for selecting a TV Channel

Byte0 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 an extended event and that the channel number is sent in byte 3 and after.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 Sub-zone for which event applies to (0-255). 255 is all sub-zones.

12.8.21 Type=24 (0x18) Select Page

This is typically for selecting a page of a film

Byte0 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 an extended event and that the page number is sent in byte 3 and after.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 Sub-zone for which event applies to (0-255). 255 is all sub-zones.

12.8.22 Type=25 (0x19) Select Chapter

This is typically for selecting a chapter of a film

Byte0 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 an extended event and that the chapter number is sent in byte 3 and after.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 Sub-zone for which event applies to (0-255). 255 is all sub-zones.

12.8.23 Type=26 (0x1A) Select Screen Format

This is for controlling screen format of a display device

Byte0 0 = Auto, 1 = Just, 2 = Normal, 3 = Zoom.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 Sub-zone for which event applies to (0-255). 255 is all sub-zones.

12.8.24 Type=27 (0x1B) Select Input Source

This is for controlling the input source of a playback device

Byte0 Device code

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 Sub-zone for which event applies to (0-255). 255 is all sub-zones.

Device Code

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

12.8.25 Type=28 (0x1C) Select Output

This is for controlling the output of a playback device

Byte0 Output Code

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 Sub-zone for which event applies to (0-255). 255 is all sub-zones.

Output Code

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

12.8.26 Type=29 (0x1D) Record

Control a recording device.

Byte0 0 - Start to record, 1 - Stop record, 2 - Disable, AGC 3 - Enable AGC.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 Sub-zone for which event applies to (0-255). 255 is all sub-zones.

12.8.27 Type=30 (0x1E) Set Recording Volume

Control a recording device.

Byte0 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 an extended event and that the specific contrast level is sent in byte 3 and after.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 Sub-zone for which event applies to (0-255). 255 is all sub-zones.

12.8.28 Type=40 (0x28) Tivo Function

This is typically for accessing TIVO functions

Byte0 TIVO Code

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 Sub-zone for which event applies to (0-255). 255 is all sub-zones.

TIVO Code

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

12.8.29 Type=50 (0x32) Get Current Title

Get the title for the current active media.

Byte0 Reserved.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 Sub-zone for which event applies to (0-255). 255 is all sub-zones.

12.8.30 Type=51 (0x33) Set media position in milliseconds

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

Byte0 Reserved

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 Sub-zone for which event applies to (0-255). 255 is all sub-zones.

Byte 3-7 Position in milliseconds, This is an integer with a size specified by the event size. This 0xFF, 0xFFFF, 0xFFFFFFFF, 0xFFFFFFFF and 0xFFFFFFFFFFFF is the maximum that can be sent for different sizes.

12.8.31 Type=52 (0x34) Get media information

Get various media information from a device.

Byte0 Type of media information requested. 1 - Current Title, 1 - Get Folders, 2 - Get Disks, 3 - Get Tracks, 4 - Get Albums/Play list,s 5 - Get Channels, 6 - Get Pages, 7 - Get Chapters

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 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 response.

12.8.32 Type=53 (0x35) Remove Item from Album

Remove an item from an album.

Byte0 0-128 - Pos to remove from album/play-list A value of 255 means that this is an extended event and that the specific contrast level is sent in byte 3 and after.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 Sub-zone for which event applies to (0-255). 255 is all sub-zones.

12.8.33 Type=54 (0x36) Remove all Items from Album

Remove all items from an album.

Byte0 Reserved.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 Sub-zone for which event applies to (0-255). 255 is all sub-zones.

12.8.34 Type=55 (0x37) Save Album/Play list

Save album/play-list to permanent storage.

Byte0 0 - Do not overwrite if it already exists 1 - Overwrite if it exists.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 Sub-zone for which event applies to (0-255). 255 is all sub-zones.

12.8.35 Type=60 (0x3C) Multimedia 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.

Response should be Type=61

Byte0 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 = Sample rate(integer)

```

12 = Bit-rate(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 = Media position in milliseconds(integer)
25 = Media-length in milliseconds(integer)
26 = Version(string)
27 = Album/Play list(string)
28 = Play file(URL)
29 = Add file to album/play-list (URL)
30 = Current Folder (URL)
31 = Folder content(URL)
32 = Set Folder(URL)
33 = Get Folder content(URL)
34 = Get Folder content albums/play-lists(URL)
35 = Get Folder content filter(string)
36 = Disks list(String)
37 = Folders list(String)
38 = Tracks list(String)
39 = Albums/Play list list(String)
40 = Channels list(String)
41 = Pages list(String)
42 = Chapters list(String)
43 = New Album/Play list(URL)

```

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 Sub-zone for which event applies to (0-255). 255 is all sub-zones.

Byte3 Index. Base 0. Increase by one for every fragment of the title sent.

Byte4-7 Data.

The last fragment is sent with no data.

Lists in string form have list items separated with a zero (0x00).

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

12.8.36 Type=61 (0x3D) Multimedia Control reasons

Response for multimedia control.

12.9 Class=50 (0x32) Alert On LAN

CLASS1.AOL

Description

AOL Event. The main idea of AOL is to send warnings to remote administrators about different PC conditions using a LAN. Info here http://en.wikipedia.org/wiki/Alert_on_LAN

12.9.1 Type = 0 (0x00)

Undefined

Undefined measurement value.

12.9.2 Type = 1 (0x01) System unplugged from power source

This node was unplugged from its power source.

Byte0 Index for record.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 Sub-zone for which event applies to (0-255). 255 is all sub-zones.

12.9.3 Type = 2 (0x02) System unplugged from network

This node was unplugged from the network.

Byte0 Index for record.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 Sub-zone for which event applies to (0-255). 255 is all sub-zones.

12.9.4 Type = 3 (0x03) Chassis intrusion

This node detected chassis intrusion.

Byte0 Index for record.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 Sub-zone for which event applies to (0-255). 255 is all sub-zones.

12.9.5 Type = 4 (0x04) Processor removal

This node detected processor removal.

Byte0 Index for record.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 Sub-zone for which event applies to (0-255). 255 is all sub-zones.

12.9.6 Type = 5 (0x05) System environmental errors

This node detected system environmental errors.

Byte0 Index for record.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 Sub-zone for which event applies to (0-255). 255 is all sub-zones.

12.9.7 Type = 6 (0x06) High temperature

This node detected high temperature.

Byte0 Index for record.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 Sub-zone for which event applies to (0-255). 255 is all sub-zones.

12.9.8 Type = 7 (0x07) Fan speed problem

This node detected Fan speed problem.

Byte0 Index for record.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 Sub-zone for which event applies to (0-255). 255 is all sub-zones.

12.9.9 Type = 8 (0x08) Voltage fluctuations

This node detected Voltage fluctuations.

Byte0 Index for record.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 Sub-zone for which event applies to (0-255). 255 is all sub-zones.

12.9.10 Type = 9 (0x09) Operating system errors

This node detected Operating system errors.

Byte0 Index for record.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 Sub-zone for which event applies to (0-255). 255 is all sub-zones.

12.9.11 Type = 10 (0x0A) System power-on errors

This node detected System power-on errors.

Byte0 Index for record.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 Sub-zone for which event applies to (0-255). 255 is all sub-zones.

12.9.12 Type = 11 (0x0B) System is hung

This node detected System is hung.

Byte0 Index for record.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 Sub-zone for which event applies to (0-255). 255 is all sub-zones.

12.9.13 Type = 12 (0x0C) Component failure

This node detected Component failure.

Byte0 Index for record.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 Sub-zone for which event applies to (0-255). 255 is all sub-zones.

12.9.14 Type = 13 (0x0D) Remote system reboot upon report of a critical failure

This node detected Remote system reboot upon report of a critical failure.

Byte0 Index for record.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 Sub-zone for which event applies to (0-255). 255 is all sub-zones.

Type = 14 (0x0E) Repair Operating System

This node detected Repair Operating System.

Byte0 Index for record.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 Sub-zone for which event applies to (0-255). 255 is all sub-zones.

12.9.15 Type = 15 (0x0F) Update BIOS image

This node detected Update BIOS image.

Byte0 Index for record.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 Sub-zone for which event applies to (0-255). 255 is all sub-zones.

12.9.16 Type = 16 (0x10) Update Perform other diagnostic procedures

This node detected Update Perform other diagnostic procedures.

Byte0 Index for record.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 Sub-zone for which event applies to (0-255). 255 is all sub-zones.

Type = 16 (0x10) Update Perform other diagnostic procedures

12.10 Class=60 (0x3C) Double precision floating point measurement

CLASS1.MEASUREMENT64

Description

Floating point double precision measurements. This class mirrors the standard measurement events is CLASS1.MEASUREMENT=10 12.4. The measurement unit is always the standard unit.

The value is a "double" - IEEE-754, 64 Bits, double precision.

```
s eeeeeeee mmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmm  
s = sign bit ( 1-bit )  
e = exponent ( 11-bits )  
m = mantissa ( 52-bits )
```

That is a total of 64-bits. The most significant byte is stored in byte 0.

Byte0 s e₁₀ e₉ e₈ e₇ e₆ e₅ e₄

Byte1 e₃ e₂ e₁ e₀ m₅₁ m₅₀ m₄₉ m₄₈

Byte2 m₄₇ m₄₆ m₄₅ m₄₄ m₄₃ m₄₂ m₄₁ m₄₀

Byte3 m₃₉ m₃₈ m₃₇ m₃₆ m₃₅ m₃₄ m₃₃ m₃₂

Byte4 m₃₁ m₃₀ m₂₉ m₂₈ m₂₇ m₂₆ m₂₅ m₂₄

Byte5 m₂₃ m₂₂ m₂₁ m₂₀ m₁₉ m₁₈ m₁₇ m₁₆

Byte6 m₁₅ m₁₄ m₁₃ m₁₂ m₁₁ m₁₀ m₉ m₈

Byte7 m₇ m₆ m₅ m₄ m₃ m₂ m₁ m₀

12.11 Class=65 (0x41) Measurement with zone

CLASS1.MEASUREZONE

Description

Measurements with zone information. This class mirrors the standard measurement events is CLASS1.MEASUREMENT=10 12.4 with the difference that index, zone, and sub-zone is added. This in turn limits the data-coding options to normalized integer (see 8 for a description). The default unit for the measurement should always be used.

Byte0 Index for sensor.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 Sub-zone for which event applies to (0-255). 255 is all sub-zones.

Byte3 Normalizer byte

Byte4-7 Normalized integer (see 8 for a description).

The number of bytes in the event tells how large the normalized integer is (1,2,3 or 4 bytes).

12.12 Class=70 (0x46) Single precision floating point measurement

CLASS1.MEASUREMENT32

Description

Floating point single precision measurements. This class mirrors the standard measurement events is CLASS1.MEASUREMENT=10 12.4. The measurement unit is always the standard unit.

The value is a "float" - IEEE-754, 32 Bits, single precision.

```
s e e e e e e e e m m m m m m m m m m m m m m m m m m  
s = sign bit ( 1-bit )  
e = exponent ( 8-bits )  
m = mantissa ( 23-bits )
```

That is a total of 64-bits. The most significant byte is stored in byte 0.

Byte0 Data codig byte

Byte1 s e₇ e₆ e₅ e₄ e₃ e₂ e₁

Byte2 e₀ m₂₂ m₂₁ m₂₀ m₁₉ m₁₈ m₁₇ m₁₆

Byte3 m₁₅ m₁₄ m₁₃ m₁₂ m₁₁ m₁₉ m₉ m₈

Byte4 m₇ m₆ m₅ m₄ m₃ m₂ m₁ m₀

For the data coding byte only the sensor number and the unit have meaning.

12.13 Class=85 (0x55) Set value with zone

CLASS1.SETVALUEZONE

Description

This class mirrors the standard measurement events is CLASS1.MEASUREMENT=10 12.4 but also have zone information and is intended for setting a value instead of providing a measurement.

Byte0 Index for sensor.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 Sub-zone for which event applies to (0-255). 255 is all sub-zones.

Byte3 Normalizer byte

Byte4-7 Normalized integer (see 8 for a description).

The number of bytes in the event tells how large the normalized integer is (1,2,3 or 4 bytes).

12.14 Class=100 (0x64) Phone

CLASS1.PHONE

Description

This class is for phone related functionality.

12.14.1 Type = 0 (0x00) Undefined.

General event.

12.14.2 Type = 1 (0x00) Incoming call.

There is an incoming phone call. Usually a caller ID node just sends out numerical information. A database event 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 separate country and area-code and therefore the form

441231122334,46657413430

will also be valid.

a database connected application can later resolve 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

Byte0 Id for the call. This is an incremental identity number for each call.

Byte1 Index of phone event (base = 0). Each call can be broken up into fragments. This is the fragment number.

Byte2 Total number of events (fragments) for this call information.

Byte3-7 Caller information. Number or real text information.

12.14.3 Type = 2 (0x02) Outgoing call.

There is an outgoing phone call.

Byte0 Id for the call. This is an incremental ID number for each call. byte
1 Index of phone event (base = 0). Each call can be broken up into fragments. This is the fragment number.

Byte2 Total number of events (fragments) for this call information.

Byte3-7 Caller information. Number or real text information.

12.14.4 Type = 3 (0x03) Ring.

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

Byte0 An ID for the call. This can for instance be a number that increases by one for each call.

12.14.5 Type = 4 (0x04) Answer.

The call has been answered.

Byte0 An ID for the call. This can for instance be a number that increases by one for each call.

Byte1 Zone for answer location.

Byte2 Sub-zone for answer location.

12.14.6 Type = 5 (0x05) Hangup.

The call has been terminated by the receiving end.

Byte0 An ID for the call. This can for instance be a number that increases by one for each call.

12.14.7 Type = 6 (0x06) Giveup.

The call has been terminated by the originating end. byte

Byte0 An ID for the call. This can for instance be a number that increases by one for each call.

12.14.8 Type = 7 (0x07) Transfer.

The call has been transferred. byte

Byte0 An ID for the call. This can for instance be a number that increases by one for each call.

12.14.9 Type = 8 (0x08) Database Info.

Byte0 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 events.

Byte1 Index of phone event (base=0). Each call can be broken up into fragments. This is the fragment number.

Byte2 Total number of events (fragments) for this call information.

Byte3-7 Caller information. Real text information.

12.15 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 implemented in a module. Escape sequences

An escape sequence is preceded 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 interpreted and it is followed by the register pos. As an example %rn2 can be used to display a normalized integer that is at register pos=2 and forward. The intended use is to have actions that store data from events in registers and that they are displayed by the escape.

Second character	Description
\$	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

- 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. See 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.

12.15.1 Type = 0 (0x00) Undefined.

General event.

12.15.2 Type = 1 (0x01) - Clear Display

Clear the display on displays in a certain zone,sub-zone.

Byte0 Code - not yet defined.

Byte1 Zone.

Byte2 Sub-zone.

12.15.3 Type = 2 (0x02) - Position cursor

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

Byte0 Code - not yet defined.

Byte1 Zone.

Byte2 Sub-zone.

Byte3 Row to move to (first row is 0).

Byte4 Column to move to (first column is 0).

12.15.4 Type = 3 (0x03) - Write Display

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

Byte0 index - Increase by one for each event sent for specific text to display.

Byte1 Zone.

Byte2 Sub-zone.

Byte3-7 Display data.

Index is increased by one for each event that builds up a specific event. 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.

12.15.5 Type = 4 (0x04) - Write Display buffer

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

Byte0 index - Increase by one for each event sent for specific text to display.

Byte1 Zone.

Byte2 Sub-zone.

Byte3-7 Display data.

Index is increased by one for each event that builds up a specific event. 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 matrices buy defining a sub-zone for the user defined matrix(es).

12.15.6 Type = 5 (0x05) - Show Display Buffer

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

Byte0 index - Increase by one for each event sent for specific text to display.

Byte1 Zone.

Byte2 Sub-zone.

12.15.7 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 transferred to the display*.

Note that there are no zone and sub-zone defined for this event and the escapes must instead be chosen to be distinct in a system. This means that &p1 will be unique within a system and updating this parameter will update on all displays that has it defined.

Byte0 Display parameter index.

Byte1 Data coding byte as of VSCP Specification.

Byte2-7 Data as of coding.

Note that the event have one byte less then standard measurement events so all coding types can not be used.

12.15.8 Type = 32 (0x20) - Show Text

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

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

Byte0 Index - Increase by one for each event sent for specific text to display.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 Sub-zone for which event applies to (0-255). 255 is all sub-zones.

Byte3-7 event.

Index is increase by one for each event that builds up a specific event. 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 themselves display data or other display events. See the new_york node for examples of this.

For a multi line display one can use different sub-zones or address different lines. One can also use macro characters to map display events to a line.

12.15.9 Type = 48 (0x30) - Set LED

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

Byte0 Index

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 Sub-zone for which event applies to (0-255). 255 is all sub-zones.

Byte3 0=off, 1=on, 2=blink

Byte4 Blink period: MSB milliseconds for ON.

Byte5 Blink period: LSB milliseconds for ON.

Byte6 Blink period: MSB milliseconds for OFF.

Byte7 Blink period: LSB milliseconds for OFF.

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

12.15.10 Type = 49 (0x31) - Set RGB Color

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

Byte0 Index

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 Sub-zone for which event applies to (0-255). 255 is all sub-zones.

Byte3 Color R to display 0-255.

Byte4 Color G to display 0-255.

Byte5 Color B to display 0-255.

If multi-byte resolution for the colors is needed use index to address the byte where 0 means the MSB byte, 1 MSB+1 byte etc (Big endian).

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

CLASS1.IR

Description

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

12.16.1 Type = 0 (0x00)

Undefined

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

A RC5 remote code. <http://www.xs4all.nl/~sbp/knowledge/ir/sirc.htm> Use the VSCP abstract remote format if possible.

Byte0 RC5 code.

Byte1 RC5 Address.

Byte2 Repeat count if any.

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

A SONY remote code. <http://www.xs4all.nl/~sbp/knowledge/ir/sirc.htm> Use the VSCP abstract remote format if possible.

Byte0 SONY code.

Byte1 SONY address.

Byte2 Repeat count if any.

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

Packed LIRC codes code. LIRC Codes are normally sent as 64-bit codes or even larger codes. Only codes with a length less than 56 bits (7-bytes) are supported by VSCP and the most significant byte of the LIRC code is not transferred. <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.

12.16.5 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.

12.16.6 Type = 49 (0x31) MAPito Remote Format.

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

Byte0 Repeat Count.

Byte1 Zone for which event applies to (0-255). 255 is all zones.

Byte2 Sub-zone for which the event applies to (0-255). 255 is all sub-zones.

Byte3 Control address MSB.

Byte4 Control address.

Byte5 Control address.

Byte6 Control address LSB.

Byte7 Key Code.

12.17 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.

12.17.1 Type = 0 (0x00) Undefined.

General one wire.

12.17.2 Type = 1 (0x01) New ID

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

Byte0-7 1-Wire ID with MSB first.

12.17.3 Type = 2 (0x02) Convert.

Command a 1-wire node to do a conversion.

Byte0-7 1-Wire ID with MSB first.

12.17.4 Type = 3 (0x03) Read ROM.

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

Byte0-7 1-Wire ID with MSB first.

12.17.5 Type = 4 (0x04) Match ROM.

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

Byte0-7 1-Wire ID with MSB first.

12.17.6 Type = 5 (0x05) Skip ROM.

Execute a 1-wire Skip ROM command.

12.17.7 Type = 6 (0x06) Search ROM.

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

Byte0-7 1-Wire ID with MSB first.

12.17.8 Type = 7 (0x07) Conditional Search ROM.

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

Byte0-7 1-Wire ID with MSB first.

12.17.9 Type = 8 (0x08) Program.

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

Byte0-7 1-Wire ID with MSB first.

12.17.10 Type = 9 (0x09) Overdrive skip ROM.

Execute a 1-wire Overdrive Skip ROM command.

12.17.11 Type = 10 (0x0A) Overdrive Match ROM.

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

Byte0-7 1-Wire ID with MSB first.

12.17.12 Type = 11 (0x0B) Read Memory.

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

Byte0-7 1-Wire ID with MSB first.

12.17.13 Type = 12 (0x0C) Write Memory.

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

Byte0-7 1-Wire ID with MSB first.

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

CLASS1.X10

Description

X10 Protocol functionality.

12.18.1 Type = 0 (0x00) Undefined.

General event.

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

Byte0 Header/Code.

Byte1 Address or Function byte.

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.

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

Bit	Description
0	Function Code
1	Function Code
2	Function Code
3	Function Code
4	House Code
5	House Code
6	House Code
7	House Code

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

Byte0 Header Code (Always 0x03)

Byte1 Function.

Byte2 Unit Code.

Byte3 Data.

Byte4 Command.

where

Bit	Description
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.

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

Byte0 Node address.

Byte1 Header/Code.

Byte2 Address or Function byte.

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

12.18.5 Type = 4 (0x04) X10 Extended Message Send.

Byte0 Node address

Byte1 Header Code (Always 0x03)

Byte2 Function.

Byte3 Unit Code.

Byte4 Data.

Byte5 Command.

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

12.18.6 Type = 5 (0x05) Simple x10 message .

Byte0 House code (Required).

Byte1 Unit code (ignored if not relevant such as All lights off).

Byte2 Command (bright, dim, on, off etc).

Byte3 Dim level (if needed).

Byte4 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.

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

CLASS1.LON

Description

LON Works functionality.

12.19.1 Type = 0 (0x00) Undefined.

General Event.

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

CLASS1.EIB

Description

EIB functionality.

12.20.1 Type = 0 (0x00) Undefined.

General Event.

12.21 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/> or in this document <http://www.hth.com/filelibrary/pdffiles/snap.pdf>

12.21.1 Type = 0 (0x00) Undefined.

General Event.

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

CLASS1.CBUS

Description

Control and interface for the CBUS protocol

12.22.1 Type = 0 (0x00) Undefined.

General event.

12.23 Class=206 (0xCE) Position (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 (0x33) Altitude.
- For Speed use CLASS1.MEASUREMENT, Type = 32 (0x20) Speed
- For Signal Quality use CLASS1.DATA, Type = 5 (0x05) Signal Quality
- For Time-stamp use CLASS1.MEASUREMENT, Type = 4 (0x04) 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 events together.

12.23.1 Type = 0 (0x00) Undefined.

General event.

12.23.2 Type = 1 (0x01) Position.

Position information as decimal Latitude + Longitude.

Byte0-3 Latitude as floating point value.

Byte4-7 Longitude as floating point value.

12.23.3 Type = 2 (0x02) Satellites.

Number of satellites used.

Byte0 # of satellites used.

12.24 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 (0x25) Token Activity.
- For IMSI code use CLASS1.INFORMATION, Type = 37 (0x25) Token Activity .
- For Signal Quality use CLASS1.DATA, Type = 5 (0x05) Signal Quality
- For Time-stamp use CLASS1.MEASUREMENT, Type = 4 (0x04) Time

12.24.1 Type = 0 (0x00) Undefined.

General event.

12.24.2 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.

Byte0-3 GSM Cell ID.

12.25 Class=509 (0x1FD) Logging i/f

CLASS1.LOG

Description

Logging functionality.

12.25.1 Type = 0 (0x00) Undefined.

General Log event.

12.25.2 Type = 1 (0x01) Log event.

Message for Log. Several frames have to be sent for a event that take up more than 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.

Byte0 ID for event.

Byte1 Log level for message.

Byte2 Idx for this message.

Byte3-7 Message.

12.25.3 Type = 2(0x01) Log Start.

Start logging.

Byte0 ID for log.

12.25.4 Type = 3 (0x03) Log Stop.

Stop logging.

Byte0 ID for log.

12.25.5 Type = 4 (0x04) Log Level.

Set level for logging.

Byte0 ID for log byte 0 Log level.

12.26 Class=510 (0x1FE) Laboratory use

CLASS1.LABORATORY

Description

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

12.26.1 Type = 0 (0x00) Undefined.

General event.

12.27 Class=511 (0x1FF) Local use

CLASS1.LOCAL

Description

This event 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.

13 Level II Events

A class in Level II is described by a number between 512-65535. Instead of writing a number the class can be described as CLASS2.XXXX indicate a specific (XXXX in this case) Level II class. Also the form CLASS2.XXXX=yy can be used where yy is the numerical form.

Class definitions can be found in the header file *vscp_class.h* which is located located in the src/vscp/common folder. In the same folder *vscp_type-h* can be found which contains defines for types. Also the *vscphelper.h/cpp* files contains stuff that are useful for handling classes/types as a programmer.

13.1 Class=512 (0x200) ... 1023 (0x3FF) - Level II Mirror Level I events

CLASS2.PROTOCOL1

Description

Class 512-1023 is 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.PROTOCOL1 the first 16 bytes of the data field 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 addressed 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 from the device will go out 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. If you send 512-events with an external node as destination GUID (which is not the GUID of an interface) the event will be put through all interfaces.

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.
```

Byte 0-15: GUID for Interface where node is located.

Byte 16: Old nickname for node.

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 verify 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.*

Byte 0-15: GUID for Interface where node is located. LSB is nickname for node.

Byte 16: Nickname for node.

Byte 17: Register to read.

Response is

*Class = 0 = Level I Protocol, Type=10 (0x0A)
Read/Write response.*

Byte 0: Register read/written.

Byte 1: Content of register.

13.2 Class=1024 (0x400) - Level II Protocol Functionality

CLASS2.PROTOCOL

Description

For Level I events class=0 defines protocol control functionality. All events 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 event 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 events a little easier. The following events have been added to the level II control events to support configuration management.

13.2.1 Type = 0 (0x0000) Undefined.

General event.

13.2.2 Type = 1 (0x0001) ReadRegister

Read a Level II register

Byte0-15 Contains the GUID of the target node (MSB->LSB).

Byte16-19 Register to read (or start index), (MSB->LSB).

Byte20-21 Number of registers to read (max 487).

Number of registers to read can also be restricted by the buffer size on the board (register 0x98). If this register is set to something else then 0 (default) this is the max size for data.

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

13.2.3 Type = 2 (0x0002) WriteRegister

Byte0-15 Contains the GUID of the target node (MSB->LSB).

Byte16-19 Register to write (or start index), (MSB->LSB).

Byte20... Data to write to register(s).

Number of registers to write can also be restricted by the buffer size on the board (register 0x98). 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.

13.2.4 Type = 3 (0x0003) ReadWriteResponse

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.

Byte0-3 Start register for register read/written.

Byte4... Data read/written.

13.2.5 Class=1025 (0x401) Level II Control

CLASS2.CONTROL

Description

Level II Control functionality.

13.2.6 Type = 0 (0x0000) Undefined.

General event.

13.3 Class=1026 (0x402) Level II Information

CLASS2.INFORMATION

Description

Level II Information events.

13.3.1 Type = 0 (0x0000) Undefined.

General event.

13.3.2 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.

Byte0 Event code.

Byte1 Zone.

Byte2 Sub-zone.

Byte3 Token device type code

Byte 4-7 Reserved.

Byte8.. Unique ID of token-device.

	Code	Description
Event codes	0	Touched and released.
	1	Touched.
	2	Released.
	3-254	Reserved.
	255	Error.

Token device type	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
	20	GSM IMSI code (International Mobile Subscriber Identity) packed in 64 bits. http://en.wikipedia.org/wiki/IMSI
	21-255	Reserved.

13.4 Class=1027 (0x404) Level II Mirror Level II Text to speech

CLASS2.TEXT2SPEECH

Description

This is an interface that translates text to speech

13.4.1 Type = 0 (0x0000) Undefined.

General event.

13.4.2 Type = 1 (0x0001) Talk

Byte0-1 Zone to talk in.

Byte2-3 Sub-zone to talk in.

Byte4 Relative volume (0-100%).

Byte5-7 Reserved.

Byte9 Text to speak.

13.5 Class=1029 (0x405) Level II Custom

CLASS2.CUSTOM

Description

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

13.5.1 Type = 0 (0x0000) Undefined.

General event

13.6 Class=1030 (0x406) Level II Display

CLASS2.DISPLAY

Description

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

13.7 Class=1040 (0x410) Measurement string

CLASS2.MEASUREMENT_STR

Description This is a mirror of CLASS1_MEASUREMENT where the data is in string form. The zero terminated string has the following form “*sensor-index, value-unit, zone, subzone, value*” where the value is a floating point value in string form and the sensor-index and value-unit is the same as for CLASS1_MEASUREMENT but where sensor index’s can be in the range 0-255 and value-unit in the range 0-3.

Byte0 Index for sensor, 0-255.

Byte1 Zone, 0-255.

Byte2 Subzone, 0-255.

Byte3 Unit from measurements, 0-3.

Byte4...n String up to the maximum data size.

13.8 Class=65535 (0xFFFF) VSCP Daemon internal events

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.

13.8.1 Type = 0 (0x0000) Undefined.

General event.

13.8.2 Type = 1 (0x0001) Loop

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

13.8.3 Type = 3 (0x0003) Pause

The machine/daemon is going to a pause state. No data is defined.

13.8.4 Type = 4 (0x0004) Activate

The machine/daemon is going from a pause state. No data is defined.

13.8.5 Type = 5 (0x0005) Second

Event is generated each new second. No data is defined.

13.8.6 Type = 6 (0x0006) Minute

Event is generated each new minute. No data is defined.

13.8.7 Type = 7 (0x0007) Hour

Event is generated each new Hour. No data is defined.

13.8.8 Type = 8 (0x0008) Noon

Event is generated at 12:00 each day. No data is defined.

13.8.9 Type = 9 (0x0009) Midnight

Event is generated at 00:00 each day. No data is defined.

13.8.10 Type = 11 (0x000B) Week

Event is generated when a new week starts (config setting). No data is defined.

13.8.11 Type = 12 (0x000C) Month

Event is generated when a new month starts. No data is defined.

13.8.12 Type = 13 (0x000D) Quarter

Event is generated when a new quarter starts. No data is defined.

13.8.13 Type = 14 (0x000E) Year

Event is generated when a new year starts. No data is defined.

13.8.14 Type = 15 (0x000F) random-minute

Event is generated randomly over a minute as is sent once each minute. No data is defined.

13.8.15 Type = 16 (0x0010) random-hour

Event is generated randomly over an hour and is sent once each hour. No data is defined.

13.8.16 Type = 17 (0x0011) random-day

Event is generated randomly over a day and is sent once each day. No data is defined.

13.8.17 Type = 18 (0x0012) random-week

Event is generated randomly over a week and is sent once each week. No data is defined.

13.8.18 Type = 19 (0x0013) random-month

Event is generated randomly over a month and is sent once each month. No data is defined.

13.8.19 Type = 20 (0x0014) random-year

Event is generated randomly over a year and is sent once each year. No data is defined.

13.8.20 Type = 21 (0x0015) Dusk

Event is from calculated dusk. No data is defined.

13.8.21 Type = 22 (0x0016) Dawn

Event is from calculated dawn. No data is defined.

13.8.22 Type = 23 (0x0017) Starting up

The machine/daemon is starting up. This is the first event sent after a machine start up. Shutting down . No data is defined.

13.8.23 Type = 24 (0x0018) Shutting down

The machine/daemon is shutting down. This is the last event sent before a machine is shutting off. No data is defined.

13.8.24 Type = 25 (0x0019) Timer started

A timer has been started.

Argument is timer ID and start time for the timer.

Byte0 32 bit timer ID. MSB.

Byte1 32 bit timer ID.

Byte2 32 bit timer ID.

Byte3 32 bit timer ID. LSB.

Byte4 Start time in milliseconds as a 32-bit value. MSB.

Byte5 Start time in milliseconds as a 32-bit value.

Byte6 Start time in milliseconds as a 32-bit value.

Byte7 Start time in milliseconds as a 32-bit value. LSB.

Max timer value is about 49 days.

13.8.25 Type = 26 (0x001A) Timer paused

A timer has been paused.

Argument is timer ID and time when timer was paused.

Byte0 32 bit timer ID. MSB.

Byte1 32 bit timer ID.

Byte2 32 bit timer ID.

Byte3 32 bit timer ID. LSB.

Byte4 Start time in milliseconds as a 32-bit value. MSB.

Byte5 Start time in milliseconds as a 32-bit value.

Byte6 Start time in milliseconds as a 32-bit value.

Byte7 Start time in milliseconds as a 32-bit value. LSB

13.8.26 Type = 27 (0x001B) Timer resumed

A timer has been restarted.

Argument is timer ID and time when timer was restarted.

Byte0 32 bit timer ID. MSB.

Byte1 32 bit timer ID.

Byte2 32 bit timer ID.

Byte3 32 bit timer ID. LSB.

Byte4 Start time in milliseconds as a 32-bit value. MSB.

Byte5 Start time in milliseconds as a 32-bit value.

Byte6 Start time in milliseconds as a 32-bit value.

Byte7 Start time in milliseconds as a 32-bit value. LSB.

13.8.27 Type = 28 (0x001C) Timer stopped

A timer has been stopped.

Argument is timer ID and time when timer was stopped.

Byte0 32 bit timer ID. MSB.

Byte1 32 bit timer ID.

Byte2 32 bit timer ID.

Byte3 32 bit timer ID. LSB.

Byte4 Start time in milliseconds as a 32-bit value. MSB.

Byte5 Start time in milliseconds as a 32-bit value.

Byte6 6 Start time in milliseconds as a 32-bit value.

Byte7 Start time in milliseconds as a 32-bit value. LSB.

13.8.28 Type = 29 (0x001D) Timer Elapsed

A timer has elapsed.

Argument is timer ID.

Byte0 32 bit timer ID. MSB.

Byte1 32 bit timer ID.

Byte2 32 bit timer ID.

Byte3 32 bit timer ID. LSB.

Part II

Software

14 The VSCP Daemon

The VSCP daemon is a server program that collects information from many drivers and lets clients collect this information over a TCP/IP interface. The daemon also has an internal scheduler that makes it possible to control different scenarios such as simple tasks like automatically turn on a group of lamps at a special time of the day to complex industrial control situations.

On Unix systems the daemon is a standard server application that is started in the background. It is possible also to compile it to start as a standard application if that is needed.

On Windows the daemon is available as a standard Windows service and as a standard application. Normally the standard application is not used, but for debugging and testing the application can be useful.

The daemon uses the CANAL interface to talk to drivers. This means that a driver normally is at VSCP Level I. However, a special driver can be implemented that just needs to implement three methods of the CANAL API (Open,Close and CanalGetLevel).

If the CanalGetLevel call returns CANAL_LEVEL_USES_TCPIP the daemon will assume this is a Level II driver that will use the TCP/IP interface of the daemon for all of its communication and therefore not communicate any further with it until it is time to close it down.

On Unix if you start vscpd from your shell prompt you will usually get back immediately to the prompt. Often people think that the program has died. But this is not an error. vscpd is a daemon. Daemons always run in background.

There is a video on you tube that takes you through some of the essentials of the VSCP Daemon. <http://www.youtube.com/watch?v=9HY2br8ASvo>

14.1 Setting up the system

14.1.1 VSCPD for Windows - run as service

About The vscpservice is a WIN32 program that runs the VSCP daemon as a WIN32 service. This service is installed as a service by the setup program.

The VSCP Service works exactly the same as the daemon but runs as a standard service. This means the control panel applet to handle windows services can be used to control how it behaves. The executable is called vscpservice.exe.

The service can be installed manually or with its own installation mechanism

Command line switches available are

Switch	Description
no switch	Start the service
-i	Install the service
-u	Uninstall the service
-v	Display service information
-? or -h	Display information

After installing the service the obvious location to control it is the control panel applet designed for this.

Manually installing the services If you want to install(de-install) the service manually follow the steps below

The service can be installed with (this is done automatically by the WIN32 setup program)

vscpservice -i

Uninstalled with

vscpservice -u

More info about different switches and the available options can be retrieved by

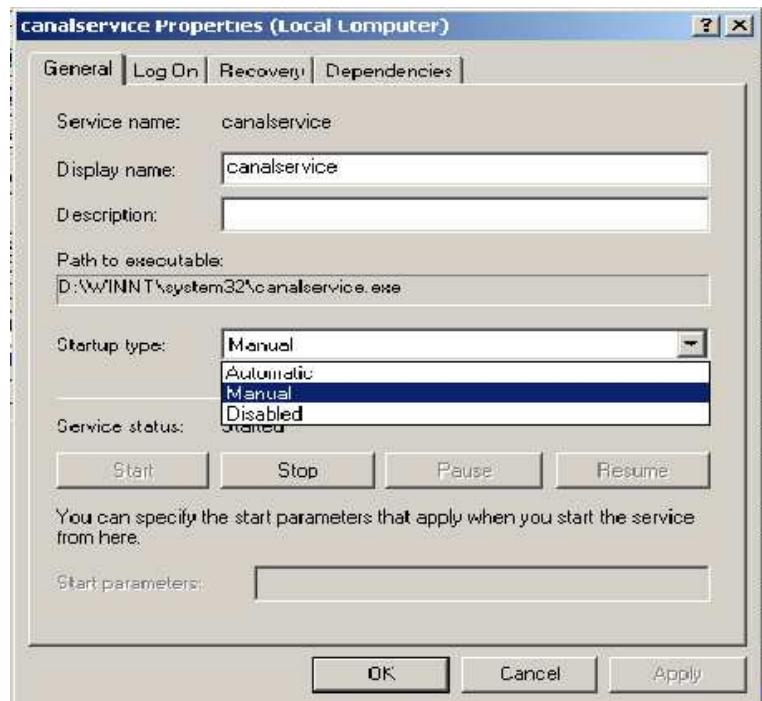
vscpservice -? or vscpservice -h

The service has a variable called start at

HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\vscpservice
that should be set to 2 (autostart). It is set to 3 (manual) after installation so that you have the option to select which server/service to use.

This can also be done in the service applet

Configure the service to run on your machine This can also be done in the service applet



Where you select automatic if you want the service to start automatically.

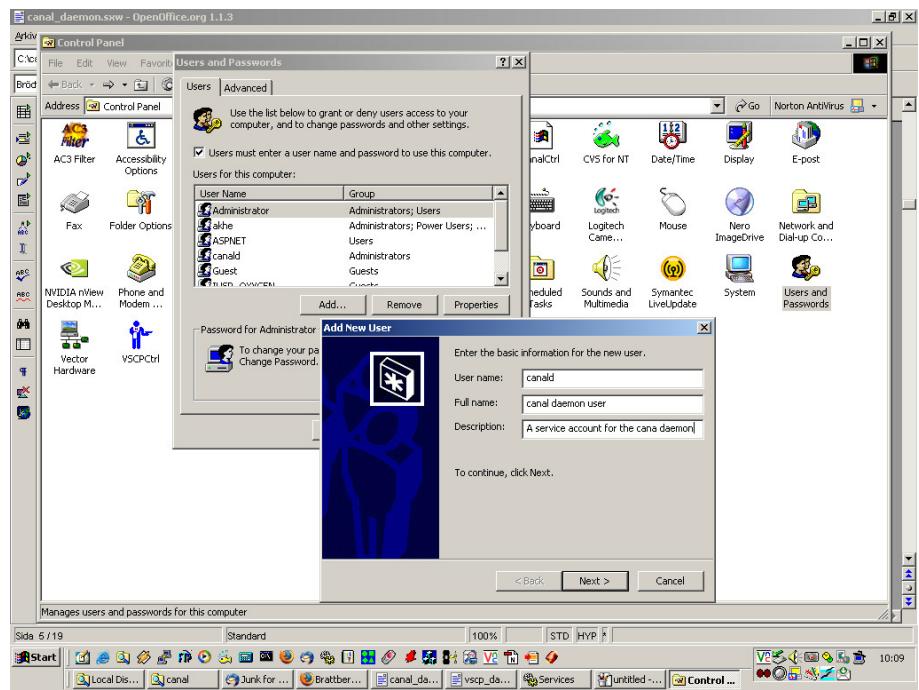
The installation program installs the service and marks it to be run automatically. You have to add the user the service runs under manually.

vscpservice needs to be run as a user belonging to the administrator group. Therefore you should add a user "canald" and add this user to the administrators group.

You do this by selecting the Users and Passwords applet in the control panel



and adding the user.



After hitting next and entering a password (recommended) this window is shown

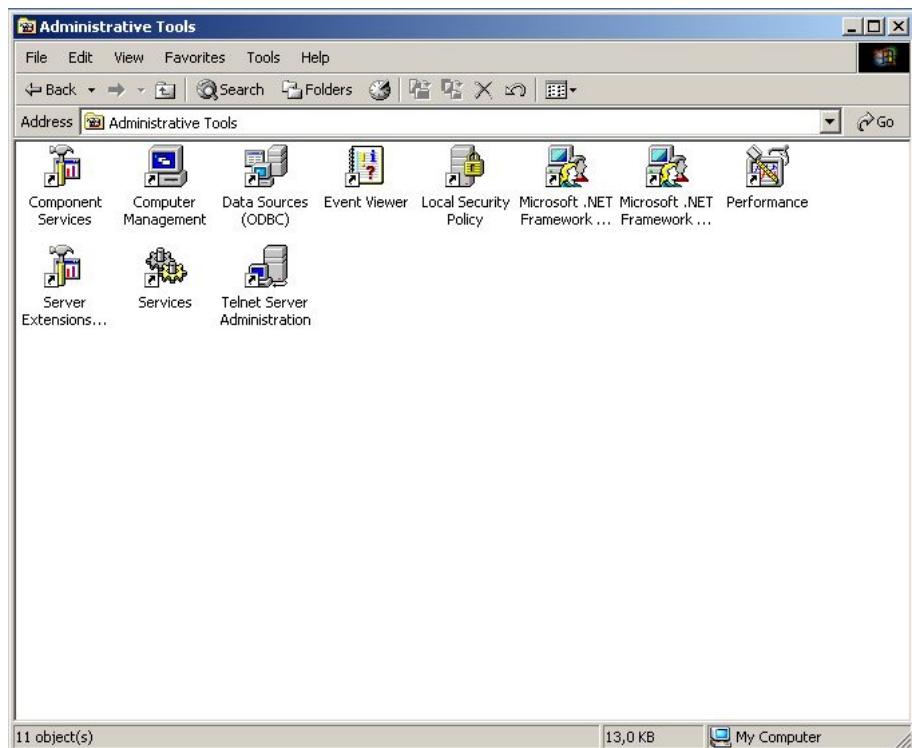


Select others and “Administrators”

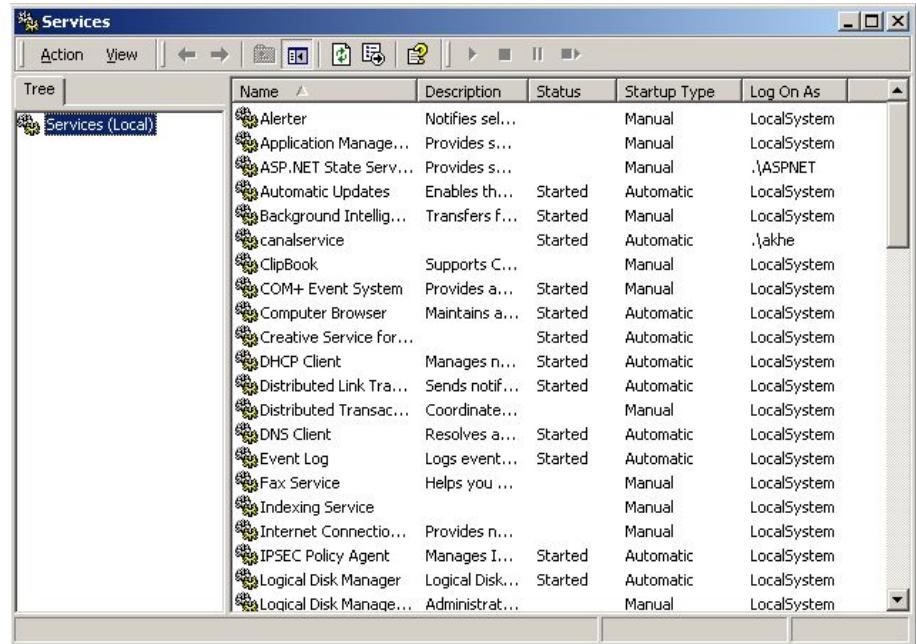
After doing that you should open the service applet under the Administrative Tools in the control panel



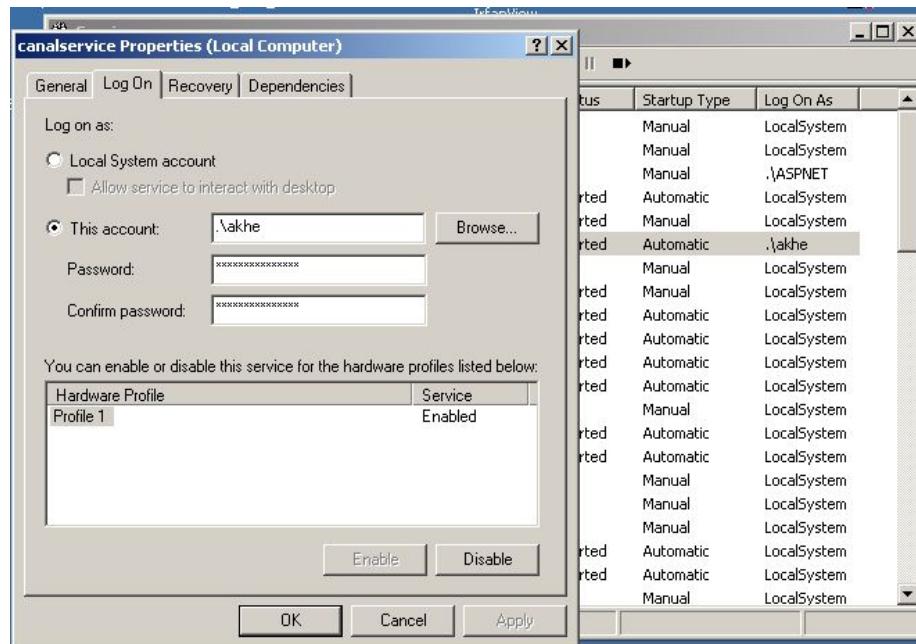
You select the services applet here



and this window will show up

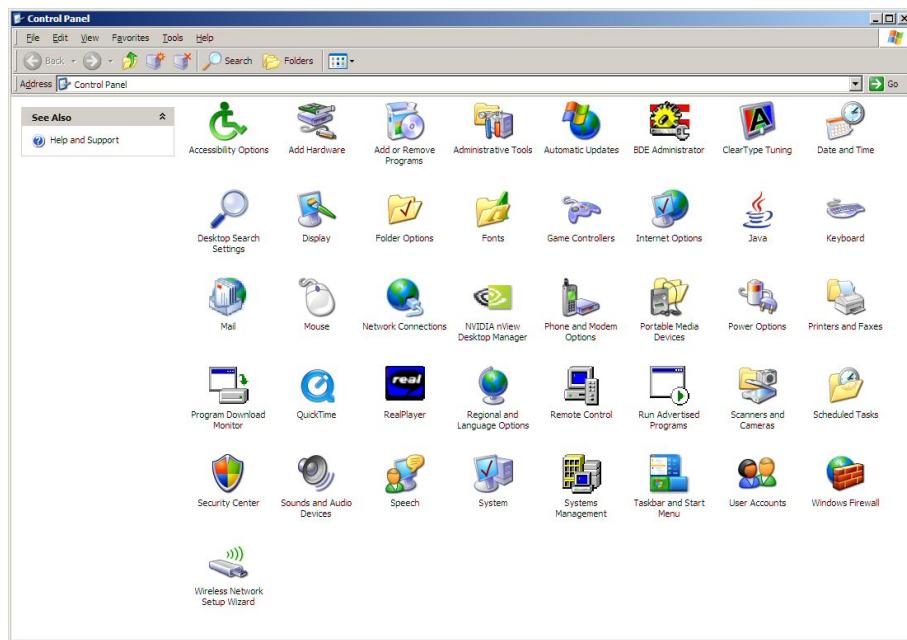


Right click the vscpservice item and select properties. Select the **Logon tab**

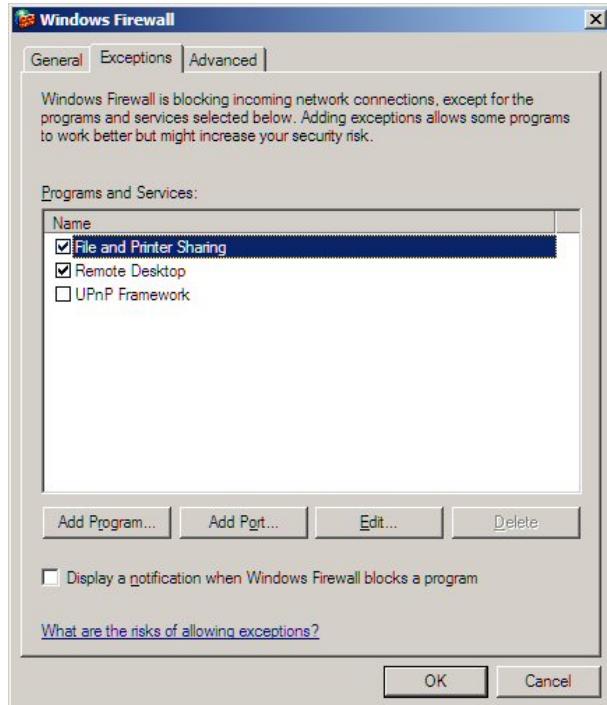


Mark "This account" and select the user you added above.
Now you can start the service in the service control applet.

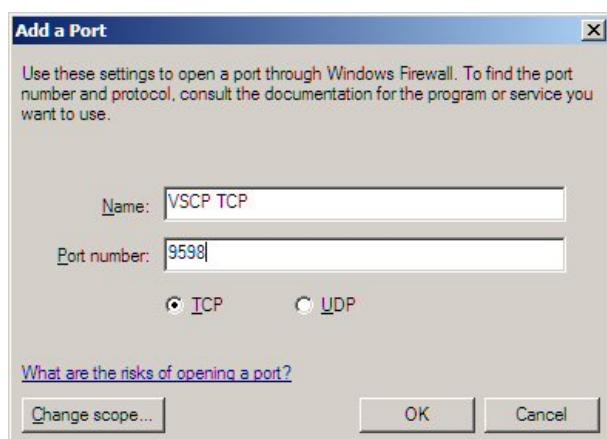
Enabling VSCP ports on the XP firewall Normally the firewall is (and should be) enabled on Windows XP. This means that the port used by VSCP (9598) is blocked for both TCP and UDP traffic. You must enable this port in the firewall to make everything working.



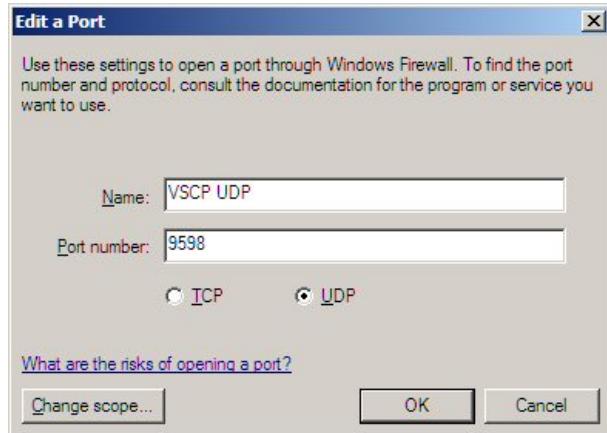
The firewall settings is found in the control panel



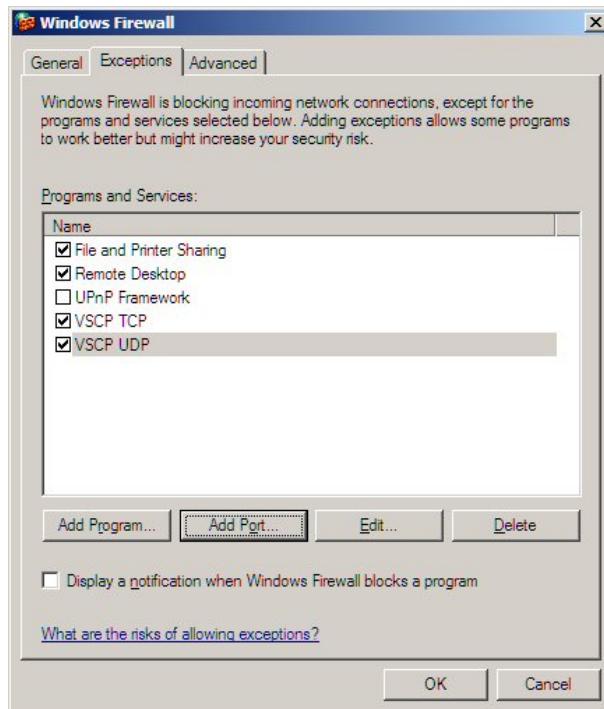
Add the ports in the exceptions tab and select add port...



Allow the TCP port.



Allow the UDP port.



This is how it should look like when both the ports has been enabled.

14.2 Windows setup & configuration

The VSCP daemon (vscpd.exe in the installation folder) is run as an ordinary application (console application). The application will normally open a console

window when run, in this window information is printed about what the daemon is working on and what command users issue.

To close the application just click on the close window icon in the top right corner of the console window.

The daemon needs a configuration file just as the service does. This configuration file should be in the file folder

| Documents and Settings | All users | Application Data | vscp

and it is called

vscpd.conf

normally the installation program will create a default file. The file is described here 14.4 on the next page.

Some switches are available that control how the daemon is started.

Short switch	Long switch	Description
-h	--help	Give information about available switches and what they do
-g	--gnu	Gives copyright information.
-v	--verbose	Give extra information useful for debugging
-c	--configpath	Tell where the program should look for the configuration file. Default is to look for this information in the general “application data” folder in a folder called vscp. Typically this is c:/documents and settings/all users/application data/vscp/vscpd.conf
-d	--debuglevel	Set the debug level (0-9). Higher value more output. Default=0
-w	--hide	Hide the console window. Can be useful if running the app. in a “service way”.

14.3 Linux setup & configuration

The daemon is the central program that do VSCP work on your box. It loads drivers for different devices and it present a TCP/IP interface from which it is possible to control the daemon and read/write events to devices.

On Linux the vscp daemon is started as any other daemon. Example startup scripts is available in the root folder.

Switch	Description
-c	Path to configuration file. Default is /etc/vscp/vscpd.conf
-s	Don't run as a daemon.
-d	Debug level 0-9. Higher value means more information. Default is 0, no debug information.
-g	Displays copyright information.
-h	Displays help information.

There are two configuration files that the daemon will load upon startup. /etc/vscp/vscpd.conf for general configuration parameters and /etc/vscp/variables.xml that hold persistent variable declarations. variables.xml should be writable by the daemon.

When you type

make install

(or install one of the binary distributions) the daemon executable will be copied to /usr/local/bin and the configuration file will be copied to /etc/vscp and a default configuration file vscpd.conf will be copied to this folder. You should edit this file to fit your needs as it is described here 40_vscpd_config_file.

To actually install the daemon as a self starting server you need to create a startup script for it in /etc/init.d. Sample scripts are available in the root of the source tree.

If you want to debug the daemon there is wxTrace enablers in the controlobject.cpp file. Just uncomment any of the wxLog::AddTraceMask in the CControlObject::init method to get the debug events you want.

14.4 Configuration file walk through

The full configuration file format is described here 40_vscpd_config_file.

14.4.1 General settings

In the <general> tab area settings that affect the behavior of the server in general is located.

<loglevel>n</loglevel> Set loglevel n in range 0-8 where 0 is no logging and 8 is debug mode. Default is 1. “none”=0, “info”=1, “notice”=2, “warning”=3, “error”=4, “critical”=5, “alert”=6, “emergency”=7 and “debug”=8 can be used for convenience instead of numerical representation.

<tcpif enable="false|true" port="9598" port="n" ifaddress="ip4-address" />

- *enable* - Enable/disable daemon TCP/IP interface. Default is enabled.
- *port* - Set port that the daemon listens on. Default is 9598.
- *ifaddress* - Set the address the interface binds to on a machine with multiple interfaces. If no address is given the tcp/ip interface will be available on all interfaces (this is the default). If an address is given the daemon listens only on that interface. Default is not set.

<canaldriver enable="true|false" /> Enable/disable daemon driver interface. If disabled no drivers will be loaded. Default is enabled.

<vscp enable="true|false" /> Enable/disable daemon internal VSCP functionality. If disabled the server will no longer react on events like the High end server probe etc. Default is enabled.

<dm enable="true|false" path="/srv/vscp/dm.xml" /> Enable/disable the internal decision matrix of the daemon. Default is enabled. The path should always be given. The standard place for the decision matrix should be */srv/vscp/dm.xml* but can be any place on storage as long as the location is writable by the VSCP daemon.

<variables enable="true|false" path="/srv/vscp/variables.xml" /> Enable/disable the internal variable handling of the daemon. Default is enabled. The path should always be given. The standard place for the persistent variable file should be */srv/vscp/variables.xml* but can be any place on storage as long as the location is writable by the VSCP daemon.

guid Set the server GUID for the daemon. This GUID is the base for all interface GUID's in the system. Interface GUID's is formed like xx:xx:xx:xx:xx:xx:xx:xx:xx:xx:xx:aa:aa:bb:bb where xx is guid set here, aa is the interface id and bb is nickname id's for devices on that interface.

If not set here a GUID will be formed from the (first) MAC address of the machine the daemon runs on (on Linux needs to be run as root to get this) or if this fails the local IP address is used to form the GUID. The GUID string should be in MSB -> LSB order and have the four lsb set to zero.

Example

FF:EE:DD:CC:BB:AA:99:88:77:66:55:44:00:00:00:00

clientbuffersize This is the default buffer size (number of events) for all clients in the system. Everything from a driver to a TCP/IP user is regarded as a client. Default is 1024.

webrootpath Path to daemon root filesystem. The daemon should have read/write permissions to this area. Webserver and websocket interface look for content here.

pathmimetypes Path to XML file with mime types the webserver understands. On a memory constraint platform it is important just to enable the mimetypes that is needed not to waste precious memory. Theformat for the file is like this

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

```

<mimetype enable="true" extension="3dm" mime="x-world/x-3dmf" />
<mimetype enable="true" extension="3dmf" mime="x-world/x-3dmf" />
...
</mimetypes>
```

set enable to “false” for entries that you don’t want to use or remove the lines all together.

<websockets enable=”true|false” port=”7681” /> Control websocket interface. The websocket interface can be used on its own as it has limited capabilities to also server web-pages or together with the full webserver.

- *port* - Port used for websockets. Default is 7681.
- *enable* - Enable or disable the websocket interface.

<webserver enable=”true|false” port=”8080” /> Enable full web server.

- *port* - Port used for full webserver. Default is 8080.
- *enable* - Enable or disable the full webserver interface.

pathcert Path to ssl certificate.

pathkey Path to ssl key.

14.4.2 Remote user settings

In the <remoteuser> tab are settings that affect which users that can access the daemon through the TCP/IP interface and through the webinterface. Each user is defined between a <user>...</user> pair.

name Username for this user.

password Password ad an md5 checksum for this user. The mkpasswd can be user to generate passwords.

privilege Users can have privileges from 0-15 that allow them to do command that are equal to or less then the set privilege level. Enter a privilege level as 0-15 here or in symbolic form as admin (==15) or user (==4). Default is 4.

allowfrom This is a comma separated list with host addresses this user is allowed to log in to the server from. wild cards can be used to indicate “all”. This mean that

```
*.*.*.*"
```

means a user can log in from all remote machines.

```
194.*.*.*
```

that he/she can log on from all IP addresses that starts with 194 and son on. Default is all so if you want to allow everyone to access your server you can skip this tag.

allowevent This is a comma separated list of events this user is allowed to send. The form is

```
class:type, class:type
```

wild-cards (*)can be used for both. This means that 20:*,30:* means that the user is allowed to send all events in class=20 and class=30. Default allows user to send all events.

mask Set the mask for incoming events to a client. The form is

```
priority="xxx" class="xxx" type="xxx" GUID="xxx">
```

The mask has a bit set for a binary digit that is of interest. Default is all is null == disabled.

filter

```
priority="xxx" class="xxx" type="xxx" GUID="xxx">
```

This is the event filter</filter>

Set the filter for incoming events to a client. The form is

```
class="xxx" type="xxx" GUID="xxx"
```

If a mask bit is set to one for a position then the filter bit must be equal to the bit of the incoming event for the client to get it.

14.4.3 automation

In the <automation> tab settings that affect the internal VSCP functionality of the server is located.

zone This is the zone for the server 0-254.

sub-zone This is the sub-zone for the server 0-254.

longitude Longitude for place where daemon is located in decimal form.

latitude Latitude for place where daemon is located in decimal form.

time Enable/disable the periodic CLASS1.PROTOCOL, Type=1 (Segment Status Heartbeat) event to be sent from the daemon. The interval between events is set in seconds.

```
<time enable="true/false" interval="seconds" />
```

sunrise Enable/disable the CLASS1.INFORMATION, Type=44 (Sunrise) to be sent. Longitude and latitude must be set for this to work correctly.

```
<sunrise enable="true/false" />
```

sunset Enable/disable the CLASS1.INFORMATION, Type=45 (Sunset) to be sent. Longitude and latitude must be set for this to work correctly.

```
<sunset enable="true/false" />
```

heartbeat Enable/disable the CLASS1.INFORMATION, Type=9 (Node Heartbeat) to be sent. The interval between events is set in seconds.

```
<heartbeat enable="true/false" interval="seconds" />
```

14.4.4 VSCP level I driver (formerly known as CANAL driver)

In the <canaldriver> tab there are settings and information for loadable Level I drivers. Tags are located between <driver>...</driver> pairs.

```
<canaldriver> <!-- Information about CANAL drivers -->

<driver enable="true|false">
  <name>logger</name>
  <config>param2 ; param2 ; param3 ; ... </config>
  <path>path_to_driver</path>
  <flags>1</flags>
  <guid>00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00</guid>
</driver>

</canaldriver>
```

enable The driver can be enabled/disabled by setting this property to true/false. Convenient if one want to disable a driver without removing it from the configuration file.

name This is the name of the driver.

config This is the driver option string.

path This is the path to the driver dll/dl(so).

flags This is the driver flags. A 32-bit numerical value.

guid This is the GUID for the driver. If not set, the client GUID which is generated from the daemon GUID, is used.

14.4.5 VSCP level II driver (former known as VSCP driver)

Drivers are specified within a <vscpdri</> tag

```
<vscpdri>    <!-- Information about Level II VSCP drivers -->

    <driver enable="true|false">
        <name>t est</name>
        <path>path_to_driver</path>
        <config>param2 ; param2 ; param3 ; ... </ config>
        <guid>00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00</ guid>
    </driver>

</vscpdri>
```

enable The driver can be enabled/disabled by setting this property to true/false. Convenient if one want to disable a driver without removing it from the configuration file.

name This is the users name for the driver. This name is used in the tcp/ip interface lists to identify the driver to users. The string is added in front of the variable name for which data is fetched by the driver from the daemon in order to configure it. For example if the driver has a variable called *_path* the actual variable fetched will be *prefix_path*. That is if the prefix is set to “logger1” the value will be fetched for the variable named *logger1_path*. This way the same driver can be configured from different variables.

path This is the path to the driver dll/dl(so) on the system where the daemon is running.

config This is the driver option string. Normally a semicolon separated list on the form *variable-name-without-prefix1;variable-name-without-prefix2;.....* The VSCP daemon will send this string to the driver with a *username;password;prefix* as the config argument. This is the logon information for the driver when it connects to the client interface and prefix used by the driver when it retrieves/writes/creates variables.

guid This is the GUID for the driver. If not set, the client GUID which is generated from the daemon GUID, is used.

14.4.6 interface

This section holds information about each TCP/IP interface the daemon service. Normally no need to enter this information.

port The TCP/IP port to use for the interface. Default is 9598.

ipaddress IP address to use for interface.

macaddress MAC address for the Ethernet card where interface is. Entered as a six digit hex number on the form

aa:bb:cc:dd:ee:ff

guid GUID for the interface.

14.5 VSCP Daemon Control Interface

This is documentation for the TCP interface of the daemon. The TCP/IP interface is a very powerful tool for users that allow for full control of the functionality of the daemon.

Originally the TCP/IP interface was added to allow for a more secure way to send events over TCP/IP links. This interface will have a fair better chance to work over wireless and other links that are hard to get to work 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.

14.5.1 GUID assigned to the interface

Just as with all clients that connect to the daemon each TCP client interface gets a GUID assigned to it. This GUID is assigned by setting it to the GUID the server was assigned at start up (in its configuration file 40_vscpd_config_file). This means that a client GUID has the following general form

yy client_id_MSB client_id_LSB
0 0

where the the assigned GUID is represented by yy and client_id is a system unique ID for this client interface.

If no GUID is assigned in the configuration file the Ethernet MAC address of the computer vscpd is running on using the VSCP Ethernet assignment method (see Assigned GUID's). This results in a GUID of the following form

FF FF FF FF FF FF FF FE yy yy yy yy yy client_id_MSB client_id_LSB
0 0

where yy is the four most significant bytes of the Ethernet address and FF is decimal 255. Note that as the two least significant bytes of the MAC address is dropped there is a chance for duplicated GUID's in a network with more than one machine with a network card from the same manufacturer. In this case set a unique GUID in the configuration file.

Another possibility is to use the IP address of the computer

```
FF FF FF FF FF FF FF FD yy yy yy yy client_id_MSB client_id_LSB
0 0
```

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

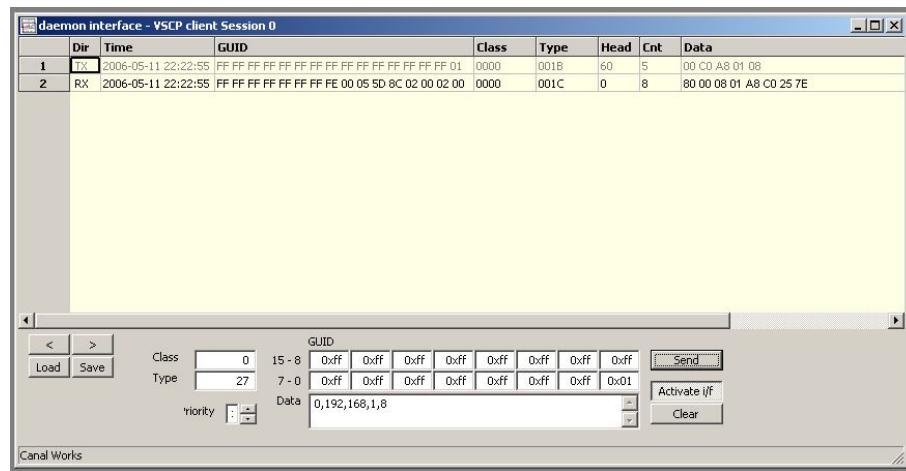
The client_id is present in each VSCP event data structure and if a client uses more than one connection to the daemon this client_id can be used to detect events that it has sent itself when they arrive on the other open interfaces. This is typical used when the RCVLOOP command is issued where typically one interface is used to send events and another is used to receive events. By saving the client_id for the transmitting channel events that is sent on this channel can be detected on the receive channel.

When an event is sent from the driver interface (a CANAL event) the nickname-ID is stored in the LSB GUID byte (GUID[0]).

14.5.2 Server discovery

A node that needs a TCP connection to a server broadcasts 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 other cases. The server configuration should have control on how it is handled.



As an example: In this picture VSCP Works 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 bit-filed 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 has 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.

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

A node can react in its own manner on the response. It can connect to the server itself or wait for the server to connect to the node.

14.5.3 Secure the TCP link

From version two a very flexible security schema has been introduced. Each user is defined with a list of parameters

username A name that this user is referred to and known by. Always required for all users.

password A password this user must give to sign in. Always required for all users. Stored in configuration files as MD5 of password.

host-list A list with locations/computers/networks this user can access the daemon. Wild-card can be used so access from a single computer can be set as "192.168.1.8" but access for the hole subclass can be set as "192.168.1.*".

access-rights This is the access level this user have. This is a 32-bit value where the lower four bits define a value 0-15 that defines a privilege value which gives access to different levels of commands. The upper part of the 32-bit value is a bit field with specific rights.

14.5.4 Username/password pairs for TCP/IP drivers

This is a driver that is started through the normal driver interface, but after the start it does all its communication through the TCP/IP interface. This is therefore actually a Level II driver as it does all its communication on the higher level. It can also use the variables the daemon gives access to through the TCP/IP interface for its own configuration and state.

A driver of this type needs a username/password pair and it should always be defined for every setup.

The Host-list for the user drivers to be used, should always be the Localhost to increase security.

14.6 VSCP TCP/IP Protocol Description

14.6.1 Port

Default Port: **9598**

14.6.2 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" command looks like this

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

An additional mode will be added to the control interface of the VSCP daemon. This interface will make it possible to start stop drivers, get information about drivers, add/delete daemon decision matrix elements etc. This thus makes it possible to remotely control many aspects of the daemon and its control functionality.

The interface can also be used from applications such as a control panel widget under windows or a home automation server such as the Open Home Automation Server.

14.6.3 Link Commands

The daemon interface can be visible also on lower end (typically TC/IP) nodes. A subset of the commands are required to be available on such a node. Commands marked with *yes* in the link column below are the ones that a link node must have. All others are not mandatory.

14.6.4 Available commands

Command	Privilege	Link	From version	Description
+	0	yes	0.0.2	Repeat the last command
NOOP	0	yes	0.0.1	No operation.
QUIT	0	yes	0.0.1	Close the connection.
USER	0	yes	0.0.1	Username for login.
PASS	0	yes	0.0.1	Password for login.
SEND	4	yes	0.0.1	Send an event.
RETR	2	yes	0.0.1	Retrieve one or several event(s).
RCVLOOP	2	yes	0.0.2	Will retrieve events in an endless loop until the connection is closed by the client.
CDTA	1	yes	0.0.1	Check if there are events to retrieve.
CLRA	1	yes	0.0.1	Clear all events in in-queue.
STAT	1	yes	0.0.1	Get statistics information.
INFO	1	yes	0.0.1	Get status information.
CHID	1	yes	0.0.1	Get channel ID.
SGID	6	yes	0.0.1	Set GUID for channel.
GGID	1	yes	0.0.1	Get GUID for channel.
VERS	0	yes	0.0.1	Get CANAL/VSCP daemon version.
SFLT	4	yes	0.0.1	Set incoming event filter.
SMSK	4	yes	0.0.1	Set incoming event mask.
HELP	1	no	0.0.2	Give command help.
TEST	15	no	0.0.2	Do test sequence. Only used for debugging.
SHUTDOWN	15	no	0.0.2	Shutdown the daemon.
RESTART	15	no	0.0.2	Restart the daemon.
DRIVER	15	no	0.0.2	Driver manipulation.
FILE	15	no	0.0.2	File handling
UDP	15	no	0.0.2	UDP
REMOTE	15	no	0.0.2	User manipulation
INTERFACE	15	no	0.0.2	Interface manipulation.
DM	15	no	0.0.2	Decision Matrix manipulation.
VARIABLE	15	no	0.2.9	Variable handling.

14.6.5 NOOP - No operation.

This operation does nothing. It just replies with "+OK".

14.6.6 QUIT - Close the connection.

Close the connection to the host.

14.6.7 USER - Username for login.

Used to enter username for a password protected server.

Used on the following form:

USER username<CR><LF>

14.6.8 PASS - Password for login.

Used to enter username for a password protected server.

Used on the following form:

PASS password<cr><lf>

14.6.9 RESTART

Restart the daemon. Must have highest privilege to be able to do this.

14.6.10 SHUTDOWN

shutdown the daemon. Must have highest privilege to be able to do this.

14.6.11 DRIVER

With this command drivers can be handled. The argument defines which operation is performed.

driver install

Install a new driver. Full format is DRIVER install “path to driver package”. It’s possible to just install a driver temporarily or make it persistent to the system so it is loaded when the daemon starts. The driver package is a zip’ed file with a manifest file in XML format that tells where different components should go. The package system is also aware of the OHAS web framework.

driver uninstall

Uninstall a driver that is currently installed.

Full format is DRIVER uninstall “drivername”/id.

driver upgrade

Upgrade a driver that is currently installed.

Full format is DRIVER upgrade “drivername”/id.

driver start

Start an installed driver.

Full format is DRIVER start “drivername” /id.

driver stop

Stop an installed driver.

Full format is DRIVER stop “drivername” /id.

driver reload

Reload an installed driver.

Full format is DRIVER re lode “drivername” /id.

14.6.12 FILE

With this command files can be handled. It is implemented to enable an administrator to handle driver and configuration files from a remote location and for client applications so they can dump collected data.

dir

Show a directory listing for a folder given by the argument.

copy

Copy a file from one location to another.

move

Move a file from one location to another.

delete

Delete a file.

list

List content of file.

14.6.13 UDP

enable

Enable the UDP interface.

Full format UDP ENABLE

disable

Disable the UDP interface.

Full format UDP DISABLE

14.6.14 REMOTE

Remote user manipulation.

list

List user(s). Full format is REMOTE list wild-card The list user command has the following format

```
user1<CR><LF>
user2<CR><LF>
...
usern<CR><LF>
+OK<CR><LF>
```

add

Add a user. Full format is REMOTE add

```
"username","MD5 password","from-host(s)","access-right-list","event-
list","filter","mask"
```

The add user parts of the arguments can be left out after password. All arguments below the one that is left out must be present. No argument in the middle can be taken away.

remove

Remove a user. Full format is REMOTE remove “username”

privilege

Set new privilege for a user. Full format is REMOTE privilege “username”, “access-right-list”

password

Set new privilege for a user. Full format is REMOTE password “username”, “MD5 for password”

host-list

Set locations user can connect from. Full format is REMOTE password “username”, “host-list”

event-list

Set list of events user can send. Full format is REMOTE password “username”, “event-list”

filter

Set user filter. Full format is REMOTE password “username”, “filter”

mask

Set user mask. Full format is REMOTE password “username”, “mask”

14.6.15 INTERFACE

list

List interfaces.

For the list interfaces command the daemon respond with *interface_id1, type, interface_GUID1, interface_realmname1<CR><LF>*
interface_id2, type, interface_GUID2, interface_realmname2<CR><LF>
...
interface_idn, type, interface_GUIDn, interface_realmname<CR><LF>
+OK<CR><LF>

type is

Type	Description
0	Unknown (should not see this).
1	Internal daemon client
2	Level I (CANAL) Driver
3	Level II Driver
4	TCP/IP Client

close

Close interfaces.

Full format is **interface close interface _GUID**

Unique access to an interface can only be queried once for one interface. So two unique operations after each other de selects the first chosen interface before acquire the second.

14.6.16 DM

enable

Enable a decision matrix row. Argument is a comma separated list with DM row number(s) or “ALL” for all rows.

disable

Disable a decision matrix row. Argument is a comma separated list with DM row number(s) or “ALL” for all rows.

list

Show a decision matrix row number. Argument is DM row number(s) or “ALL” for all rows.

The list command gives a list of the following format

enabled,from,to,weekday,time,mask,filter,index,zone,sub-zone,control-code,action-code,action-param,comment,trig-counter,error-counter<CR><LF>

....

+OK<CR><LF>

- **Enabled:** First parameter is enabled if the row is active else disabled (“enabled”/“true”/1 or “disabled”/“false”/0).

- **From:** Allowed date plus time from which the action can be triggered. Form is *YYYY-MM-DD HH:MM:SS*.

- **To:** Allowed date plus time up to which the action can be triggered. Form is *YYYY-MM-DD HH:MM:SS*.
- **Weekdays:** Allowed weekdays for the action to be triggered.
- **Time:** The time(s) when the action should be triggered. This is a string on the form *YYYY-MM-DD HH:MM:SS*. where both parts can be replaced with a '*' to indicate that it is a no care. This means that * * is for all dates and all time while * HH:MM:SS is for all dates but a specific times. Further, all elements such as YYYY, MM, DD, HH, MM, SS can be replaced with a * to represent a no care for each place where it is present. Each can also be given as a list separated with '/' characters to indicate several choices. So *YYYY-MM-DD HH:0/5/10;SS* means the action should be performed on a specific date and hour on every full hour (0), five minutes past (5) and ten minutes past (10).
- **Mask:** A standard VSCP mask on the form *Priority;Class;Type;GUID* where all describe the masks part.
- **Filter:** A standard VSCP filter on the form *Priority;Class;Type;GUID* where all describe the filter part.
- **Index:** Is also part of the filter but is an absolute value from data-byte 0.
- **Zone:** Is also part of the filter but is an absolute value from data-byte 1.
- **Sub-zone:** Is also part of the filter but is an absolute value from data-byte 2.
- **Control-code:** The 32-bit control code. Note that enable/disable is part of this code.
- **Action-code:** The action code follows next.
- **Action-param:** A string consisting of action parameters.
- **Comment:** A comment for the row.
- **Trig-counter:** This is a counter for how many time the action for the row has been executed.
- **Error-counter:** This is a counter for action execution errors for the row.

See the DM description in the VSCP specification for information about the content of the control code etc.

add

Add a decision matrix row. The add-command needs a parameter of the following format

enabled,from,to,weekday,time,mask,filter,index,zone,sub-zone,control-code,action-code,action-param,comment

See the list command 14.6.16 for a detailed description of the items.

delete

Delete a decision matrix row.

Argument is a comma separated list with DM row number(s).

reset

Resets all variables and read in persistent values.

clrtrig

Clear trig counter for a decision matrix row.

Argument is a comma separated list with DM row number(s) or “ALL” for all rows.

clrerr

Clear error counter for a decision matrix row.

Argument is a comma separated list with DM row number(s) or “ALL” for all rows.

save

Save current decision matrix to a file. If no filename is given *dm.xml* is used with the path set to the configuration folder. See 14.7.4 for decision matrix file format.

load

Load decision matrix from a file. If no filename is given *dm.xml* is used with the path set to the configuration folder. A path + filename can be used and this file will be used instead. The last argument is *replace* or *add* (default). *replace* - replaces the old decision matrix content with the new and the option *add* adds the decision matrix from the file at the end of the current decision matrix. See 14.7.4 for decision matrix file format.

14.6.17 VARIABLE

list

List all defined internal variables and their values or use a wild-card to list specific variables. *all*, ***, **aaaa**, *aaaa** or **aaaa*

- *variable list* - lists all variables.
- *variable list all* - lists all variables.
- *variable list ** - lists all variables.
- *variable list abc** - lists all variables that have names starting with *abc*

- *variable list *abc* - lists all variables that end with *abc*
- *variable list *abc** - lists all variables that have names containing *abc*.

write

Set an internal variable value. If the variable is not already defined it is created.
Argument is

“*name of variable*”, *type*, “*persistence*”, “*value*”

type is the numerical variable type and is optional (default is string). *persistence* can be “true” or “false” for variable persistence and is optional (default is false). It is possible to proceed the variable with a '\$' to indicate persistent and '@' for an array. To just write a value use the format

“*name of variable*”, “*value*”

The type of a variable can not be changed if it already exists. In this case *remove* it first.

read

Read a variables value. Arguments is “*name of variable*” The response is

value\r\n
+OK

or if the variable is not defined

-OK – Variable is not defined.\r\n

The format for different variable types is as follows

Variable Type	Format for returned value
String	<i>String value</i>
Boolean	<i>true / false</i>
Integer	<i>Integer value</i>
Long	<i>Long value</i>
Double	<i>Double value</i>
VSCP measurement	String representing the measurement.
VSCP event	<i>head, class, type, obid, time-stamp, GUID, data1, data2, data3....</i>
VSCP GUID	Standard GUID string format. <i>FF:FF:FF:FF:FF:FF:FF:FF:FF:FF:FF:FF:FF:FF:FF:FF</i>
VSCP event data	Comma separated list with decimal values. <i>1,2,3,4,5,6,7,8....</i>
VSCP event class	<i>Integer value</i>
VSCP event type	<i>Integer value</i>

reset

If the variable value is persistent the persistent value is reloaded. If not the variable is set to its default value.

readreset

This is a combination of read + reset doing them together in an atomic way.

remove

Removes a variable. Argument is name of variable.

readremove

This is a combination of read + remove doing them together in an atomic way.

length

Get the length for a string variable. No effect for other variable types (returns 0).

save

Save persistent variables.

Output format from daemon for variable list/read commands is
"variable name",type,"persistence","value"<CR><LF>

...

+OK<CR><LF>

Persistence is presented as "true" or "false".

Defined Types

Code	Description
0	Unassigned (variable has not been used).
1	String format
2	Boolean e.g. true,false or 0,1
3	Integer format
4	Long integer format
5	Floating point value (double)
6	VSCP measurement data see Data Coding.
7	VSCP Event e.g. priority=5 class=10,type=40,data=0x12,1,2,3,4,5
8	GUID e.g. 00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00
9	Data e.g. 1,2,3,4,5...
10	Class e.g. 10
11	Type e.g. 22
12	Time-stamp 2008-11-07 20:10:00

- All numerical variables can be given as decimal values or as hexadecimal values (preceded with 0x).
- All variable beginning with “VSCP _” is reserved.
- New variables with names starting with \$ will be persistent.
- New variables with names starting with @ will be arrays. (Not available yet)
- Variable are read in by the daemon at start up from the variable file

```

<?xml version="1.0" encoding="utf-8"?>

<persistent>

    <!-- persistant storage is here -->

        <!-- string -->
        <variable type="string" >
            <name>testvariable_string</name>
            <value>This is a string</value>
            <note>This is a note about this variable that can contain info</note>
        </variable>

        <!-- boolean -->
        <variable type="bool" > <!-- type="boolean" also works -->
            <name>testvariable_bool</name>
            <value>true</value> <!-- numericals can also be used == 0 is false -->
        <note>This is a note about this variable that can contain info</note>
        </variable>

        <!-- integer -->
        <variable type="int" > <!-- type="integer" also works -->
            <name>testvariable_int</name>
            <value>1956</value>
            <note>This is a note about this variable that can contain info</note>
        </variable>

        <!-- long -->
        <variable type="long" >
            <name>testvariable_long</name>
            <value>0xFFFFFFFF</value>
            <note>This is a note about this variable that can contain info</note>
        </variable>

    <!-- Floating point value -->

```



```

        <value>1</value>
        <note>This is a note about this variable that can contain info</note>
</variable>

<!-- VSCP timestamp -->
<variable type="timestamp" >
    <name>testvariable_timestamp</name>
    <value>100</value>
    <note>This is a note about this variable that can contain info</note>
</variable>

<!-- Date + Time in ISO format -->
<variable type="datetime" >
    <name>testvariable_datetime</name>
    <value>2009-02-14 18:05:00</value>
    <note>This is a note about this variable that can contain info</note>
</variable>

<!-- numericals also work -->

<variable type="1">
    <name>test1</name>
    <value>This is some sample text</value>
    <note>This is a note about this variable that can contain info</note>
</variable>

<variable type="1">
    <name>test2</name>
    <value>This is also some sample text</value>
    <note>This is a note about this variable that can contain info</note>
</variable>

<variable type="2">
    <name>boolenavar</name>
    <value>true</value>
    <note>This is a note about this variable that can contain info</note>
</variable>

<variable type="3">
    <name>intvar1</name>
    <value>65535</value>
    <note>This is a note about this variable that can contain info</note>
</variable>

<variable type="3">
    <name>intvar2</name>

```

```
        <value>-23</value>
        <note>This is a note about this variable that can contain info</note>
    </variable>
</persistent>
```

This is how the variables look like using the *variable list all* command in the TCP/IP interface.

14.6.18 SEND - Send an event.

Used on the following form:

send head, class, type, obid, time-stamp, G UID, 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 globally unique.

If time-stamp is set to zero a time-stamp will be provided by the daemon. This time-stamp 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.

CLASS=512 - CLASS=1023 is treated in a special way by the daemon. If you use the interface GUID as the first sixteen data bytes the event will only be set through that interface. This is the correct way to send a Level I event through the daemon. You can send Level I events without the GUID also but in that case the event will be sent out on all interfaces and as each interface can have nodes with this nickname all will be affected.

Example:

Send a full GUID event

```
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<CR><LF>
```

Send Event. The example is the same as above but the GUID of the interface will be used.

```
send 0,20,3,0,0,-,0,1,35<CR><LF>
```

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

It is possible to send Level I events to a specific interface. To do this use the Level II mirror Level I events (Class=512-1023 VSCP Level II Level I events - CLASS2.LEVELI). This is events with class equal to 512 - 1023 which mirrors the Level I events but have the destination GUID in data bytes 0-15. Thees data-bytes is set to the interface (14 upper bits) and the node-ID for the node one wants to communicate with is in GUID[0]. This event will be sent to the correct interface.

14.6.19 RETR - Retrieve one or several event(s).

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

```
head,class,type,obid,time-stamp,GUID,data0,data1,data2,.....
```

GUID with MSB first.

Used on the following form:

```
RETR 2<CR><LF>
```

```
0,20,3,0,0,FF:FF:FF:FF:FF:FF:FF:FE:0:5:93:140:2:32:0:1  
0,20,4,0,0,FF:FF:FF:FF:FF:FF:FF:FE:0:5:93:140:2:32:0:1  
+OK -
```

If no events are available in the queue

```
-OK - No event(s) available.
```

is received by the client.

VSCP Event originating from a CANAL driver have the nickname of the node in the LSB of the GUID (GUID[0]). The rest of the GUID is the GUID for the interface.

If no argument is given one event is fetched even if there are more in the queue.

If you try to receive more events then there is events in the buffer -OK will be returned after the available number of events have been retrieved and been listed.

14.6.20 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 events.

To help in determining that the line is alive

+OK

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

Some applications may not implement this feature and should output

[+OK - Command not implemented<CR><LF>]

to indicate this.

14.6.21 CDTA - Check if there are events to retrieve.

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

Used on the following form:

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

14.6.22 CLRA - Clear all events in in-queue

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

Used on the following form:

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

14.6.23 STAT - Get statistics information.

Get interface statistics information. The returned format is

cntBusOff,cntBusWarnings,cntOverruns,cntReceiveData,cntReceiveFrames,cntTransmitData,cntTransmitF

Example:

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

14.6.24 INFO - Get status information.

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

channel_status,lasterrorcode,lasterrorsbocode,lasterrorstr

Example:

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

14.6.25 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.

Example:

```
CHID<CR><LF>
1234<CR><LF>
+OK - <CR><LF>
```

14.6.26 SGID - Set GUID for channel.

Set the GUID for this channel. The GUID is given 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>
```

14.6.27 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>
```

14.6.28 VERS - Get VSCP daemon version.

Get the current version for the daemon. The returned format is
major-version,minor-version,sub-minor-version

14.6.29 SFLT - Set incoming event filter.

Set the incoming filter. The format is

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

example

1,0x0000,0x0006,ff:ff:ff:ff:ff:ff:01:00:00:00:00:00:00:00

Note: The GUID values always is given as hexadecimal values without a preceding “0x”.

Note: If you want to filter on nickname-ID you should filter on GUID LSB byte.

14.6.30 SMSK - Set incoming event mask.

Set the incoming mask. The format is

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

example

1,0x0000,0x0006,ff:ff:ff:ff:ff:ff:01:00:00:00:00:00:00:00

Note: that the GUID values always is given as hexadecimal values without a preceding “0x”.

Note: If you want to mask on nickname-ID you should mask on GUID LSB byte.

14.7 Decision Matrix

This is a decision matrix definition for high level nodes like PC based or higher end embedded devices not the standard type from the specification. No thoughts has been wasted on trying to minimize the use of resources like memory here as it has for the embedded decision matrix where size is of great importance.

Events described here are internal to the high-end node and is therefore not visible for the nodes on the bus. On the other hand all events on the bus are also feed to the DM and events can also be sent from an element in the DM.

14.7.1 Level I

Action parameter is one byte with user specified content

For a high end node level I events are handled by the Level II matrix below by the Level I events being mirrored to the Level II equivalents.

14.7.2 Level II

Action parameter can have any length. There is also no limit in the number of DM rows available even if there naturally is a practical system limit.

14.7.3 Row format

Each decision matrix row contains a number of elements. The row itself can be *active* or *inactive* and have a *groupid* that can be used to hold rows that perform a combined operation together. For the *groupid* any alphanumeric combination can be used.

mask The mask is a standard VSCP Level II mask. The bits that are ones in the mask specifies the bits that should be checked in the filter. There are masks for all parts of a VSCP Level II event such as *priority*, *class*, *type*, *GUID*. Set one of these to zero if they are of now interest and to ones if they are or use bit combinations to create ranges of interest.

filter The mask is a standard VSCP Level II filter. The filter is combined with the mask. First the incoming event is masked this means all bits where the mask is zero is also set to zero in the event. The result is compared to the filter and if they are the same we have a match and the decision matrix row is triggered.

If you set the mask for a row to all ones and the filter equal to the event you are interested in that row will be triggered when that event is received by the daemon. In this case you can have the mask for priority, GUID set to zero so that events with all priorities and from all units (with different GUID's) will trigger the row. If you only are interested in events from a specific node. Set the GUID part of the mask to ones (FF:FF:FF:FF:FF:FF:FF:FF:FF:FF:FF:FF) and the GUID part for the filter to the nodes GUID.

Another way one can use to match a single event is to set both filter and mask to the same value. This will have the same effect as the above method. Just remember that the mask should have a zero in a bit position for a don't care and a one for a significant bit and the filter should have the corresponding bit set to the value that is wanted and all other bits set to zero.

The truth table for all this looks like this:

filter ^ event-bit	mask	result
0	0	1
0	1	1
1	0	1
1	1	0

control This is the 32-bit DM control word.

Bit	Description
31	This bit is the same as the rows active/inactive status for a decision matrix row.
30	This bit can be used to disable all decision matrix entries below the row that has this bit set.
5	Specifies that the index should be checked.
4	Specifies that the zone should be checked.
3	Specifies that the sub-zone should be checked.

allowed_from It is possible to define a date/time *from* which the action is allowed to be executed. Specify a time on ISO format as *1970-01-01 00:00:00*. If not specified there is no restriction.

allowed_to It is possible to define a date/time *upto* which the action is allowed to be executed. Specify a time on ISO format as *2099-12-31 23:59:59*. If not specified there is no restriction.

allowed_weekdays This tells which weekdays the action is allowed to happen on. The format is mtwtfss each representing a weekday starting with Monday. Replace the weekday with a '-' to mark that day as not allowed to execute the action on.

allowed_time Determines the timespan when the action is allowed to occur. This method parses a string on the form YYYY:MM:DD HH:MM:SS. Both parts can be replaced with a '*' to indicate that it is a no care meaning that '*' is for all dates and all time while '*' HH:MM:SS is for all dates but a specific time. Further all elements such as YYYY, MM, DD, HH, MM, SS can be replaced with a '*' to represent a no care for each where it is present. Each can also be given as a list separated by '/' characters to indicate several choices. So YYYY:MM:DD HH:0/5/10;SS means the action should be performed on a specific date and hour on every full hour, five minutes past and ten minutes past.

action This is a 32-bit action code that specifies which action should be executed if the row is triggered. Actions and there codes are described here .

param The parameter is a text string that makes it possible to control how the action is performed. Before the action is performed the string is checked for escapes and this makes it possible to pass runtime information in an easy way. Escapes are defined here 14.7.6

comment Make your own comments of what the row is there for here.

14.7.4 The decision matrix file format

When the daemon is started up the internal decision matrix is loaded from the dm.xml in the folder set as configuration path. The format for the file is as follows

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

<!-- Version 0.0.2          2013-08-06
-->
<!-- This files holds the decion matrix for the daemon
-->

<dm>
    <!-- Enabled row, belongs to group "test" -->
    <row enabled="true" groupid="test">

        <!-- Mask to trigger row -->
        <mask
            priority="1"
            class="0xFFFF"
            type="55"
            guid="0F:0E:0D:0C:0B:0A:09:08:07:06:05:04:03:02:01:00">
    </mask>

        <!-- Filter to trigger row -->
        <filter
            priority="7"
            class="1000"
            type="0xAA"
            guid="FF:EE:DD:CC:BB:AA:99:88:77:66:55:44:33:22:11:00">
    </filter>

        <!-- Date and time from which action should trigger -->
        <allowed_from>1970-01-01 00:00:00</allowed_from>
```

```

<!-- Date and time up to which action should trigger -->
<allowed_to>2099-12-31 23:59:59</allowed_to>

<!-- List of weekdays which action should trigger -->
<allowed_weekdays>mtwtfss</allowed_weekdays>

<!-- A specific time (or pattern) when the action should trigger -->
<allowed_time>*-*-* *:0 /5 /10 ;0</allowed_time>
    <!-- Every zero, five and ten minutes-->
    <!-- Optional - Set Index to check -->
    <!-- If bMeasurement is true just the lower three bits
        will be compared -->
    <index bMeasurement="true | false">0</index>

    <!-- Optional - Set Zone to check -->
    <zone>0</zone>

    <!-- Optional - Set Subzone to check -->
    <subzone>0</subzone>

    <!-- Control code -->
    <control>0x80000000</control>

    <!-- Action code -->
    <action>0x10</action>

    <!-- Action parameter -->
    <param>1 ,t44 ;canal ;12 ;25 ;abc ;90</param>

    <!-- Comment for decion matrix row -->
    <comment>This is a dumb comment</comment>

</row>

</dm>

```

14.7.5 Scheduler

Each event that is received by the daemon is feed through the decision matrix. When the event is received it is placed on the DM input queue after passing an initial set of filter that removes events that are of no interest to the specific implementation.

The first event in the DM queue is then run through the matrix and each DM row is compared and if there is a match the action for that row is executed.

The daemon itself place some events on the DM queue. See Internal DM events below. For example the internal LOOP event is run through the matrix

between every external event feed to the queue. The LOOP event will also be seen in the matrix when no other events are present. In this case a configuration value is used to set the time between two LOOP events (sleep time) if no other events arrive.

14.7.6 Actions

Actions can have long argument lists. If you are an end user don't be scared by this. The setup of DM rows is done with program support and you fill in the values in a nice GUI.

The semicolon character ';' is used as a separator for arguments and if needed in an argument "%;" should be used. Also % is a special character used for escapes and should be written as %%.

Variable substitution for parameters Action parameters are strings that are passed to actions and in that way can be used to configure the action to solve different problems. The action string can contain escape values which are replaced with real values before the action is performed. The following escapes are currently defined.

Substitution	Description
%%	The character '%'
%;	The character ';'. Semicolon is normally used to separates arguments of an action.
%cr	A carriage return.
%lf	A line feed.
%crlf	A carriage return + a line feed.
%tab	A tab.
%bell	A bell.
%event	A full event in the standard text form.
%event.class	The class for the event as a number.
%event.class.str	The class for the event as a descriptive string.
%event.type	The type for the event as a number.
%event.type.str	The type for the event as a descriptive string.
%event.head	head of the event.
%event.priority	The priority for the event (from head).
%event.sizedata	The number of databytes the event have.
%event.data	All data-bytes as a comma separated list. If no data 'empty' is set.
%event.data[n]	Data byte n. If no data 'empty' is set.
%event.obid	obid of event.
%event.hardcoded	Hardcoded bit f head as 0/1.
%event.guid	GUID of the event.
%event.timestamp	Timestamp of the event.
%isodate	Date in ISO format YY-MM-DD
%isotime	Time in ISO form HH:MM:SS.
%mstime	Current time in milliseconds.
%hour	Current hour.
%minute	Current minute.
%second	Current second.
%week0	Current week number (1-52(53)). Week starts with Sunday.
%week1	Current week number (1-52(53)). Week starts with Monday.
%weektxt	Get week name in textual short form.
%weektxfull	Get week name in textual long form.
%month	Current month number (1-12).
%monthtxt	Get current month in textual short form.
%monthtxfull	Get current month in textual long form.
%year	Current year.
%quarter	Current quarter(1-4).
%variable:variable_name	value of variable.
%path_config	Return the directory containing the system config files. Unix: /etc Windows: C:\Documents and Settings\All Users\Application Data Mac: /Library/Preferences
%path_datadir	Return the location of the applications global, i.e. not user-specific, data files. Unix: prefix/share/appname Windows: the directory where the executable file is located Mac: appname.app/Contents/SharedSupport bundle subdirectory
%path_documentsdir	Return the directory containing the current user's (the account the daemon/server is run as) documents. Unix: ~ (home directory) Windows: %userprofile%\Documents

Can perfectly be used as arguments for triggered external program execution.

NOOP. No operation.

Execute external program. Run an external program. The daemon must have execute access rights to be able to launch the program. It is important that this program returns as fast as possible. If lengthy operations is needed please fork another process in your external program and then return.

Parameters

- Full path to program + arguments to program. (possibly with with substitution elements %event.class %event.type %date etc).
- Optional variable that is set to true if the execution was successful.

Note that the semicolon symbol (';') is used internally and if it is part of a program argument %; should be written.

Example Start a new instance of notepad every minute at a specific date and time interval.

```
1 <row enabled="true">
2
3     <!-- Mask to trigger row - zero for a bit is don't care -->
4     <mask priority="0"
5         class="0xFFFF"
6         type="0xFF"
7         guid="00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00">
8     </mask>
9
10    <!-- Filter to trigger row --
11        if mask have a one in a bit it is compared with filter --
12        <filter priority="7"
13            class="0xFFFF"
14            type="5"
15            guid="00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00">
16    </filter>
17
18    <!-- Date and time from which action should trigger --
19    <allowed_from>2020-01-01_00:00:00</allowed_from>
20
21    <!-- Date and time up to which action should trigger --
22    <allowed_to>2020-01-01_05:59:59</allowed_to>
23
24    <!-- Date and time up to which action should trigger -->
```

```

25 <allowed_weekdays>mtwtfss</allowed_weekdays>
26
27 <!-- A specific time (or pattern) when the action should trigger -->
28 <!-- Every ten seconds -->
29 <allowed_time>*-*-*:*:0</allowed_time>
30
31 <!-- Control code -- Note that the row property enabled is doubled
32 as the highest bit here -->
33 <control>0x80000000</control><!-- enable_row -->
34
35 <!-- Action code -->
36 <action>0x00000010</action>
37
38 <!-- Action parameter -->
39 <param>c:\windows\notepad.exe</param>
40
41 <!-- Comment for decision matrix row -->
42 <comment>Start a new instance of notepad every minute at a
43 specific date and time interval.
44 </comment>
45
46 </row>

```

http GET, PUT, POST This action access a given URL given by the action parameter. Can be used to write data to a remote data source using web server scripts or similar.

Parameters

- url
- Access method: 'get', 'put' or 'post'
- URL-encoded form data post data if 'post' is used above. This data should be on the form

$$name=Lucy\&neighbors=Fred+26+Ethel$$
 Ignored if access method is 'get'
- Optional proxy on the form <hostname>:<port number> (just leave blank if no proxy should be used).

Send event to remote VSCP daemon This action can be used to send an event to a remote VSCP daemon.

Parameters

- address to server.
- port
- username
- password
- event to send

Store in variable Store data in a variable. The variable is named in the argument. If the variable is not available it is created.

Parameters

- Variable name with assignment on the form "variable name",type,"persistence","value"

Example parameter data

*example with integer: test_int,3,true,24000
example with string: test_string,1,true,"This is a string"
example with event: test_event,7,false,0,20,1,2,3,4,5,6*

Add to a variable Add data to a variable. The variable is named in the argument.

Parameters

- \$variable; numerical value to add.

If the variable does not exist it is created.

Subtract from a variable Subtract the data from a variable. The variable is named in the argument.

Parameters

- \$variable; value to subtract.

Multiply with a variable Multiply the data with a variable. The variable is named in the argument.

Parameters

- \$variable; Value to multiply variable with

Divide the data with a variable Divide the data with a variable. The variable is named in the argument.

Parameters

- \$value; value to divide variable with

Check variable, set to true Check if variable is greater then a specified value and set some other variable to *true* if it is. The variable is named in the argument. If the variable is not available it is created. This action is typically used as a timer together with the send event conditional action to stop resends on timeout.

Parameters

- value to check.
- variable to check.
- variable to set to *true*.

Example parameter data

500;counter;flag

Will check if counter is greater then 500 and set flag to *true* when it is.

Check variable, set to false Check if variable is greater then a specified value and set some other variable to *false* if it is. The variable is named in the argument. If the variable is not available it is created. This action is typically used as a timer together with the send event conditional action to stop resends on timeout.

Parameters

- value to check.
- variable to check.
- variable to set to *false*.

Example parameter data

500;counter;flag

Will check if counter is greater then 500 and set flag to *false* when it is.

Send event Send event when another event is received.

Parameters

- Event to send. Should be on the form *head, class, type, obid, time-stamp, GUID, data1, data2, data3....*
- Optional variable that will be set to true. If the variable is used it should be separated from the event with a | so a full parameter list with a variable should take the following form
0,20,3,0,0,0:1:2:3:4:5:6:7:8:9:10:11:12:13:14:15,0,1,35/bSent

Example This example send an event every ten seconds. It happens all weekdays during a specified time period from 1977-01-01 00:00:00 to 2099-12-31 23:59:59. The optional variable *bSent* is set to false when the event has been sent.

```

1 <row enabled="true">
2
3     <!-- Mask to trigger row - zero for a bit is don't care -->
4     <mask priority="0"
5         class="0xFFFF"
6         type="0xFF"
7         guid="00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00">
8     </mask>
9
10    <!-- Filter to trigger row --
11        if mask have a one in a bit it is compared with filter --
12        <filter priority="7"
13            class="0xFFFF"
14            type="5"
15            guid="00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00">
16    </filter>
17
18    <!-- Date and time from which action should trigger --
19    <allowed_from>1977-01-01_00:00:00</allowed_from>
20
21    <!-- Date and time up to which action should trigger --
22    <allowed_to>2099-12-31_23:59:59</allowed_to>
23
24    <!-- Date and time up to which action should trigger --
25    <allowed_weekdays>mtwtfss</allowed_weekdays>
26
27    <!-- A specific time (or pattern) when the action should trigger --
28    <!-- Every ten seconds --
29    <allowed_time>*-*-*:*:0 /10/20/30/40/50</allowed_time>
30
31    <!-- Control code -- Note that the row property enabled is doubled
```

```

32   .....as the highest bit here -->
33   <control>0x80000000</control> <!-- enable row -->
34
35   <!-- Action code -->
36   <action>0x00000010</action>
37
38   <!-- Action parameter -->
39   <param>0,20,3,0,0,0,0:1:2:3:4:5:6:7:8:9:10:11:12:13:14:15,0,1,35|bSent</pa
40
41   <!-- Comment for decision matrix row -->
42   <comment>Send an event every ten seconds</comment>
43
44 </row>

```

Send event conditional Send an event if a specified variable is true.

This action is specified to make it possible to define DM entries that wait for a response from a node and resend a specified event a number of times until this response is received or a timeout occurred.

Parameters

- Control variable name. Event is sent if variable is true.
- Event to resend.

The variable is set to true by the action that initially sent the event that needs a reply. Typically this can be an event that triggered an ON event and expects a confirm. When the variable is true this action gets triggered and if triggered by one of the internal time events it will start to send periodic events with a period equal to the internal time events period. The filter/mask should be set to trigger on some of the internal time events (loop, second, minute etc). This choice determines the interval between resend of events.

To get all this to work four DM rows are required.

1. A row that sends the original event and sets the flag to true.
2. A row where a timer is started that sets the variable to false when it elapses.
3. A row with this action. The event is resent until a response is received which sets the flag to false. *Check variable set to false* can be used for this as well.
4. A row that sets the flag to true when the correct response is received. The stop timer action is perfect for this. The trigger should be the event which is the expected response.

Send event(s) from file This action sends event(s) from a named file.

Parameters

- Path to file.

The format for the XML file is

Write to file plain This action writes (appends) the substituted action argument string to a file. If the string should be written one on each row be sure to add %crlf for windows and %lf for Unix at the end of the action-string.

Parameters

- Path to file
- 1 for append. 0 for overwrite.

Start a timer This action starts a timer identified by a numerical ID that will count down from a specified time set in seconds. If the timer is already defined it will be reused and if active it will be reinitialized.

Parameters

- Timer ID.
- Count down time in seconds.
- Variable that is set to *setvalue* when the timer elapses.
- *setvalue* (true or false).

Stop a timer This action stops an active timer identified by a numerical ID. If the timer does not exist or is inactive the action does nothing.

Parameters

- Timer ID.
- Variable that will be set to false.

14.7.7 Internal DM Events

DM events is a special type of events that just are available in the DM loop and is generated internally by the daemon. Actions can be triggered by these events.

The available events are:

- **Internal loop event** - Event is generated each loop in the DM or if no events is received every 100 ms (configurable value).
- **Internal starting up event** - The first event that is sent through the DM when the daemon is started.
- **Internal shutting down event** - The last event that is sent through the DM when the daemon is shutting down in a controlled way.
- **Internal pause event** - The machine/daemon is going to a pause state.
- **Internal activate event** - The machine/daemon is going from a pause state.
- **Internal second event** - This event is generated once a second.
- **Internal minute event** - This event is generated once a minute.
- **Internal hour event** - This event is generated once each hour.
- **Internal day event** - This event is generated at midnight.
- **Internal week event** - This event is generated when a new week is stated.
- **Internal month event** - This event is generated when a new month is stated.
- **Internal year event** - This event is generated when a new year is stated.
- **Internal quarter event** - This event is generated when a new quarter is stated.
- **Internal random minute event** - This event is generated randomly once each minute.
- **Internal random hour event** - This event is generated randomly once each hour.
- **Internal random day event** - This event is generated randomly once each day.
- **Internal random month event** - This event is generated randomly once each month.
- **Internal random year event** - This event is generated randomly once each year.

- **Internal dusk event** - If enabled dusk event will be generated.
- **Internal dawn event** - If enabled dawn event will be generated.
- **Internal timer started event** - Sent when a new timer is started. For argument definition look at specification level II events.
- **Internal timer paused event** - Send when a timer is paused. For argument definition look at specification level II events.
- **Internal timer resumed event** - Sent when a paused timer is resumed. For argument definition look at specification level II events.
- **Internal timer stopped event** - Sent when a timer is permanently stopped. For argument definition look at specification level II events.
- **Internal timer elapsed event** - Sent when a timer has elapsed. This is the wakeup call. For argument definition look at specification level II events.

14.7.8 Timing parameter data format

- String with days the action can occur in as *mtwtfss* use '-' for day when action should not be performed.
- **Start Data/time** for when the action can occur in ISO format as *YYYY-MM-DD HH:MM:SS* Use a wild-card character '*' if any of them is don't care.
- **End Data/time** for when the action can occur in ISO format as *YYYY-MM-DD HH:MM:SS* Use a wild-card character '*' if any of them is don't care.
- **Action date/time** when action should occur in ISO format as *YYYY:MM:DD HH:MM:SS*

The start and the end Date/time fields set the date/time range when the action is allowed to occur.

The action date set the date/time when the action should occur. This can be a one shot in which case the full date/time is filled in as in *2009-11-02 14:30:00* which will perform action at this time once if the start/end dates allow for that. To get repeats it is possible to use wild-cards. For example **-*-* 14:30:** (can also be written as ** 14:30:**) will perform the operation every seconds from 14:30 to (but not including) 14:31 in the time range set by the start/end date/time range.

For repeating operation that is just dependent on time filter on the SECOND event so that this is the only event that trigger the row.

It is also possible to specify more then one of each element in the date/time by separating them with a slash. For example ** 14:0/10/20/30/40/50:00* will do the action every ten minutes between two a clock and three a clock in the afternoon.

14.7.9 External standard functionality

This functionality is provided as dll/dl with the package. It is not built into the daemon itself because of portability of the daemon itself.

Log to OHAS database The Open Home Automation Server have a database with many useful tables. This function writes data to this table in a standardized way.

Log to syslog/eventlog Send message to syslog server or to event log.

Send email First argument is receiver, second argument is subject, third argument is content.

Send SMS First argument is receiver, second argument is subject, third argument is content.

Access URL Only argument is URL.

14.7.10 String substitution keywords

All string substitution keywords are preceded with a %

14.7.11 Examples

Execute a program with argument when the ON event is received

- Set filter to see only the TURN-ON event and possibly zone (second data byte) and sub-zone (third data byte) if only interested in a specific location.
- The action code is set to *Run external program*.
- First parameter is path to program.
- Second parameter is argument to program.

Use substitutions for the arguments if event data should be transferred to the external program.

Execute heyu with argument at a specific time with constraints The external program heyu is used to control X10 devices. Here we want to turn on a group of lamps at 18:00 each night except during the summer. But never do this on Saturdays.

- Set filter/Mask so only the SECOND internal daemon event is interesting.
- The action code is set to *Run external program at specific time*.

- First parameter is set to *mtswtf-s* to allow the action to take place all days except Saturdays.
- Second parameter is *10-01;04-01* to allow the action to occur from October first to April first our summer definition.
- Third parameter is **:*;*:** as we have no timing constraints.
- Fourth parameter is **-*-* 18:00:00* the time we want the lamp to turn on.
- Fifth parameter is *path to heyu*
- Sixth parameter is argument to heyu. In this case “*on A13*” to turn on switches set to A13.

14.8 Drivers

The daemon recognize two types of drivers. Level I drivers that confirm to the CANAL interface specification (27) and Level II drivers that use the higher end VSCP package format and full GUID.

14.8.1 Level I (CANAL) drivers

This is just a .dll (Windows) or a .so (Linux) file with the CANAL interface exported. The interface is described here 27. Several CANAL driver comes with the VSCP & Friends package and information about them is here VIII.

The good thing with the CANAL interface is that you can add the .dll or .so as a driver to canal daemon or use the dll directly from your own application.

To make a CANAL driver just create a dynamically linked library that export the CANAL interface. There are plenty of examples to use as a starting point for creating your own driver in the source tree at *src/vscp/drivers/level1*.

14.8.2 Level II drivers

VSCP drivers or Level II drivers (fully described here IX) have two advantages over Level I drivers.

- They always use the Level II event format which means that the full GUID is used.
- They interface the daemon through the full TCP/IP interface giving a lot of possibilities.

The ability to use the full GUID is good as there is no need for translation schema's between the actual GUID and GUID's used in interfaces. The node-ID is unique all over the world.

Letting the driver talk to the daemon over the TCP/IP interface is favorable in that it can do many things that previously was not possible. The most

exciting is that it can read and write variables (even create new ones if needed). This is the recommended way to use for configurations of a Level II driver. It means that configuration of all drivers can be made in one place (the daemon variable file), it gives a possibility to change runtime values in real time etc.

The level II driver is, just as the Level I driver, a dynamic link library with a specific set of exported methods. The exported methods are four of the methods from the CANAL interface and uses identical calling parameters and return values. There are some differences however fully described here IX.

The drivers are configured in the vscpd.conf file (format is here 14.4.5). If you use more than one driver with different configuration it is very important that the prefix is set to different values for each of them. The prefix is prepended to a variable name before it is fetched from the daemon or set for that matter.

14.9 HTML5 websocket interface

The daemon exports a HTML5 websocket interface from version 0.3.3. This interface makes it possible to have web widgets that are self contained and entirely written in JavaScript which can send and receive VSCP events. This means that you can create a simple web page, place your widgets on it and with or without a stand alone web server have a lightweight user interface. As phone, tablets and other devices generally also support HTML5 you have a general way for user interface creation. If you prefer apps, you can compile your interface code using phonegap or similar tools.

The websocket server can be reached on port 7681(default). The VSCP & Friends package comes with a few simple test pages.

One important design goal when this interface was designed was to create an interface that also could be implemented on low end devices which needs a user interface. With just a few commands and some simple rules we have managed to do that. This means that we can expect user interfaces interacting with both small devices and larger software units as the VSCP daemon.

14.9.1 Subprotocols

Four different sub protocols are currently implemented. Two are for test purposes, one is plain html and one is for VSCP usage.

http Just serve simple html pages. Variable substitutions will be added to this interface in the future.

dumb-increment-protocol This protocol just increments a value and the counter can be rested. Thats all.

Algorithm 1 *dumb-increment-protocol* sample code.

```
var socket_di;

if ( BrowserDetect.browser == "Firefox" &&
    BrowserDetect.version < 12) {
    socket_di =
        new MozWebSocket( get_appropriate_ws_url(),
                          "dumb-increment-protocol");
}
else {
    socket_di = new WebSocket( get_appropriate_ws_url(),
                               "dumb-increment-protocol");
}
try {
    socket_di.onopen = function() {
        document.getElementById("wsdi_statustd").style.backgroundColor =
            "#40ff40";
        document.getElementById("wsdi_status").textContent =
            "websocket connection opened";
    }
    socket_di.onmessage =
        function got_packet(msg) {
            document.getElementById("number").textContent =
                msg.data + "\n";
        }
    socket_di.onclose =
        function(){
            document.getElementById("wsdi_statustd").style.backgroundColor =
                "#ff4040";
            document.getElementById("wsdi_status").textContent =
                "websocket connection CLOSED";
        }
}
catch(exception) {
    alert('<p>Error' + exception);
}

function reset() {
    socket_di.send("reset\n");
}
```

lws-mirror-protocol Just mirrors what is sent to all other open sockets using the same protocol. In the sample all that is drawn by the mouse on one canvas is also drawn on all other canvas.

Algorithm 2 *lws-mirror-protocol* sample code.

```
var down = 0;
var no_last = 1;
var last_x = 0, last_y = 0;
var ctx;
var socket_lm;
var color = "#000000";

if (BrowserDetect.browser == "Firefox" && BrowserDetect.version < 12) {
    socket_lm =
        new MozWebSocket(get_appropriate_ws_url(),
        "lws-mirror-protocol");
}
else {
    socket_lm = new WebSocket(get_appropriate_ws_url(),
        "lws-mirror-protocol");
}
try {
    socket_lm.onopen = function()
    {
        document.getElementById("wslm_statustd").style.backgroundColor =
            "#40ff40";
        document.getElementById("wslm_status").textContent =
            "\u26bd\u200dconnection\u200dopened";
    }
    socket_lm.onmessage = function got_packet(msg) {
        j = msg.data.split(';');
        f = 0;
        while (f < j.length - 1) {
            i = j[f].split(',');
            if (i[0] == 'd') {
                ctx.strokeStyle = i[1];
                ctx.beginPath();
                ctx.moveTo(+i[2]), +(i[3]));
                ctx.lineTo(+i[4]), +(i[5]);
                ctx.stroke();
            }
            if (i[0] == 'c') {
                ctx.strokeStyle = i[1];
                ctx.beginPath();
                ctx.arc(+i[2], +i[3], +(i[4]), 0, Math.PI*2, true);
                ctx.stroke();
            }
            f++;
        }
    }
    socket_lm.onclose = function(){
        document.getElementById("wslm_statustd").style.backgroundColor =
            "#ff4040"; document.getElementById("wslm_status").textContent =
            "\u26bd\u200dconnection\u200dCLOSED";
    }
}
catch(exception) {
    alert('<p>Error' + exception);
}

var canvas = document.createElement('canvas');
canvas.height = 300;
canvas.width = 480;
ctx = canvas.getContext("2d");
document.getElementById('wslm_drawing').appendChild(canvas);
canvas.addEventListener('mousemove', ev_mousemove, false);
canvas.addEventListener('mousedown', ev_mousedown, false);
canvas.addEventListener('mouseup', ev_mouseup, false);
offsetX = offsetY = 0;
element = canvas;

if (element.offsetParent) {
    do {
        offsetX += element.offsetLeft;
        offsetY += element.offsetTop;
    } while ((element = element.offsetParent));
}
function update_color() {
    color = document.getElementById("color").value;
}
function ev_mousedown (ev) {
    down = 1;
}
function ev_mouseup(ev) {
    down = 0;
    no_last = 1;
}
function ev_mousemove (ev) {
    var x, y;

    if (ev.offsetX) {
        x = ev.offsetX;
        y = ev.offsetY;
    }
    else {
        x = ev.layerX - offsetX;
        y = ev.layerY - offsetY;
    }
    if (!down)
        return;
    if (no_last) {
        no_last = 0;
        last_x = x;
        last_y = y;
        return;
    }
    socket_lm.send("d" + color + "l" + last_x + "u" +
        last_y + "l" + x + "u" + y + ';');
    last_x = x;
    last_y = y;
}
```

very-simple-control-protocol For Very Simple Control Protocol usage and what's described here.

A simple sample for a button widget looks like this.

Algorithm 3 VSCP websocket button example

```
new button_kitchenLight( "kitchenLight",
                         "url",
                         port,
                         x,y,
                         "name_of_div",
                         width,height,
                         "event(s)_to_send_when_pressed",
                         "event(s)_wanted_for_confirmation",
                         timeout
);
```

And for a temperature widget

Algorithm 4 VSCP websocket temperature example

```
new temp_outside( "outsideTemp",
                  "url",
                  port,
                  x,y,
                  "name_of_div",
                  width,height,
                  vscp_class,
                  vscp_type,
                  zone,
                  subzone
);
```

As seen by the above two examples it is very easy to place and construct a dynamic widget. The only additional code to the above is defining and naming the div's in css. The rest is handled by the widget themselves. Tools will be available to do this for you without any knowledge of JavaScript or HTML what so ever. Just drag and drop.

Another thing that should be noted is that the widget have the url and port of the server as a parameter. This means that widgets on one page actually can connect to several different servers exporting a websocket api. With the websocket api being so low weight we can expect the api to be implemented also directly by low end Internet enabled devices.

This document will not describe the actual use of the widgets and some other document will be available later that describes usage in more detail.

14.9.2 Packets and their formats

Message traffic on the socket is

command / 'C' ; command ; optional data that may be separated by additional semicolons. /

Positive reply / '+' ; originator /

Negative reply / '-' ; originator ; Error code ; Error in real text /

Example: -;E;Transmit buffer full

Event / 'E' ; head , vscp_class , vscp_type , obid, timestamp, GUID, data /

Example: E;0,30,5,0,0,FF:FF:FF:FF:FF:FF:FF:FE:00:26:55:CA:00:06:00:00,0x01,0x01

14.9.3 Packet prefixes

Code	Description
C	Command. Parameter is command and optional parameter(s).
+	Positive Reply. If send in response to a command the command is also returned on the form '+;command'.
-	Negative reply. Payload is error code followed by error message in English.
E	Event. Payload is VSCP event.

Table 9: Message prefixes

14.9.4 Commands

The following commands are currently available

Command	Description
NOOP	No command. Positive response is returned. Example: “C;NOOP” response is “+;NOOP”
OPEN	Start receiving events. Collected events in queue will be sent. Positive/negative response is returned. Example: “C;OPEN” response is “+;OPEN”
CLOSE	Stop receiving events. Events are still collected in queue on server side. Positive/negative response is returned. Example “C;CLOSE” response is “+;CLOSE”
CLRQUE	Clear events in input queue. Positive/negative response is returned. Example: “C;CLRQUE” response is “+;CLRQUE”
SETFILTER	Set filter/mask for incoming data. Positive/negative response is returned. Example: “C;SETFILTER;filter-priority, filter-class, filter-type, filter-GUID;mask-priority, mask-class, mask-type, mask-GUID” response is “+;SETFILTER” Note that there is a semicolon between filter and mask information and commas between parts of each filter and mask.
READVAR	Read a variable. Argument: Name of variable. Positive/negative response is returned. Example: “C;READVAR;name” response is “+;READVAR;type;variable-value” where type is variable type and value is different depending on which type of variable it is. For a string the response is “+;READVAR;1;hello world” and for a boolean variable “+;READVAR;2;true”/“+;READVAR;2;false” etc. See 14.6.17 for more information on variable types.
WRITEVAR	Set value for existing variable. Argument: Name of variable;new value. Positive/negative response is returned. Example: “C;WRITEVAR;test;false” which set the binary value to false response is “+;WRITEVAR”
ADDVAR	Add a new variable. “C;ADDVAR;name;type;bPersistens;value” Positive response is returned. Example: “C;ADDVAR;test;1;0;hello world” which add the non persistent string value “hello world” response is “+;ADDVAR” ²⁹⁰
SAVEVAR	Save persistent variables. Positive/negative response is returned. Example: “C;SAVEVAR” response is “+;SAVEVAR”

Table 10: Websocket commands

Algorithm 5 Error messages codes

Error code	Error message
1	Syntax error.
2	Unknown command.
3	Transmit buffer full.
4	Variable is already defined.
5	Unable to find variable.

NOOP is typically used to test if a connection is working. OPEN/CLOSE can be used to stop the stream from incoming events. Note that events still are collected at the server side and will be sent after a closed socket has been opened again. Use CLRQUE to clear the queue at the server before opening the stream if you don't want to receive collected events.

Variables on the server is a powerful tool. Variables are used by Level II drivers and the internal decision matrix of the daemon. This means that driver parameters can be changes and complex setups can be accomplished together with the decision matrix functionality. As an example you can set a variable to true and have an action executed by the daemon that does something useful. The possibilities are endless. Variables can be persistent meaning they will live over time so it can also me a method to save states.

14.9.5 Send events

The same format is used to send events as they are received.

/'E';head,vscp_class,vscp_type,obid,timestamp,GUID,data/

and

/+;EVENT/

or

/-;EVENT;error-code;realtext-error/

is returned.

14.9.6 Error messages

15 VSCP Works

15.1 Introduction

VSCP Works is a graphical toolbox for VSCP developers and users that is available for Linux and Windows. It can be used to update code in in-house nodes and nodes that sit at a location on the other side of the earth. It can be used to view events sent by a node, to configure a node using a high end interface and to simulate a node. New functionality will be added to VSCP Works as needs arise.

There is a three video walk through on You tube that look at some of the essentials of VSCP Works

<http://www.youtube.com/watch?v=uIMSG8ewvIw>
<http://www.youtube.com/watch?v=XenCTnhWJKM>
<http://www.youtube.com/watch?v=fd6cNc7iIzE>

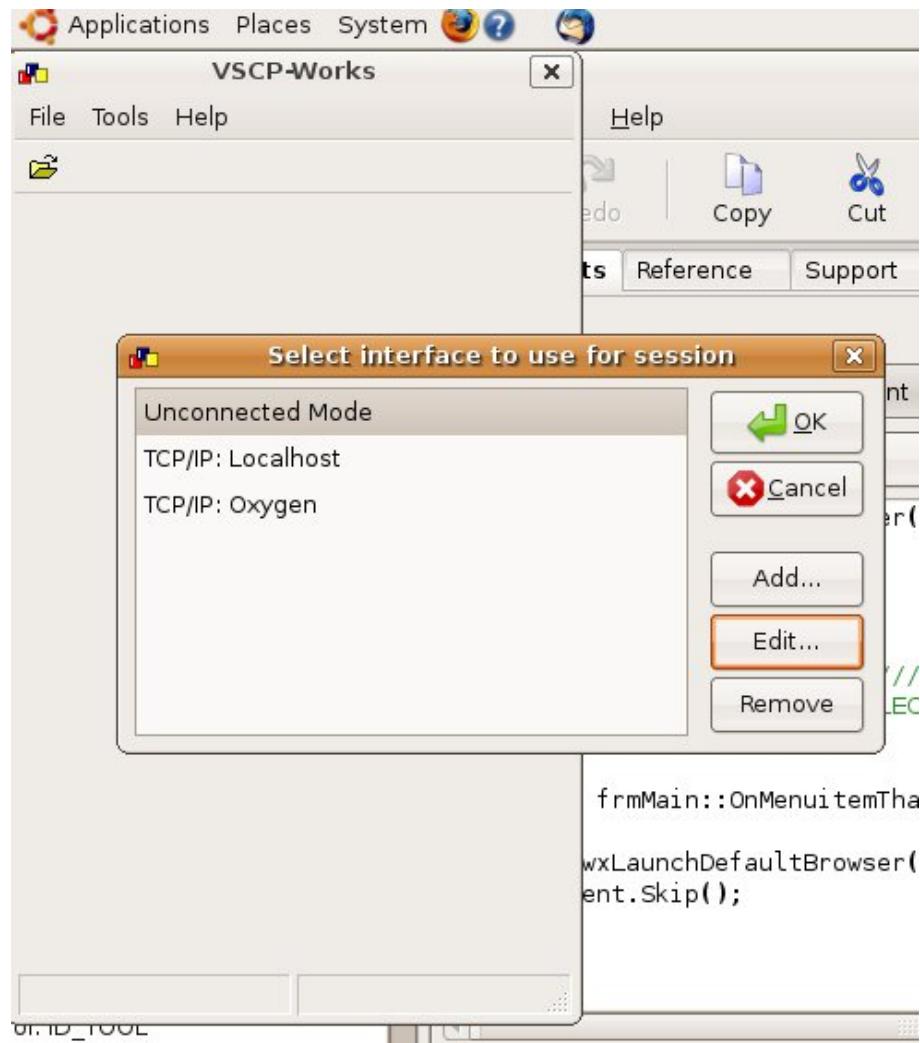
15.2 The VSCP Client Window

With the client window of VSCP works it is possible to open communication sessions either

- Directly to a hardware driver. Depending how the driver, one or more sessions can be open to the driver. The normal is one.
- Open a session to a local or remote VSCP daemon which in turn has drivers connected to it.

Use the File/New VSCP Client window to open a new client window. You can now select a predefined interface to connect to or create a new session.

Below is a picture of how the window looks as on Ubuntu Linux



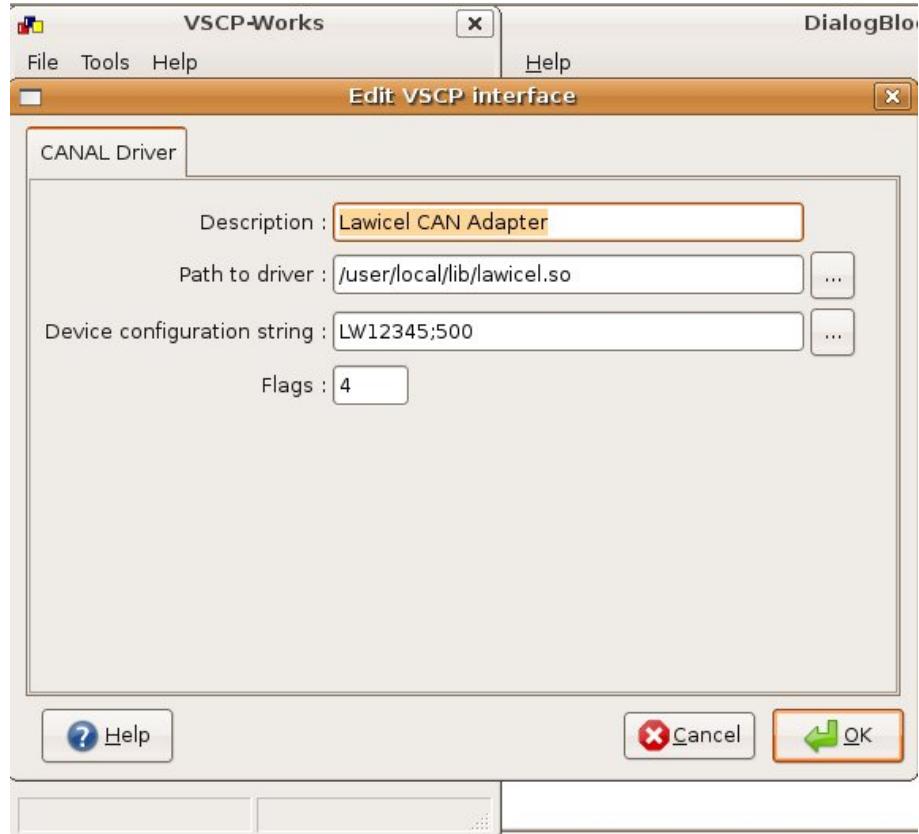
and on windows

FIXME

Remote interfaces have "TCP/IP:" in front of them and local interface or direct connections to CANAL drivers have "CANAL:" in front of them. The first type listed is the "Unconnected Mode" which opens the window without a connection either to a remote server or a CANAL driver. This is here to able to investigate saved session data.

With the Add button a new TCP/IP or CANAL interface can be added. With the Edit button an available interface setting can be edited and last the remove button can be used to remove an interface.

The image below shows how the CANAL session edit window looks like on Ubuntu Linux



and on windows

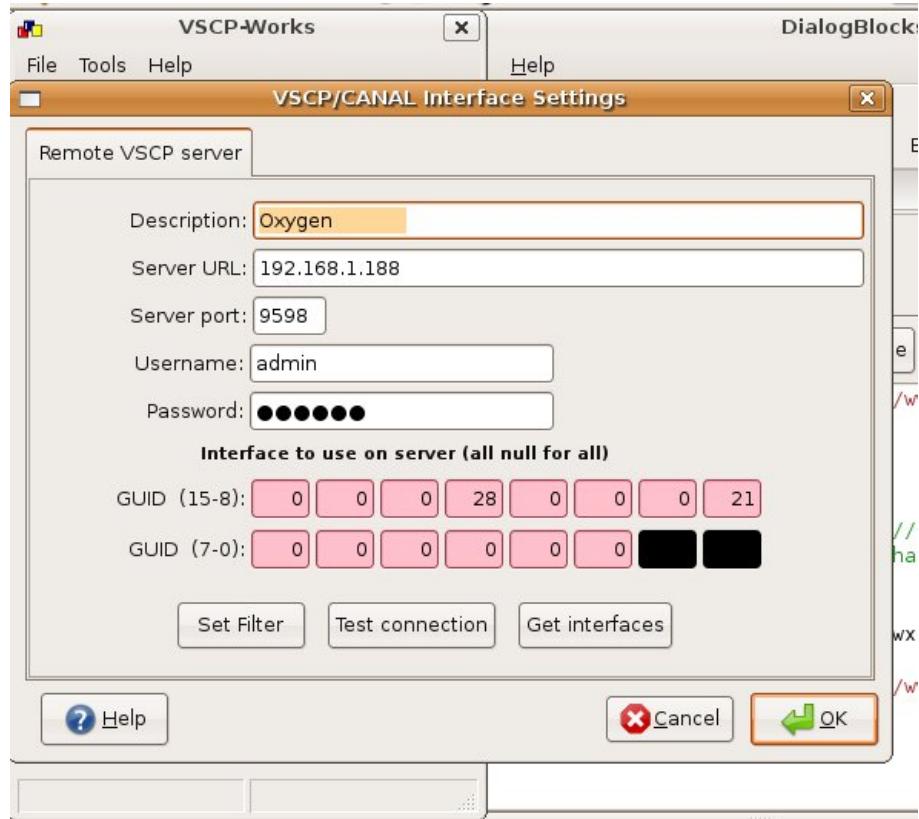
FIXME

Standard CANAL interface settings are available here.

- *Description* - Set a descriptive name here for use in listboxes etc.
- *Path to Driver*. The box to the right allows location to be found on the system.
- *Device configuration string*. The string used to configure the driver.
- *Flags*. The numerical flag value used to configure the driver.

A CANAL driver can have an internal XML file stored that describes the configuration string. If the driver has such information the button to the right of the configuration string can be used to run a wizard that helps in setting up both the flags value and the configuration string without needing to find the drivers documentation.

The image below shows how the remote TCP/IP server edit window looks like on Ubuntu Linux



and on windows

FIXME

For the remote TCP/IP interface some parameters need to be set also

- Description - Set a descriptive name here for use in listboxes etc.
- Server URL. Hostname for server
- Server port. Port that VSCP server is listening on. Normally 9598.
- Username - Username used to login to server.
- Password - Password used to login to server. Leave bland to have server ask before connecting.

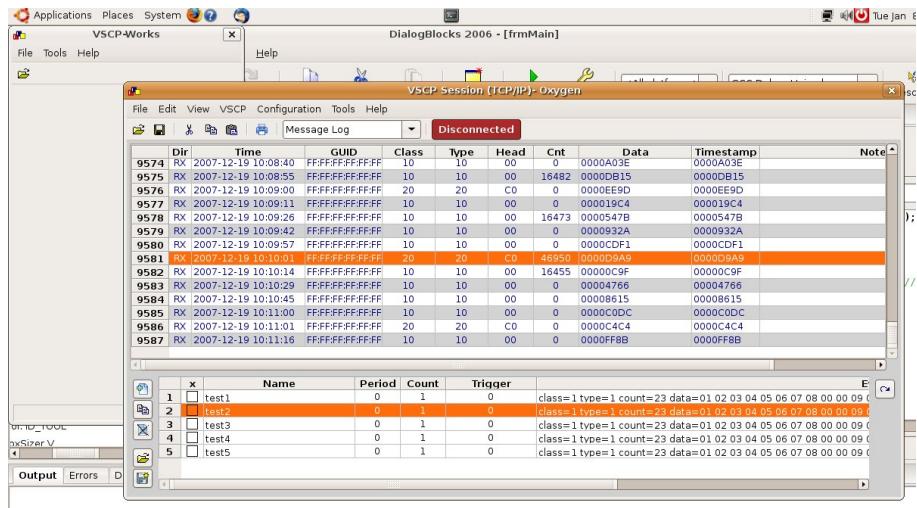
The GUID for the interface to use on the server should be set to all zero for a standard server connection. In this case a sent event from our client will be sent on all other interfaces. If a valid interface GUID is entered here all send events will only go to devices on the selected interface. As a help to set the correct interface *Get Interface* button is available which will fetch all available interfaces from the server and allow a selection among them in a listbox.

Use the *Set Filter* button to set an incoming filter from this session. Only events that satisfy the filter/mask combination will be received.

Use the *Test Connection* button to test the connection to the server.

Selecting one of the interfaces and pressing OK (or double clicking the line) opens the session window

Ubuntu Linux session window



and on windows

FIXME

The VSCP communication session window is divided into two areas. The upper area is the receive list and the lower area is the transmission list. Events that are received for this client is listed in the receive list also transmitted events are showed here. Further more it is possible to save the list to disk and later load it for investigation.

15.2.1 Receive event list

Currently only a chronological view is available (message log). Later another view will be added that makes it possible to see how many events of a certain class/type pair that has been received (message count).

By right clicking on the receive list some functionality will be available

- Edit the selected row.
- Add a note for the selected row.
- Copy/Cut/paste rows.
- Transmit one or many selected rows.
- Edit and transmit one or many selected rows.

- Save selected row(s) as transmission objects.
- Load VSCP events from file.
- Save VSCP events from file.

Naturally it is also possible to clear the receive list window.

15.2.2 Transmission object list

In the transmission object list rows are available that can be transmitted on the interface. It is possible to load and save transmission row sets which can be handy in many situations. It is possible to create rows that are automatically transmitted with a selected period expressed in milliseconds and which continue to do so as long as they are active. It is also possible to create transmission row objects that automatically send out one or more event when another event is received.

On the right of the Transmission object list is a set of buttons



and on windows

FIXME

that control rows in the list.

The first button add a new transmission object. The second button edit a selected row and the third button delete a row.

The forth button load a transmission row set from a file and the fifth button save the current rows.

On the right is a single button



and on windows

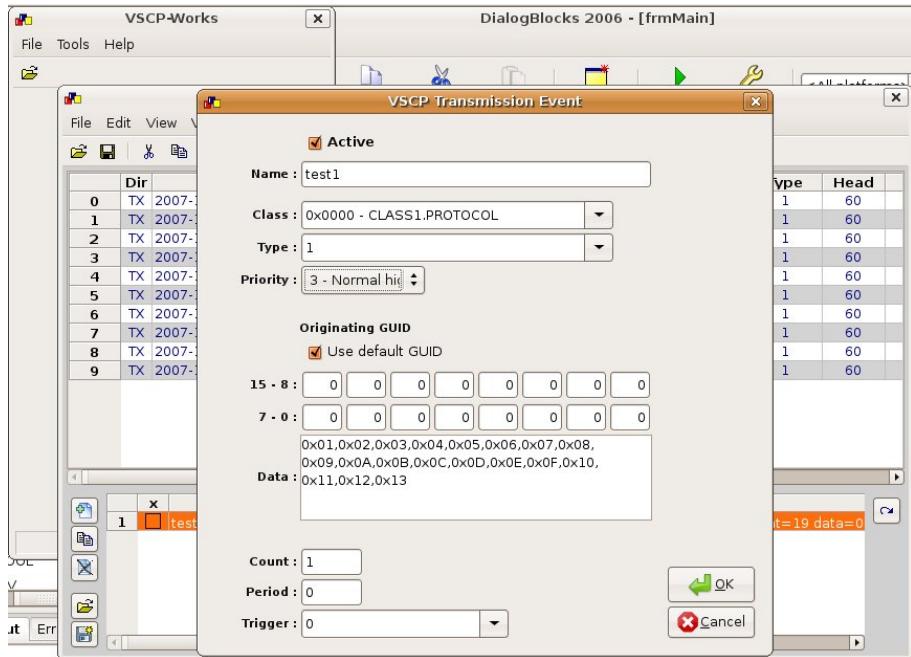
FIXME

This button is used to send the selected row(s). A single row can also be sent by double clicking on it but by selection several rows and clicking on this button the rows will be sent in sequence.

Some additional functionality is available by right clicking on the transmission object list.

- Transmit selected row(s).
- Add/edit/Delete rows.
- Clone row.
- Enable/disable periodic transmission.

The transmission row edit window looks like this on Ubuntu Linux



and like this on Windows

FIXME

Standard settings for events are available here. To use the GUID assigned by the daemon as the originator for the event set GUID to all zeros.

Data for the event is just a comma separated (mixed if needed) list of hexdecimal or decimal values.

Count is the number of sends of the transmission row.

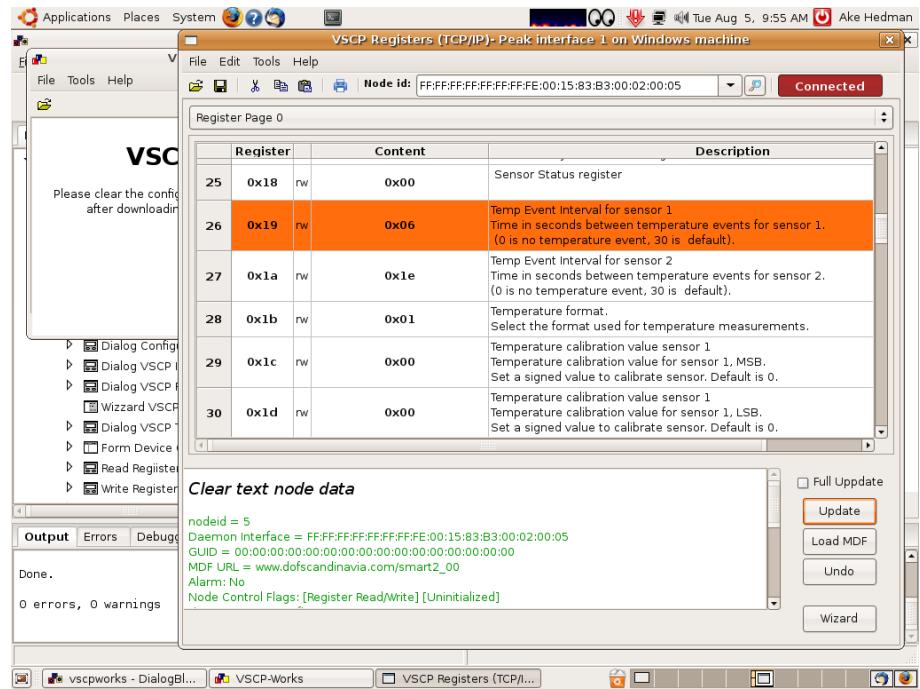
Period is the time in milliseconds between automatically transmitted rows.

Trigger is a selected trigger among the available triggers.

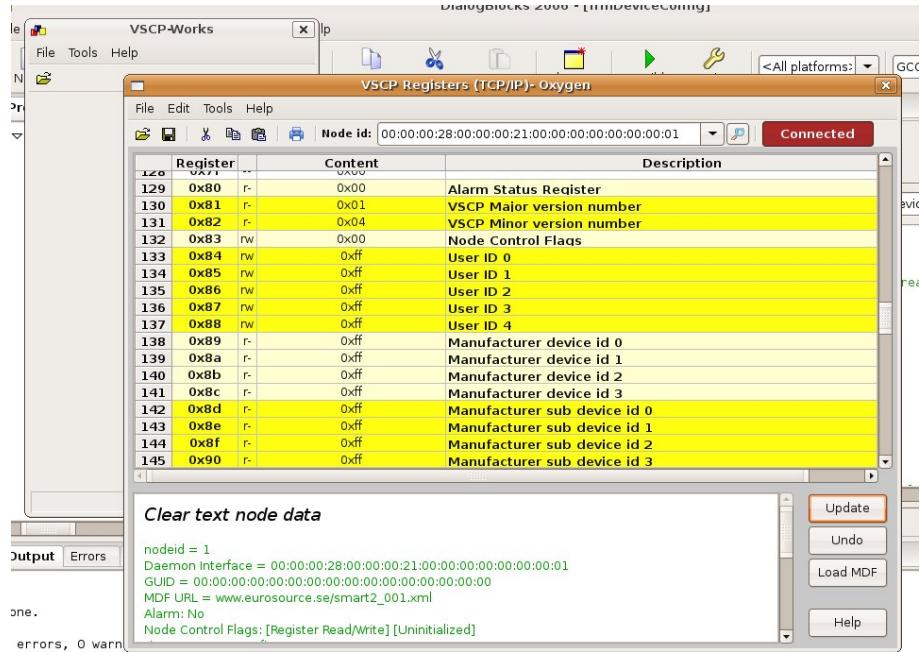
15.3 The Node configuration Window

The node configuration window can be used to set parameters for a device.

On Ubuntu Linux this window looks like this



In the above screen shot I have highlighted the registry entry that set the interval for temperature reports on the Kelvin module. To change it just click on the value and update. The operations are the same if the module is locally connected or located on the other side of the earth.



And on Windows

FIXME

Before this windows a session must be selected just as for the communication settings window. It is possible to connect through a local CANAL driver or a remote TCP/IP server. The only difference is that you need to enter a valid nickname-ID for a CANAL session before clicking update and that this ID must be a full interface ID with the least significant byte set to the nickname-ID of the device on that interface.

The update loads all registers from the device, both custom and standard. A clear text field is available at the bottom of the screen where data is presented in a more human suitable form. For instance the MDF URL is in clear text etc.

After updating the registers they can be edited and have their register content changed and after that another click on the update button will write the content back.

The undo button can be used to reset all register to the content they had when the register session was opened.

In the clear register mode VSCP Work only recognizes standard registers. It is therefore impossible for it to differentiate between read only and read/write registers in the custom register space. The MDF button address this.

After an update a click on the MDF button will automatically download the MDF file belonging to the device. It will then parse it and use the correct descriptions also for the custom registers. Further more now the abstractions can be used and entering register values can be done in a more human friendly way,



The node-id field looks different when connected to a remote server as above or to a local CANAL driver. For a remote server the interface GUID is part of the ID for the node and it tells which interface to talk to on the server. The least significant byte is the node-ID for the device connected to that interface. For a session connected directly to a CANAL device just the node-ID is entered here.

To the right of the node-ID field is a *search button*. It can be used to check that a node with the entered node-ID really is present.

The *connected/disconnected* button tell if the connection to the server is active or not.

15.4 The Bootloader Window

FIXME

15.5 The Simulated Node Window

FIXME

15.6 Shortcuts

[[VSCP Works shortcuts]]

15.7 FAQ

[[VSCP Works FAQ]]

15.8 Technical Information

15.8.1 ===== VSCP Works configuration file format =====

```
<code xml> <?xml version = "1.0" encoding = "UTF-8" ?>
<!-- Version 0.0.1 2007-10-26 -->
<vscpworksconfig>
    <general>
        <width>Client window startup width</width>
        <height>Client window startup height</height>
        <xpos>Client window X startup position</xpos>
        <ypos>Client window Y startup position</ypos>
        <path2tempfiles>Path to folder holding temporary files</path2tempfiles>
        <path2logfile enable="true|false">
            level="1">Path to log file</path2logfile>
    </general>
    <vscpclient>
        <VscpRcvFrameRxTextColour
            r="value"
            g="value"
            b="value">
```

```

</VscpRcvFrameRxTextColour>
<VscpRcvFrameRxBgColour
    r="value"
    g="value"
    b="value">
</VscpRcvFrameRxBgColour>
<VscpRcvFrameTxTextColour
    r="value"
    g="value"
    b="value">
</VscpRcvFrameTxTextColour>
<VscpRcvFrameTxBgColour
    r="value"
    g="value"
    b="value">
</VscpRcvFrameTxBgColour>
<VscpRcvFrameLineColour
    r="value"
    g="value"
    b="value">
</VscpRcvFrameLineColour>
<VscpRcvFrameFont></VscpRcvFrameFont>

<!-- Enable/Disable grey background on odd rows -->
<VscpRcvPyjamasStyle>true|false</VscpRcvPyjamasStyle>

<!-- Enable/Disable autoscroll when filling grid -->
<Autoscroll>true|false</Autoscroll>
<VscpRcvShowField0 enable="true|false"></VscpRcvShowField0>
<VscpRcvShowField1 enable="true|false"></VscpRcvShowField1>
<VscpRcvShowField3 enable="true|false"></VscpRcvShowField3>
<VscpRcvShowField4 enable="true|false"></VscpRcvShowField4>
<VscpRcvShowField5 enable="true|false"></VscpRcvShowField5>
<VscpRcvShowField6 enable="true|false"></VscpRcvShowField6>
<VscpRcvShowField7 enable="true|false"></VscpRcvShowField7>
<VscpRcvFieldText0>Dir</VscpRcvFieldText0>
<VscpRcvFieldText1>Time</VscpRcvFieldText1>
<VscpRcvFieldText2>GUID</VscpRcvFieldText2>
<VscpRcvFieldText3>Class </VscpRcvFieldText3>
<VscpRcvFieldText4>Type</VscpRcvFieldText4>
<VscpRcvFieldText5>Head</VscpRcvFieldText5>
<VscpRcvFieldText6>Cnt</VscpRcvFieldText6>
<VscpRcvFieldText7>Data</VscpRcvFieldText7>
<VscpRcvFieldText8>Timestamp</VscpRcvFieldText8>
<VscpRcvFieldText9>Note </VscpRcvFieldText9>
<VscpRcvFieldOrder0>0</VscpRcvFieldOrder0>
<VscpRcvFieldOrder1>1</VscpRcvFieldOrder1>
<VscpRcvFieldOrder2>2</VscpRcvFieldOrder2>
<VscpRcvFieldOrder3>3</VscpRcvFieldOrder3>
<VscpRcvFieldOrder4>4</VscpRcvFieldOrder4>
<VscpRcvFieldOrder5>5</VscpRcvFieldOrder5>
<VscpRcvFieldOrder6>6</VscpRcvFieldOrder6>
<VscpRcvFieldOrder7>7</VscpRcvFieldOrder7>
<VscpRcvFieldOrder8>8</VscpRcvFieldOrder8>
<VscpRcvFieldOrder9>9</VscpRcvFieldOrder9>
<VscpRcvFieldWidth0>30</VscpRcvFieldWidth0>
<VscpRcvFieldWidth1>110</VscpRcvFieldWidth1>
<VscpRcvFieldWidth2>250</VscpRcvFieldWidth2>
<VscpRcvFieldWidth3>70</VscpRcvFieldWidth3>
<VscpRcvFieldWidth4>70</VscpRcvFieldWidth4>
<VscpRcvFieldWidth5>60</VscpRcvFieldWidth5>
<VscpRcvFieldWidth6>60</VscpRcvFieldWidth6>
<VscpRcvFieldWidth7>300</VscpRcvFieldWidth7>
<VscpRcvFieldWidth8>90</VscpRcvFieldWidth8>
<VscpRcvFieldWidth9>200</VscpRcvFieldWidth9>
<VscpTrmitFrameTextColour
    r="value"
    g="value"
    b="value">

```

```

        b="value">
    </VscpTrmitFrameTextColour>
    <VscpTrmitFrameBgColour
        r="value"
        g="value"
        b="value">
    </VscpTrmitFrameBgColour>
    <VscpTrmitFrameLineColour
        r="value"
        g="value"
        b="value">
    </VscpTrmitFrameLineColour>
    <VscpTrmitFrameFont></VscpTrmitFrameFont>
    <VscpTrmitShowField0 enable="true | false"></VscpTrmitShowField0>
    <VscpTrmitShowField1 enable="true | false"></VscpTrmitShowField1>
    <VscpTrmitShowField2 enable="true | false"></VscpTrmitShowField2>
    <VscpTrmitShowField3 enable="true | false"></VscpTrmitShowField3>
    <VscpTrmitFieldText0></VscpTrmitFieldText0>
    <VscpTrmitFieldText0></VscpTrmitFieldText0>
    <VscpTrmitFieldText1></VscpTrmitFieldText1>
    <VscpTrmitFieldText2></VscpTrmitFieldText2>
    <VscpTrmitFieldText3></VscpTrmitFieldText3>
    <VscpTrmitFieldText4></VscpTrmitFieldText4>
    <VscpTrmitFieldOrder0></VscpTrmitFieldOrder0>
    <VscpTrmitFieldOrder1></VscpTrmitFieldOrder1>
    <VscpTrmitFieldOrder2></VscpTrmitFieldOrder2>
    <VscpTrmitFieldOrder3></VscpTrmitFieldOrder3>
    <VscpTrmitFieldOrder4></VscpTrmitFieldOrder4>
    <VscpTrmitFieldWidth0></VscpTrmitFieldWidth0>
    <VscpTrmitFieldWidth1></VscpTrmitFieldWidth1>
    <VscpTrmitFieldWidth2></VscpTrmitFieldWidth2>
    <VscpTrmitFieldWidth3></VscpTrmitFieldWidth3>
    <VscpTrmitFieldWidth4></VscpTrmitFieldWidth4>
</vscpcient>
<vscpinterface>

    <canaldriver> <!-- One or many -->
        <description>Interface description for listbox</description>
        <path>oath to driver</path>
        <config>configstring</config>
        <flags>driver flags</flags>
    </canaldriver>

    <vscpdaemon> <!-- One or many -->
        <description>Interface description for listbox</description>
        <host></host>
        <port></port>
        <username></username>
        <password></password>
        <filter priority="xxx"
            class="xxx"
            type="xxx"
            GUID="xxx" >This is the row filter</filter>
        <mask priority="xxx"
            class="xxx"
            type="xxx"
            GUID="xxx" >This is the row mask</mask>
    </vscpdaemon>

</vscpinterface>
</vscpworksconfig>
</code>

```

15.8.2 VSCP Works receive/transmission data file format

<code xml>

```

<?xml version = "1.0" encoding = "UTF-8" ?> <!-- Version 0.0.1 2007-11-27
<vscprxdata>
    <event>
        <dir>rx|tx</dir>
        <time>(real)-time when event was received</time>
        <head>priority etc</head>
        <class>class code</class>
        <type>type code</type>
        <guid>XX:YY:ZZ.....</guid>
        <data>Comma separated list with data values</data>
        <timestamp>Relative timestamp for event</timestamp>
        <note>User note about line</note>
    </event>
</vscprxdata>
</code>

```

15.8.3 VSCP Works Transmission set file format

```

<code xml>
    <?xml version = "1.0" encoding = "UTF-8" ?>
    <!-- Version 0.0.1 2007-11-27 -->
    <vscptxset>
        <set>
            <active>true | false</active>
            <name>Descriptive name for transmission line</name>
            <class>class code</class>
            <type>type code</type>
            <priority>priority 0-7</priority>
            <guid default="yes | no">xx.yy.xx..... </guid>
            <data>Comma separated list with data values</data>
            <count>Count value</count>
            <period>Period value</period>
            <trigger>Code for trigger</trigger>
        </set>
    </vscptxset>
</code>

```

16 vscpcmd

vscpcmd is a small command line tool that can be used send and receive events from the VSCP daemon.

16.1 Command line switches

16.1.1 -e/-event

event data on the form *head, class, type, obid, timestamp, G UID, data1, data2, data3...*

16.1.2 -m/-measurement

Set this event as a measurement event CLASS1.MEASUREMENT. Argument can be given as three bytes as **format,data-unit,index**

Format

- 0 or 0x00 - Bit format
- 1 or 0x20 - Byte format
- 2 or 0x40 - String format.
- 3 or 0x60 - Integer format.
- 4 or 0x80 - Normalized integer format

Data-unit Bits 0-3 Measurement data-unit for example Kevin, Celsius, Fahrenheit. Defaults to zero.

index 0-7 Sensor index. Defaults to zero.

16.1.3 -y/-value

Encoding, VSCP type and Value to for a measurement or other data. A string should have enclosing ". ". Decimal numbers use a point "."

16.1.4 -q/-host

host as *user:password@host* or just *host*. Default user is “admin”. Default Password is “secret” and default host is “localhost”.

16.1.5 -u/-user

Username. Default user is “admin”.

16.1.6 -p/-password

Password. Default password is “password”.

16.1.7 -n/-count

Number of events to send or receive. Default password is one.

16.1.8 -z/-zone

Zone to use.

16.1.9 -s /--subzone

Subzone to use.

16.1.10 -t /--test

Interface test mode (for vscpd developers only).

16.1.11 -v /--verbose

Set verbose mode.

16.1.12 -h /--help

Set verbose mode.

16.1.13 Examples

example 1 Send A temperature measurement value as an integer in Celsius with the value 66 degrees Celsius.

```
vscpcmd --event= "0,10,6,0,0,FF:EE:DD:CC:BB:AA:99:88:77:66:55:44:00:00:00:00,0x68,0x42"--host=admin:secret@localhost
```

example 2 Send same as above but use the interface GUID

```
vscpcmd -e "0,10,6,0,0,-,0x68,0x42"--host=admin:secret@localhost
```

example 3 Send same as above but using the measurement switches

```
vscpcmd --measurement="4,1,0"--value="33.4"--host="admin:secret@localhost"
```

Part III

HTML5 user interface components & tools

17 HTML5 websocket widgets

To make it easier to use the websocket interface of the daemon a number of widgets that makes it easier to create dynamic HTML5 web pages has been developed. Also a general JavaScript library is available that both help in developing and interfacing the VSCP websocket interface of the VSCP daemon.

17.1 vscpws_stateButton

This information is preliminary and can change at any time.



The *vscpws_stateButton* is a button that can be used as a state button or as a stateless button. It has the ability to send an event when it changes its state (one for each state) or for a stateless button one when it is pressed and one when released. Also the button can change its state according to incoming events or a monitored VSCP Daemon variable value. The button can have one of several appearances shown below or a custom appearance defined by the user.

To assign a monitored variable to the button is a good solution if a state should be monitored over time. Instead of waiting for the next event that tell the button state a variable is read from the daemon which is kept persistent and up to date to get the state. This variable is written/changed in the decision matrix or by a user or an application through the tcp/ip interface of the daemon and can thus be influenced and controlled by some simple or advanced criteria. This also can be used to monitor the state for a specific condition by monitor a variable and just show the state on the page not allowing clicking the button. Typical scenarios can be to indicate an open window, an alarm condition or things like that.

The decision matrix of the VSCP daemon is also perfect to use if one wants to have a button that sends several events. This can be anything of course. Some dimmer commands combined with some on/off's and some curtain roll downs. This is then just added to the decision matrix and triggered on the event sent from the button. Typical use is to control for example any lamps from one button to set a specified scenery for a house or a stage.

A **state button** as default send

CLASS1.CONTROL, TurnOn index=0, zone=0, subzone=0

when activated and

CLASS1.CONTROL, TurnOff index=0, zone=0, subzone=0

when inactivated. The button will change its state either when a monitored variable change its state or go to the active state when a

CLASS1.INFORMATION, On-event index=0, zone=0, subzone=0

is received and go to its inactive state when a

CLASS1.INFORMATION, off-event index=0, zone=0, subzone=0

is received.

A **stateless button** as default send

CLASS1.CONTROL, TurnOn index=0, zone=0, subzone=0

when pressed and

```
CLASS1.CONTROL.TurnOff index=0, zone=0, subzone=0
```

when released.

Creating a button is typically done with

```
var btn1 = new vscpwstateButton( "ws://192.168.1.20:7681" , "buttonKitchen");
```

which will create a default state button of type=0 looking like this. *Here "buttonKitchen"* refers to a canvas where the button is placed as explained below.

If the connection to the server is lost for some reason stateButtons will look like this



where the cross over them indicate a broken connection to the server.

17.1.1 Parameters

```
bt1 = vscpws_stateButton( url , // url to VSCP websocket i/f  
                          canvasName ,  
                          // Where it should be placed  
                          bLocal ,  
                          // Local visual control  
                          btnType ,  
                          // Button type  
                          bNoState ,  
                          // True for nonstate button  
                          bDisable ,  
                          // Disable user interactions  
                          customupsrc ,  
                          // Custom up image  
                          customdownsrc ) // Custom
```

url This is the url to the websocket server. This typically is on the form

```
"ws://192.168.1.20:7681"
```

With all widgets having their own url specified for the websocket server it is possible to create web pages that control nodes/units/hardware that are located in different locations from the same page.

canvasName This is the name of the canvas element where the button should be placed. Typically this is defined on the form

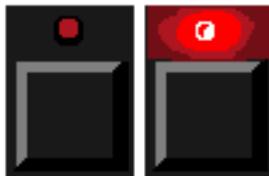
```
<canvas id="buttonKitchen"  
        style="z-index: 3;  
               position: absolute;  
               left:190px;  
               top:310px;" height="30px" width="22px">  
    Your browser does not support HTML5 Canvas.  
</canvas>
```

The *id* is the parameter that goes for the *canvasName*. This name is also used to create the instance name for the button and the name set is preceded with *vscpws_*. The canvas specifies the position and the size for the button. Also z-order is possible to define so that objects can be placed behind, or partially behind each other to get nice visual effects.

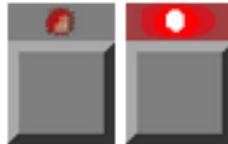
bLocal If set to true visual indication is handled locally. For a state button appearance will be changed on every click and for a stateless button appearance will be changed when the button is pressed and then again when it is released. It is better to have the visual appearance of the buttons controlled by external events as it also a confirmation that the sent event actually did what it was intended to do. In some situations however it may be good to handle this locally for some reason. Default=false

btnType This is the visual appearances of the button. Check for possible values below. -1 is reserved for a user defined button. When used one has to supply a path to an image for both the "down" and the "up" state for the button in the *customupsrc* and *customdownsrc* parameters. Default = 0.

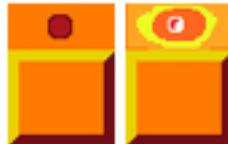
Appearances types The images below is shown in there default size. Most of the buttons is designed for a different size and with an appropriate background color.



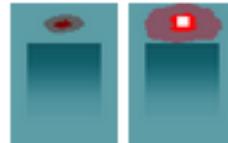
Type = 0 Default



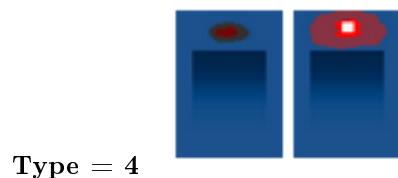
Type = 1



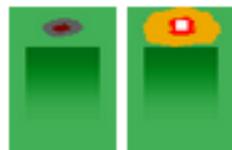
Type = 2



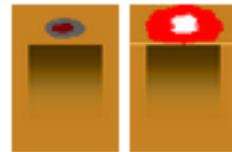
Type = 3



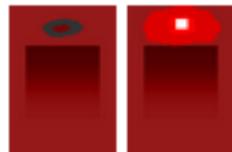
Type = 4



Type = 5



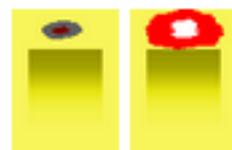
Type = 6



Type = 7



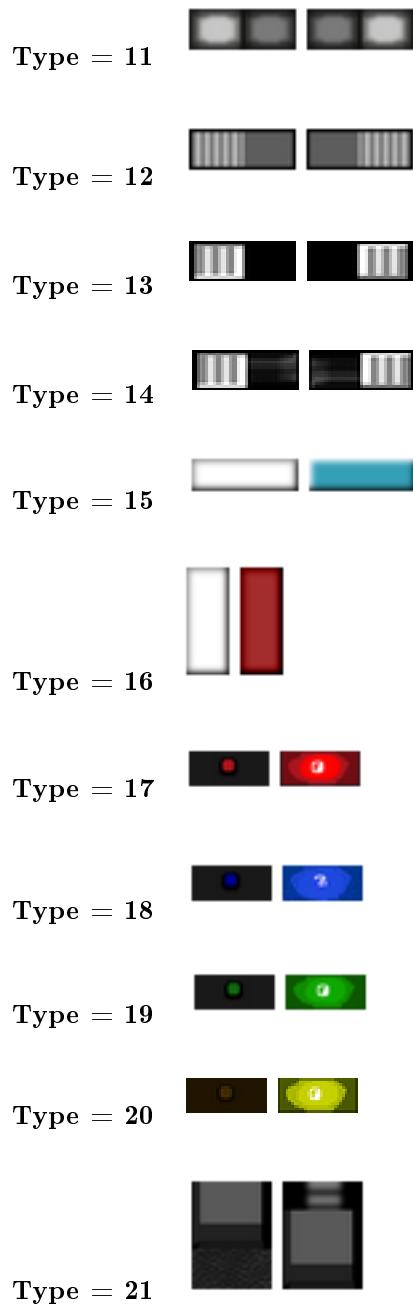
Type = 8

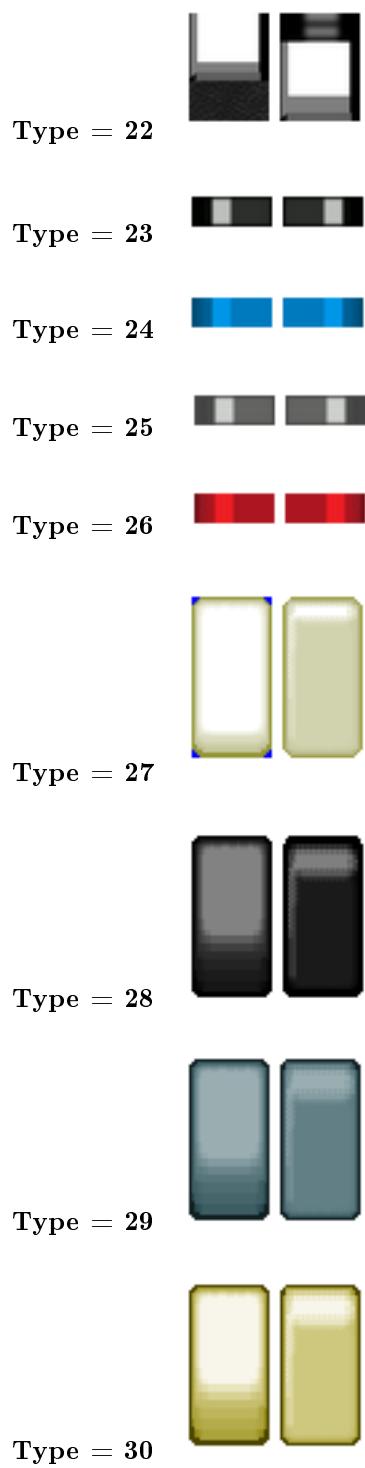


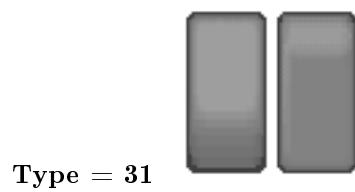
Type = 9



Type = 10



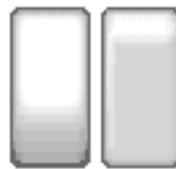




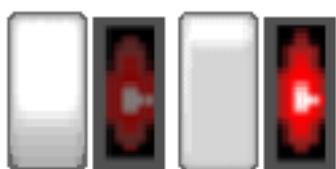
Type = 31



Type = 32



Type = 33



Type = 34



Type = 35



Type = 36



Type = 37



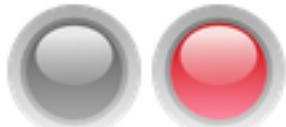
Type = 38



Type = 39



Type = 40



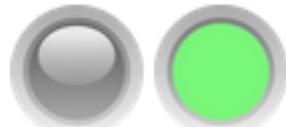
Type = 41



Type = 42



Type = 43



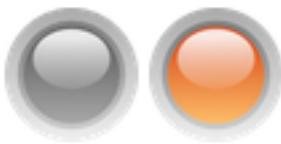
Type = 44



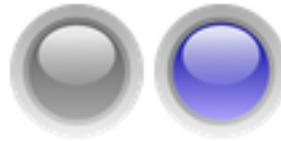
Type = 45



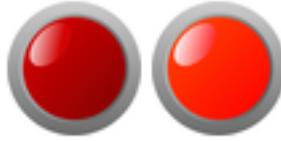
Type = 46



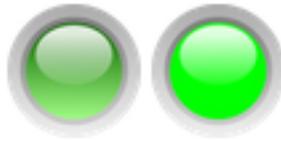
Type = 47



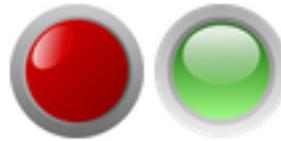
Type = 48



Type = 49



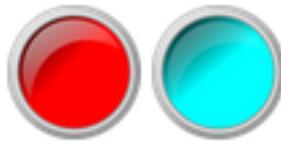
Type = 50



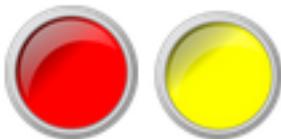
Type = 51



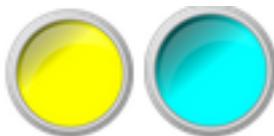
Type = 52



Type = 53



Type = 54



Type = 55



Type = 56



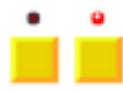
Type = 57



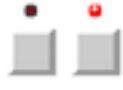
Type = 58



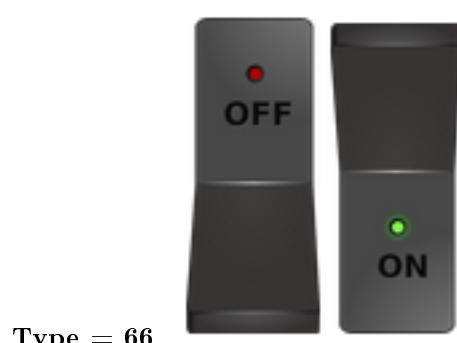
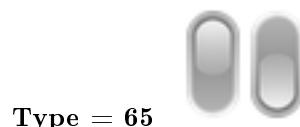
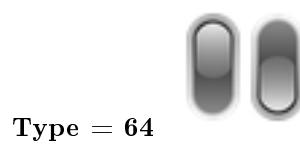
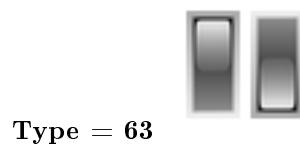
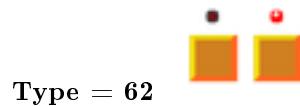
Type = 59



Type = 60

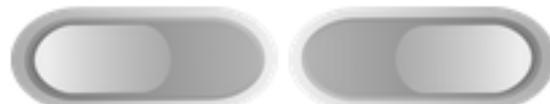


Type = 61

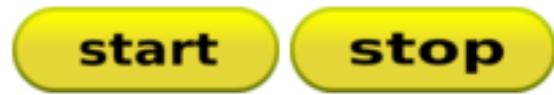




Type = 69



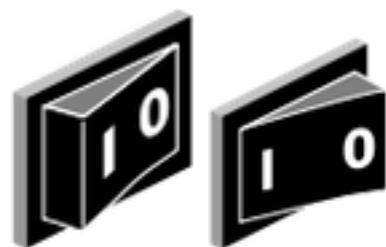
Type = 70



Type = 71



Type = 72



Type = 73



Type = 74



Type = 75



Type = 76



Type = 77



Type = 78



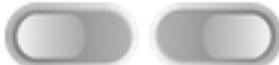
Type = 79



Type = 80

PWR OFF PWR ON

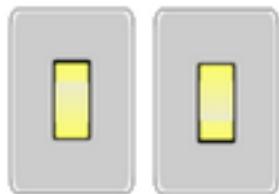
Type = 81



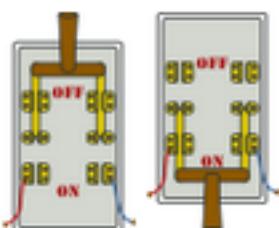
Type = 82



Type = 83



Type = 84



Type = 85



Type = 86



Type = 87



Type = 90



Type = 91 Blinking green and blinking red.



Type = 92



Type = 93



bNoState Set to true to have a stateless button. This button will send one event when pressed and one when released. The default is to send

CLASS1.CONTROL, TurnOn index=0, zone=0, subzone=0

when pressed and

CLASS1.CONTROL, TurnOff index=0, zone=0, subzone=0

when released. Default = false;

bDisable Set to true to disable user interactions. Button clicks will no longer work. This is useful if one instead wants to monitor a variable. Default is false.

customupsrc This is the path to the "up" image for a custom button when type is -1.

customdownsrc This is the path to the "down" image for a custom button when type is -1.

17.1.2 Methods

setOnTransmittEvent Set the event that is sent when a state button is activated or a stateless button is pressed down.

Usage

```
setOnTransmittEvent( vscpclass , // VSCP class
                     vscptype ,
                     // VSCP type
                     data ,
                     // Array with databytes
                     guid )
                     // GUID to use
```

vscpclass VSCP class to use for the event. Default=30 CLASS1.CONTROL

vscptype VSCP type to use for the event. Default=5 as for CLASS1.CONTROL TurnOn-Event.

data Array with data bytes to use for the event. Its the responsibility of the caller to use the correct number of data bytes. Default is "0,0,0" meaning index=0, zone=0, subzone=0,

guid This is the sending GUID. Normally one just use the GUID of the sending interface of the VSCP daemon. So the default is 00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00 or "-" with is a short hand version for the long form. Default="-"

If you don't want an event to be sent for the on-state set vscpclass=0.

setOffTransmittEvent Set the event that is sent when a state button is activated or a stateless button is pressed down.

```

setOffTransmittEvent( vscpclass , // VSCP class
                      vscptype , // VSCP type
                      data ,
// Array with databytes
                      guid )
// GUID to use

```

vscpclass VSCP class to use for the event. Default=30 CLASS1.CONTROL

vscptype VSCP type to use for the event. Default=6 as for CLASS1.CONTROL TurnOff.

data Array with data bytes to use for the event. Its the responsibility of the caller to use the correct number of data bytes. Default is "0,0,0" meaning index=0, zone=0, subzone=0,

guid This is the sending GUID. Normally one just use the GUID of the sending interface of the VSCP daemon. So the default is 00:00:00:00:00:00:00:00:00:00:00:00:00:00 or "-" which is a short hand version for the long form. Default="-"

If you don't want an event to be sent for the off-state set vscpclass=0.

setOnReceiveEvent Set the event that should be received to set the visual indication of a button to it's on state.

Usage

```

setOnReceiveEvent( vscpclass , // VSCP class
                      vscptype , // VSCP type
                      data , // Array with databytes
                      guid ) // GUID to use

```

vscpclass

VSCP class that the incoming event must have. Default=20 CLASS1.INFORMATION

vscptype VSCP type that the incoming event must have. Default=3 as of the CLASS1.INFORMATION On-Event.

data Array with data bytes that the event should have. Set a data byte to -1 for a don't care. Default is "-1,0,0" meaning index=-1, zone=0, subzone=0,

guid This is the GUID the receiving event should have. Just leave undefined if the GUID should not be checked. Default=""

setOffReceiveEvent Set the event that should be received to set the visual indication of a button to it's on state.

Usage

```
setOffReceiveEvent (vscpclass ,      // VSCP class  
                    vscptype , // VSCP type  
                    data ,  
                    // Array with databytes  
                    guid)  
// GUID to use  
vscpclass  
VSCP class that the incoming event must have. Default=20 CLASS1.INFORMATION
```

vscptype VSCP type that the incoming event must have. Default=3 as of the CLASS1.INFORMATION Off-Event.

data Array with data bytes that the event should have. Set a data byte to -1 for a don't care. Default is "-1,0,0" meaning index=-1, zone=0, subzone=0,

guid This is the GUID the receiving event should have. Just leave undefined if the GUID should not be checked. Default=""

setOnTransmittZone As it is common to use one of the types in CLASS1.CONTROL as on-event for a button this is a convenient method to use to change the default zone/subzone for the outgoing event.

Usage

```
setOnTransmittZone (zone ,      // zone  
                     subzone , // subzone  
zone  
Zone to direct event to. 255 is all.
```

subzone Subzone to direct event to. 255 is all.

setOffTransmittZone As it is common to use one of the types in CLASS1.CONTROL as off-event for a button this is a convenient method to use to change the default zone/subzone for the outgoing event.

Usage

```
setOffTransmittZone (zone ,     // zone  
                     subzone) // subzone  
zone  
Zone to direct event to. 255 is all.
```

subzone Subzone to direct event to. 255 is all.

setOnReceiveZone As it is common to use one of the types in CLASS1.INFORMATION as a trigger for the on-state of a button this is a convenient method to use to change the default zone/subzone for the incoming event.

Usage

```
setOnReceiveZone(zone,           // zone
                  subzone) // subzone
```

zone

Zone to direct event to. 255 is all.

subzone Subzone to direct event to. 255 is all.

setOffReceiveZone As it is common to use one of the types in CLASS1.INFORMATION as a trigger for the off-state of a button this is a convenient method to use to change the default zone/subzone for the incoming event.

Usage

```
setOffReceiveZone(zone,           // zone
                  subzone) // subzone
```

zone

Zone to direct event to. 255 is all.

subzone Subzone to direct event to. 255 is all.

setMonitorVariable With this method one can set a VSCP daemon boolean variable that should be monitored with a specific interval. If the variable is true the state for the state button will be set to true and vice versa. The variable is also updated the other way around If the button is down the variable will be set to true and vice versa. If the variable does not exist it will be created.

Usage

```
setMonitorVariable(variablename, // variable name
                   interval) // monitoring interval in
```

variablename The name for the VSCP daemon boolean variable.

interval The interval in milliseconds between variable reads. Set to zero to disable. Default=1000 (one second). To test try to set the variable name to "statbutton1" and issue

```
variable write statbutton1,,,true
```

and

```
variable write statbutton1,,,false
```

in the VSCP daemon TCP/IP interface to see the change the state for the state button in your browser page.

draw Draw the widget. Normally no need to use.

Usage

```
draw() // Draw the widget
```

setValue

Set the value for the widget. The state will also be

Usage

```
setValue( value , // Boolean value for state  
         bUpdate ) // True (default) if a draw should be done  
                      // after value is set .
```

value

Boolean value that set the state of the widget.

bUpdate True (default) if a draw should be done after value is set.

17.1.3 Using the vscpws_stateButton widget

TODO

17.2 vscpws_simpleText

This information is preliminary and can change at any time.

The vscpws_simpleText is a widget for formatted dynamic text that can be displayed on a web page. This may not sound very exciting but it is. In the past Ajax was used for this and the server was polled for data. Here websockets are used giving a much more lightweight data handling. The text can be a dynamically updated. Temperature value or some other measurement which is formatted nicely as you have all formatting capabilities of HTML5 at your disposal. You can display information from received events or data from a VSCP daemon variable.

17.2.1 Function arguments

```
vscpws_simpleText( url , // url to VSCP websocket i/f  
                    id , // Where it should be p  
                    vscpclass , // VSCP event class e.g. 1000  
                    vscptype , // VSCP Event type  
                    sensorindex , // Datacoding index  
                    decimals , // Number of decimals  
                    formatstr , // A value format string  
                    guid , // GUID we are interested in  
                    fncallback ) // If set function to call  
                                // when data ar
```

url This is the url to the websocket server. This typically is on the form "ws://192.168.1.20:7681". Address to the server and the port just as we are used to to request a standard web page.

With all widgets having there own url specified for the websocket server it is possible to create web pages that control nodes/units/hardware that are located in different locations from the same page.

id This is the id for the HTML element where the result should be placed. This can be any HTML element which can display text and all formating capabilities of HTML can obviously be used.

vsciclass This is the the VSCP class of the event we are interested in. Normally this is one of the measurement events which have all SI units and more (usually derived units) specified. Set to -1 for don't care.

vscptype This is the the VSCP type of the event we are interested in. Set to -1 for don't care.

sensorindex This is the sensor index for the sensor in the device sending out the event. This information is part of the datacoding and can be a value 0,1,2,3,4,5,6,7. Set to -1 for don't care. Default=0.

decimals Number of decimals in the displayed value. Default is two.

formatstr This is a string that can be used to format the value before it is written to the HTML element. The string have a format on this form

"{0} degrees Celsius"

where the escapes in {n} will be replaced by real time data.

Escape	Description
{0}	Measurement value e.g "-12.5".
{1}	Measurement unit in real text form e.g. for temperature "Celsius", "Fahrenheit" etc.
{2}	Measurement unit in numerical form e.g. "0", "1", "2" etc.
{3}	Sensor index e.g. "0", "1", "2"..."7".

guid This can be used to match a specific GUID the event should come from. Default is "" meaning events from all sensors are of interest.

index This can be used to match a specific index. Default is -1 meaning don't care. Only VSCP_CLASS1_MEASUREZONE and VSCP_CLASS1_SETVALUEZONE have an index to check among the measurement classes.

zone This can be used to match a specific zone. Default is -1 meaning don't care. Only VSCP_CLASS1_MEASUREZONE and VSCP_CLASS1_SETVALUEZONE have a zone to check among the measurement classes.

subzone This can be used to match a specific subzone. Default is -1 meaning don't care. Only VSCP_CLASS1_MEASUREZONE and VSCP_CLASS1_SETVALUEZONE have a subzone to check among the measurement classes.

fncallback Here it is possible to enter a user defined function that get the value, the unit and the full event as its arguments. The function should have the following form

```
userfunction (value , unit , event)
```

This function should return a string that will be written to the HTML page or null if the callback do this write itself or for some other reason does not want the page to be updated. This can be useful if one wants to do some special handling with the event.

17.2.2 Methods

setExtraParameters To use the vscpw_simpleTextEvent widget with events in class VSCP_CLASS1_MEASUREZONE and VSCP_CLASS1_SETVALUEZONE you may want to filter on more than sensorindex and guid. This method allows the extra info available in these classes to be checked as well.

```
setExtraParameters( index ,           // Index if applicable
                    zone ,          // Zone if applicable
                    subzone)        // Subzone if applicable
```

index This can be used to match a specific index. Default is -1 meaning don't care. Only VSCP_CLASS1_MEASUREZONE and VSCP_CLASS1_SETVALUEZONE have an index to check among the measurement classes.

zone This can be used to match a specific zone. Default is -1 meaning don't care. Only VSCP_CLASS1_MEASUREZONE and VSCP_CLASS1_SETVALUEZONE have a zone to check among the measurement classes.

subzone This can be used to match a specific subzone. Default is -1 meaning don't care. Only VSCP_CLASS1_MEASUREZONE and VSCP_CLASS1_SETVALUEZONE have a subzone to check among the measurement classes.

17.2.3 Using the widget

Suppose you have a temperature sensor like the Kelvin NTC (<http://www.auto.grodansparadis.com/kelvinntc10>) which sends out temperature events at specific intervals. For this device you will get events like

VSCP CLASS = 10 i.e. CLASS1.MEASUREMENT we have a measurement event.

VSCP Type = 6 Telling that this is a temperature measurement.

DATA = 0x8A,0x02,0x09,0x72 telling that this is data from sensor 2 and that the

If you are interested in the format and want to understand how to decode it please see the specification 8.

To display this temperature measurement in a HTML page we can insert a line like this

```
<div> This is just some text in a div where a temperature is equal to <span id="
```

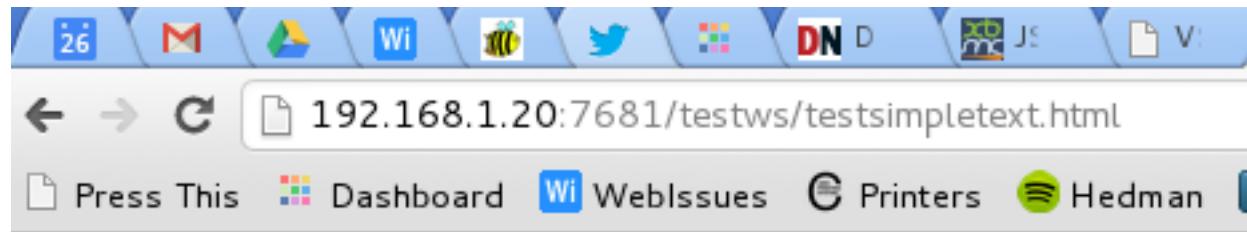
Nothing to fancy. The "id1" is the interesting part here. This where the temperature value will be placed. Naturally this span-element can have all attributes a HTML element can have just as the <div>. Also it can be some other HTML element that can display text.

To display values with two decimals from sensor 2 we insert this code in our HTML page

```
<script type="text/javascript">
    var txt1 = new vscpw_simpleTextEvent( "ws://192.168.1.20:7681" ,
```

```
</script>
```

This is all that is needed. A live web page is now available that updates the temperature with the interval set when configuring the sensor.



VSCP HTML5 websocket vscpw...

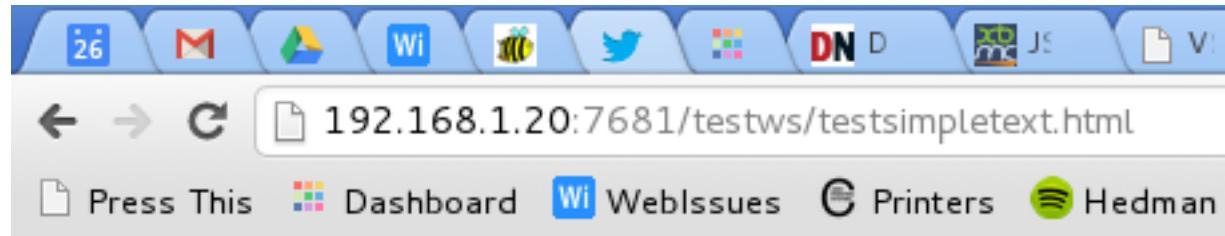
[Go back to main page](#)

This is just some text in a HTML element where a temperature is equal to 23

You can add a format string if you want to change the appearance of the output. Add a line to the above so it looks like this instead

```
<script type="text/javascript">
    var txt1 = new vscpw_simpleTextEvent( "ws://192.168.1.20:7681",
                                          "Temperature is %d degrees", 23);
</script>
```

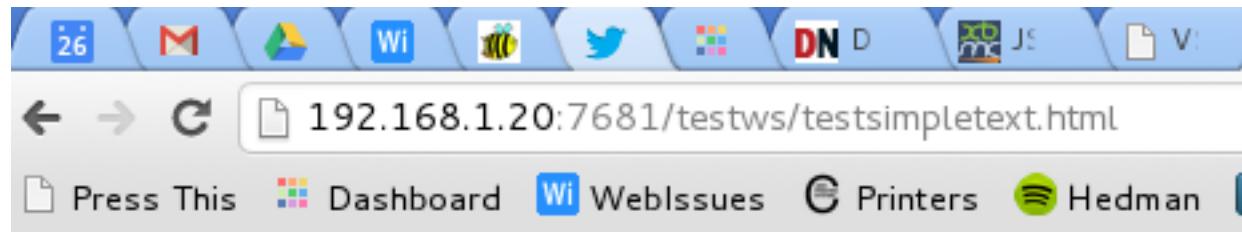
which will look like this



In the format string you can put in any text you like. {0} is the temperature value but there are also other escapes you can use. For example {1} is the unit symbol (e.g. for temperature "Kelvin", "Celsius", "Fahrenheit" etc.) for the measurement. This is possible because all VSCP events carry this information with them. Trying this

```
<script type="text/javascript" >
    var txt1 = new vscpw_simpleTextEvent( "ws://192.168.1.20:7681",
                                            "id1",
                                            VSCP_CLASS1_MEASUREMENT,
                                            VSCP_TYPE_MEASUREMENT_TEMPERATURE,
                                            2,
                                            3,
                                            "{0} degrees {1}");
</script>
```

Gives this result



VSCP HTML5 websocket vscpw_

[Go back to main page](#)

This is just some text in a HTML element where a temperature is equal to 23

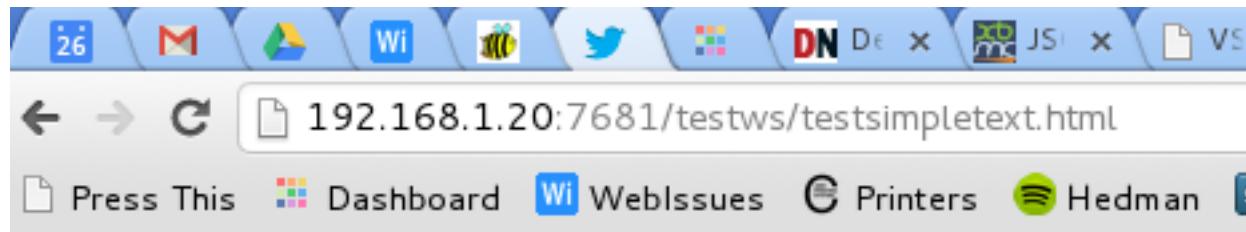
But by inserting

```
<div>
    Temperature placed inside &lt;H1&gt; tags. <h1 id="id2"></h1>
</div>
```

and creating the control

```
<script>
    var txt2 = new vscpw_simpleTextEvent( "ws://192.168.1.20:7681" ,
                                            "id2",
                                            VSCP_CLASS1_MEASUREMENT,
                                            VSCP_TYPE_MEASUREMENT_TEMPERATURE,
                                            0,
                                            3,
                                            "Temperature is {0} degrees {1}");
</script>
```

also works and give this result



VSCP HTML5 websocket vscpw...

[Go back to main page](#)

This is just some text in a HTML element where a temperature is equal to 28.840 degrees Celsius placed inside <H1> tags.

Temperature is 28.840 degrees Celsius

showing dynamic temperature values for sensor 0.

If you instead want to have the measurement value in some other from you can use the callback to do some calculations on the value before it is written to the page. The callback is called before the formatting string is applied. For example if you want the example above to be displayed with the Fahrenheit unit instead of Celcius you can use the built in conversion code and

```
script>
    var txt2 = new vscpw_simpleTextEvent( "ws://192.168.1.20:7681" ,
        "id2",
        VSCP_CLASS1_MEASUREMENT,
        VSCP_TYPE_MEASUREMENT_TEMPERATURE,
        0,
        3,
```

```

        "Temperature is {0} degrees Fahrenheit",
        function(value, unit, eve
    return (" " + vscpws_toFixed(vscpws_convertCelsiusToFahrenheit(value),2))
</script>

```

or in a more readable form

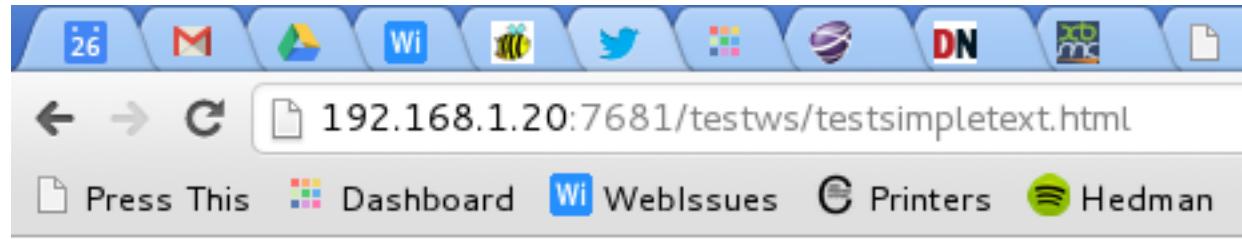
```

<script>
    var fnconvert = function(value, unit, event)
{
    return (" " + vscpws_toFixed(vscpws_convertCelsiusToFahrenheit(value),2))

    var txt2 = new vscpws_simpleTextEvent( "ws://192.168.1.20:7681",
                                            "id2",
                                            VSCP_CLASS1_MEASUREMENT,
                                            VSCP_TYPE_MEASUREMENT_TEMPERATURE,
                                            0,
                                            3,
                                            "Temperature is {0} degrees Fahrenheit",
                                            fnconvert );
}
</script>

```

which outputs



VSCP HTML5 websocket vscpw

[Go back to main page](#)

This is just some text in a HTML element where a temperature is equal to 23 Temperature placed inside <H1> tags.

Temperature is 83.138 degrees F

The widget can be used for other events then measurement event also. To code something that lists the GUID for each new node discovered in the system one can use the following code

```
<div>
    <h1>Unit discovery</h1>
    <div id="id3"></div>
</div>
```

and

```
<script>
    var idArray = new Array();
```

```

var newnode = function(value ,unit ,event) {
    // alert("Heartbeat:" + event);
    var guid = event[3];
    var bFound = false;

    for (i=0;i<idArray.length ;i++) {
        if ( guid == idArray[i]) {
            bFound = true;
            break;
        }
    }

    if (!bFound) {
        var strOut = new String();
        idArray.push(guid);
        for (i=0;i<idArray.length ;i++) {
            strOut += idArray[i] + " - ";
        }
        document.getElementById("id3").textContent = strOut;
    }
    return null;
};

new vscpws_simpleTextEvent( "ws://192.168.1.20:7681" ,
    null ,
    VSCP_CLASS1_INFORMATION,
    VSCP_TYPE_INFORMATION_NODE_HEARTBEAT,
    -1,
    -1,
    "",
    "",
    newnode );
</script>

```

which list the nodes

Temperature is 83 degrees Fahrenheit

Unit discovery

00:01:02:03:04:05:06:07:08:09:0A:0B:00:07:00:00 - 00:01:02:03:04:05:06:07:08:09:0A:0B:00:07:08:09:0A:0B:00:08:09:0A:0B:00:09:0A:0B:00:0A:0B:00:0B:00:0C:00:0D:00:0E:00:0F:00:0G:00:0H:00:0I:00:0J:00:0K:00:0L:00:0M:00:0N:00:0O:00:0P:00:0Q:00:0R:00:0S:00:0T:00:0U:00:0V:00:0W:00:0X:00:0Y:00:0Z:00:0

17.3 vscpws_thermometerCelsius

This information is preliminary and can change at any time.

The vscpws_thermometerCelsius is a widget that can be used to display temperature values in an analog way. It is very easy to use and with just one line of code you can have a dynamic thermometer on your webpage without any programming knowledge.

17.3.1 Function arguments

```
vscpws_thermometerWidgetCelsius( url , // url to VSCP websocket i/f
                                    canvasName , // Placeholder for widget
                                    widgetType , // Widget type
                                    vscpclass , // Event class 10/60/65
                                    vscptype , // Event type
                                    sensorIndex , // Datacoding sensor index
                                    bNumeric , // Add numeric printout
                                    guid ) // GUID we are interested in
```

url This is the url to the websocket server. This typically is on the form

```
"ws://192.168.1.20:7681"
```

With all widgets having their own url specified for the websocket server it is possible to create web pages that control nodes/units/hardware that are located in different locations from the same page.

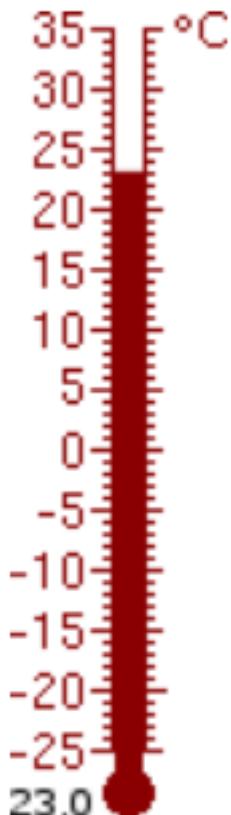
canvasName This is the name of the canvas element where the widget should be placed. Typically this is defined on the form

```
<canvas id="mythermometer1"
        style="z-index: 1;
        position: absolute;
```

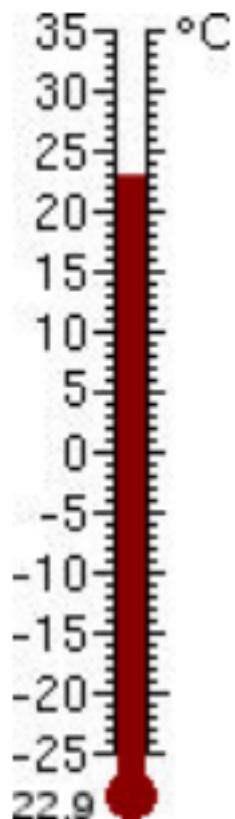
```
left :100px;  
top :200px;">  
Your browser does not support HTML5 Canvas.  
</canvas>
```

The *id* is the parameter that goes for the *canvasName*. This name is also used to create the instance name for the button and the name set is preceded with vscpws_. The canvas specifies the position for the widget, size is set by to the widgets size. Also z-order is possible to define so that objects can be placed behind, or partially behind each other to get nice visual effects.

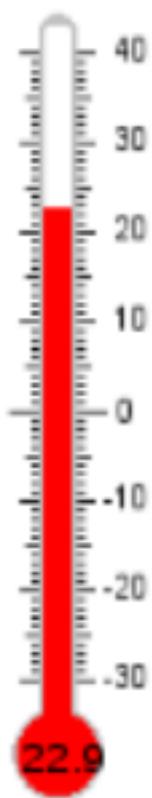
widgetType This is the appearance of the widget. Zero is the default value. The following widgets is defined.



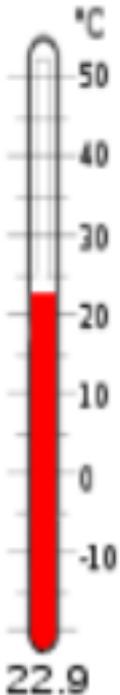
Type = 0 (Default)



Type = 1



Type = 2



Type = 3

vsciclass This is the the VSCP class of the event we are interested in. This parameter should be set to one of the measurement events

VSCP_CLASS1_MEASUREMENT=10 (default),
 VSCP_CLASS1_DATA=15,
 VSCP_CLASS1_MEASUREMENT64=60 or
 VSCP_CLASS1_MEASUREZONE=65.
 Set to -1 for don't care. Default is VSCP_CLASS1_MEASUREMENT.

vscptype This is the the VSCP type of the event we are interested in. This is normally set to VSCP_TYPE_MEASUREMENT_TEMPERATURE=6 , Set to -1 for don't care. Default is VSCP_TYPE_MEASUREMENT_TEMPERATURE=6.

sensorindex This is the sensor index for the sensor in the device sending out the event. This information is part of the datacoding and can be a value 0,1,2,3,4,5,6,7. Set to -1 for don't care. Default=0.

bNumeric Set to true to get a numerical representation of the temperature printed out as well. Default=true.

guid This can be used to match a specific GUID the event should come from. Default is "" meaning events from all sensors are of interest.

17.3.2 Methods

setExtraParameters To use the vscpws_simpleTextEvent widget with events in class VSCP_CLASS1_MEASUREZONE and VSCP_CLASS1_SETVALUEZONE you may want to filter on more than sensorindex and guid. This method allows the extra info available in these classes to be checked as well.

```
setExtraParameters( index,           // Index if applicable
                    zone,          // Zone if applicable
                    subzone)       // Subzone if applicable
```

index This can be used to match a specific index. Default is -1 meaning don't care. Only VSCP_CLASS1_MEASUREZONE and VSCP_CLASS1_SETVALUEZONE have an index to check among the measurement classes.

zone This can be used to match a specific zone. Default is -1 meaning don't care. Only VSCP_CLASS1_MEASUREZONE and VSCP_CLASS1_SETVALUEZONE have a zone to check among the measurement classes.

subzone This can be used to match a specific subzone. Default is -1 meaning don't care. Only VSCP_CLASS1_MEASUREZONE and VSCP_CLASS1_SETVALUEZONE have a subzone to check among the measurement classes.

setMonitorVariable With this method one can set a VSCP daemon variable that should be monitored with a specific interval. The current value of the variable will be written.

```
setMonitorVariable( variablename, // variable name
                     interval)      // monitoring interval in
```

variablename The name for the VSCP daemon boolean variable.

interval The interval in milliseconds between variable reads. Set to zero to disable. Default=1000 (one second). To test try to set the variable name to "temp1" and issue

```
variable write temp1,,,20.5
```

and

```
variable write temp1,,, -12
```

in the VSCP daemon TCP/IP interface to see the change in your browser page.

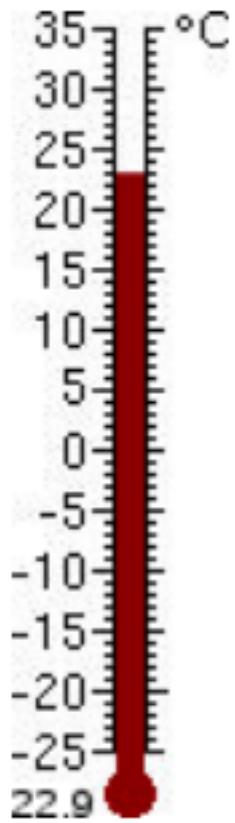
17.3.3 Using the widget

It's very easy to use this widget. Set up a data source that send temperature events on even intervals and then tell the temperature widget to look for them. As always you also need to position your widget somewhere and you do that in the canvas definition. A working example looks like this

```
<canvas id="mythermometer2"
        style="z-index: 1;
               position: absolute;
               left:200px;
               top:200px;">
    Your browser does not support HTML5 Canvas.
</canvas>

var temp2 = new vscpwsthermometerCelsius( "ws://192.168.1.20:7681",
                                             "mythermometer2",
                                             1,
                                             VSCP_CLASS1_MEASUREMENT,
                                             VSCP_TYPE_MEASUREMENT_TEMPERATURE,
                                             2 );
```

which displays measurement temperature data with a widget of type=1 from sensor 2 and the numerics are printed out by default.



No GUID is set here and should be added do distinguish data from different sources.

17.4 vscpws_speedometerCelsius

This information is preliminary and can change at any time.

The vscpws_speedometerCelsius is a widget that can be used to display temperature values in an analog way. It is very easy to use and with just one line of code you can have a dynamic thermometer on your web page without any programming knowledge.

17.4.1 Function arguments

```
vscpws_speedometerCelsius( url ,           // url to VSCP websocket i/f
                           canvasName, // Placeholder for widget
                           widgetType, // Widget type
                           vscpclass , // Event class 10/60/65
```

```
vscptype , // Event type  
sensorIndex ,// Datacoding sensor index  
bNumeric , // Add numeric printout  
guid ) // GUID we are interested in
```

url This is the url to the websocket server. This typically is on the form

```
"ws://192.168.1.20:7681"
```

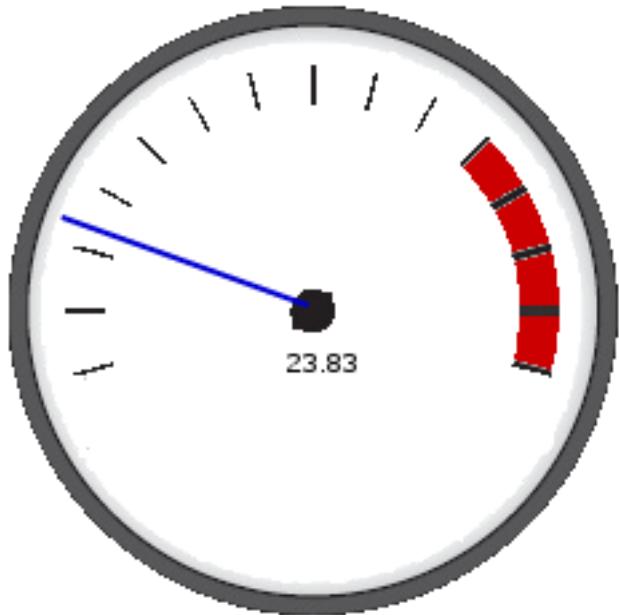
With all widgets having there own url specified for the websocket server it is possible to create web pages that control nodes/units/hardware that are located in different locations from the same page.

canvasName This is the name of the canvas element where the widget should be placed. Typically this is defined on the form

```
<canvas id="mythermometer1"  
style="z-index: 1;  
position: absolute;  
left:100px;  
top:200px;">  
Your browser does not support HTML5 Canvas.  
</canvas>
```

The *id* is the parameter that goes for the *canvasName*. This name is also used to create the instance name for the button and the name set is preceded with vscpws_. The canvas specifies the position for the widget, size is set by to the widgets size. Also z-order is possible to define so that objects can be placed behind, or partially behind each other to get nice visual effects.

widgetType This is the appearance of the widget. Zero i the default value. The following widgets is defined.



Type = 0 (Default)

vsciclass This is the the VSCP class of the event we are interested in. This parameter should be set to one of the measurement events

VSCP_CLASS1_MEASUREMENT=10 (default),
VSCP_CLASS1_DATA=15,
VSCP_CLASS1_MEASUREMENT64=60 or
VSCP_CLASS1_MEASUREZONE=65.
Set to -1 for don't care. Default is VSCP_CLASS1_MEASUREMENT.

vscptype This is the the VSCP type of the event we are interested in. This is normally set to VSCP_TYPE_MEASUREMENT_TEMPERATURE=6 , Set to -1 for don't care. Default is VSCP_TYPE_MEASUREMENT_TEMPERATURE=6.

sensorindex This is the sensor index for the sensor in the device sending out the event. This information is part of the datacoding and can be a value 0,1,2,3,4,5,6,7. Set to -1 for don't care. Default=0.

bNumeric Set to true to get a numerical representation of the temperature printed out as well. Default=true.

guid This can be used to match a specific GUID the event should come from. Default is "" meaning events from all sensors are of interest.

17.4.2 Methods

setExtraParameters To use the vscpws_simpleTextEvent widget with events in class VSCP_CLASS1_MEASUREZONE and VSCP_CLASS1_SETVALUEZONE you may want to filter on more than sensorindex and guid. This method allows the extra info available in these classes to be checked as well.

```
setExtraParameters( index,           // Index if applicable
                    zone,          // Zone if applicable
                    subzone)       // Subzone if applicable
```

index This can be used to match a specific index. Default is -1 meaning don't care. Only VSCP_CLASS1_MEASUREZONE and VSCP_CLASS1_SETVALUEZONE have an index to check among the measurement classes.

zone This can be used to match a specific zone. Default is -1 meaning don't care. Only VSCP_CLASS1_MEASUREZONE and VSCP_CLASS1_SETVALUEZONE have a zone to check among the measurement classes.

subzone This can be used to match a specific subzone. Default is -1 meaning don't care. Only VSCP_CLASS1_MEASUREZONE and VSCP_CLASS1_SETVALUEZONE have a subzone to check among the measurement classes.

setMonitorVariable With this method one can set a VSCP daemon variable that should be monitored with a specific interval. The current value of the variable will be written.

```
setMonitorVariable( variablename, // variable name
                     interval)      // monitoring interval in
```

variablename The name for the VSCP daemon boolean variable.

interval The interval in milliseconds between variable reads. Set to zero to disable. Default=1000 (one second). To test try to set the variable name to "temp1" and issue

```
variable write temp1,,,20.5
```

and

```
variable write temp1,,, -12
```

in the VSCP daemon TCP/IP interface to see the change in your browser page.

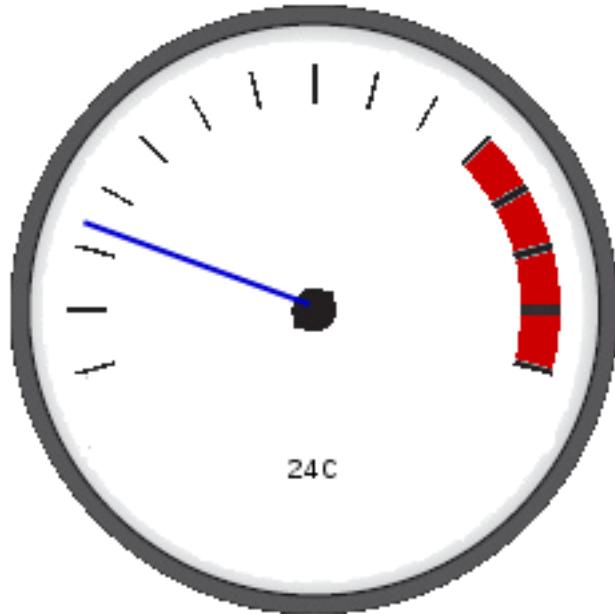
17.4.3 Using the widget

It's very easy to use this widget. Set up a data source that send temperature events on even intervals and then tell the temperature widget to look for them. As always you also need to position your widget somewhere and you do that in the canvas definition. A working example looks like this

```
<canvas id="mythermometer2"
        style="z-index: 1;
               position: absolute;
               left:200px;
               top:200px;">
    Your browser does not support HTML5 Canvas.
</canvas>

var temp2 = new vscpwsspeedometerCelsius( "ws://192.168.1.20:7681",
                                             "mythermometer2",
                                             0,
                                             VSCP_CLASS1_MEASUREMENT,
                                             VSCP_TYPE_MEASUREMENT_TEMPERATURE,
                                             2 );
temp2.setDecimals(0);
```

which displays measurement temperature data with a widget of type=0 from sensor 2 and the numerics are printed out by default. Also no decimals is printed out.



No GUID is set here and should normally be added do distinguish data from different sources.

Part IV

Source code

The source repository is located at github. To get the software source tree use

```
git clone https://github.com/grodansparadis/vscp_software
```

To get the firmware source tree use

```
git clone https://github.com/grodansparadis/vscp_firmware
```

18 How the source tree is organized.

18.1 The software repository tree

The src tree holds PC and higher level code for Windows, PC and Mac.

18.2 The firmware repository tree

The firmware tree hold lower level firmware code for different hardware platforms

Part V

Software api

The VSCP friends package have a lot of software api defined.

- First is the Level I interface (previous called CANAL interface) which is used for VSCP Level I drivers. This is typically driver software for hardware drivers. Events are exchanged with the daemon using Level I. The interface is fully described in the CANAL section VII
- Second is the VSCP Level II driver interface. This interface is used to configure, start and stop drivers that live there life on the standard TCP/IP client interface. A Level II driver talks Level II events and have full access to all functionality of the client interface. Username and password is generated by the daemon when it starts the driver. A Level II driver can fetch its configuration data from daemon variables and therefore also be dynamically configured by another client of the interface.
- The vscphelper.dll/dl(so) is a helper dll/dl(so) that have most of the common functionality needed to work with VSCP on the application level.
- The CanalSuperwrapper class is a wrapper above both the CANAL and the TCP/IP interface. Thus the same code can be used to access both. One can use dllwrapper to just interface CANAL dll/dls or vscptcpipif to work with the tcp/ip interface only.

19 VSCP Level II driver interface.

19.1 long open(char *username, char *password, char *prefix, char *parameters)

Open the VSCP Level II interface and let the driver connect to the local server using the server received username/password pair and log in to it. Parameters are a semicolon separated list with configuration parameters. The prefix is a prefix used before variable names on the form *prefix_variablename* that is used by the client when it get configuration data from the server. If the driver has no need to get configuration data it can be set to NULL.

It is very good if a driver fetch configuration data from the server instead of having its own configuration method such as reading from a file . First configuration data is in one place. Secondly configurations can be changed remotely from the client interface. If the prefix is different for drivers of the same sort several drivers of a kind can be used configured in totally different ways still fetching there configuration data from the same place.

19.2 long close(void)

Close the interface and instructs the driver to log off from the client TCP/IP interface.

19.3 long change (char *variable, char *variablevalue)

A variable that has been changed by the client interface is reported back to the driver in this call.

20 vscphelper.dll/dl(so)

VSCP helper dll with the tools for VSCP program development. This file includes most of the stuff that is needed by a programmer to interface Level I drivers (previously called CANAL drivers) and the talk to a VSCP daemon remotely over tcp/ip. All helper functionality and all communication functionality is included in this dynamically loadable library. Also ready to work with threads for receiving and transmitting events.

The difference of opening a level I driver and talking to the same driver through the TCP/IP interface is that you in the later case can have many programs working against the same interface. When you open a Level I driver you are the only one that can access it.

The API is also described in the wiki here http://www.vscp.org/wiki/doku.php/coding/helper_lib

20.1 Description of the exported API

20.1.1 void vscp_setInterfaceTcp(const char *pHost, const short port, const char *pUsername, const char *pPassword)

Set (not open) interface to talk to tcp/ip interface of the VSCP daemon.

pHost String pointing to hostname of host to connect to. This can be a name that needs to be resolved e.g. “www.grodansparadis.com” or a dotted IP-address e.g. “192.168.1.1”

port Port to use.

pUserName Username to use as credentials for login.

pPassword Password to use as credentials for login.

20.1.2 void vscp_setInterfaceDll(const char *pName, const char *pPath, const char *pParameters, unsigned long flags, unsigned long filter, unsigned long mask)

Set (not open) the Level I interface to talk to. The level I interface was previously called CANAL interface.

pName Points to a string that is the name of the interface. You can set this name to anything you like.

pPath Points to a string that contain the path to the driver to use.

pParameters Points to a string with the parameter you want to use when opening the interface. Check the driver documentation to see what parameters that are available.

flags Flags to use for the interface. Check the driver documentation to see what flags that are available.

filter Here you can set a filter for the driver so that you only get the events that you are interested in.

mask Here you can set a mask for the driver so that you only get the events that you are interested in.

20.1.3 long vscp_doCmdOpen(const char *pInterface, unsigned long flags)

Open the named interface.

pInterface Pointer to a string with the name of interface to open.

flags Flags to use for the interface.

return True if channel is open or false if error or the channel is already opened.

20.1.4 int vscp_doCmdClose(void)

Close the interface.

return true if the close was successful.

20.1.5 int vscp_doCmdNoop(void)

This is a command that can be used for test purposes. It does not do anything but uses the interfaces and return true if everything is OK.

return True if success, false if not.

20.1.6 unsigned long vscp_doCmdGetLevel(void)

Get the driver level. This method has no use on a TCP/IP connection.

return CANAL_LEVELUSES_TCPIP for a Level II driver and CANAL_LEVEL_STANDARD for a Level I driver. Will return CANAL_ERROR_NOT_SUPPORTED if the driver reports some unknown level.

20.1.7 int vscp_doCmdSendCanal(canalMsg *pMsg)

Send a CANAL message on the open channel.

pMsg The CANAL message to send.

Return True if success, false if not.

20.1.8 int vscp_doCmdSendEvent(const vscpEvent *pEvent)

Send a VSCP event on the open channel.

pEvent The level I or level II event to send.

Return True if success, false if not.

20.1.9 int vscp_doCmdSendEventEx(const vscpEventEx *pEvent)

Send a VSCP event on the open channel. The difference to doCmdSendEvent is that the vscpEventEx structure have the data in the structure and vscpEvent that have a pointer to the data in the structure.

pEvent The level I or level II event to send.

Return True if success, false if not.

20.1.10 int vscp_doCmdReceiveCanal(canalMsg *pMsg)

Receive a CANAL message. This method has no use on a TCP/IP connection.

pMsg Pointer to CANAL structure that will receive the message.

Return True if success, false if not.

20.1.11 int vscp_doCmdReceiveEvent(vscpEvent *pEvent)

Receive a VSCP event.

pEvent Pointer to VSCP data structure that will receive the event.

Return True if success, false if not.

20.1.12 int vscp_doCmdReceiveEventEx(vscpEventEx *pEvent)

Receive a VSCP event. The difference to doCmdReceiveEvent is that the vscpEventEx structure have the data in the structure and vscpEvent that have a pointer to the data in the structure.

pEvent Pointer to VSCP data structure that will receive the event.

Return True if success, false if not.

20.1.13 int WINAPI EXPORT vscp_doCmdDataAvailable(void)

Check if an event is available.

Return The number of messages available or if negative an error code.

int WINAPI EXPORT vscp_doCmdStatus(canalStatus *pStatus)

Receive CANAL status.

Return True if success, false if not.

20.1.14 int vscp_doCmdStatistics(canalStatistics *pStatistics)

Get CANAL statistics. This method has no use on a TCP/IP connection.

Return True if success false if not.

20.1.15 int vscp_doCmdFilter(unsigned long filter)

Set a CANAL filter. This method has no use on a TCP/IP connection.

filter Filter to set that together with the mask decides what messages will be received.

Return True if success, false if not.

20.1.16 int vscp_doCmdMask(unsigned long mask)

Set a CANAL mask. This method has no use on a TCP/IP connection.

mask Mask to set that together with the filter decides what messages will be received.

Return True if success, false if not.

20.1.17 int vscp_doCmdVscpFilter(const vscpEventFilter *pFilter)

Set VSCP filter/mask.

pFilter Pointer to a VSCP filter structure that should be applied to the input stream.

Return True if success, false if not.

20.1.18 int vscp_doCmdBaudrate(unsigned long baudrate)

Set the baudrate for a CANAL interface.

baudrate Baudrate to set for the CANAL interface.

Return True if success, false if not.

20.1.19 unsigned long vscp_doCmdVersion(void)

Get version of interface.

Return Version of interface.

20.1.20 unsigned long vscp_doCmdDLLVersion(void)

Get interface driver version

Return Version of interface driver.

20.1.21 const char * vscp_doCmdVendorString(void)

Fetch the vendor string from the driver.

Return A pointer to string with the maker of the driver in use.

20.1.22 const char * vscp_doCmdGetDriverInfo(void)

Get driver information.

Return A pointer to string with XML formated information about the driver in use. This information (if available) can be used to help a user to configure the driver among other things.

20.1.23 int vscp_getDeviceType(void)

Get device type that is active.

Return USE_DLL_INTERFACE if direct DLL interface active. USE_TCPIP_INTERFACE if TCP/IP interface active.

20.1.24 bool vscp_isOpen(void)

Check if communication channel is open.

Return True or false if not.

20.1.25 int vscp_doCmdShutDown(void)

Shut down the daemon.

Return True if success, false if not.

20.1.26 VscpTcpIf * vscp_getTcpIpInterface(void)

Get pointer to TCP/IP interface object.

Return Pointer to the TCP/IP interface object if OK, NULL if failure.

20.1.27 `unsigned long vscp_readStringValue(const char * pStrValue)`

Convert a string to a value. The string can be C/C++ codes (0xff etc).

pStrValue String representation of value.

Return The converted number.

20.1.28 `unsigned char vscp_getVscpPriority(const vscpEvent *pEvent)`

Extract the event priority from a VSCP event.

pEvent VSCP event.

Return Priority.

20.1.29 `void vscp_setVscpPriority(vscpEvent *pEvent, unsigned char priority)`

Set Event priority.

pEvent VSCP event.

priority Priority to set.

20.1.30 `unsigned char vscp_getVSCPheadFromCANid(const unsigned long id)`

Get the VSCP head from a CANAL message id (CAN id).

id Extended 29-bit CAN id to get head from.

Return VSCP head byte.

20.1.31 `unsigned short vscp_getVSCPclassFromCANid(const unsigned long id)`

Get the VSCP class from a CANAL message id (CAN id).

id Extended 29-bit CAN id to get head from.

Return VSCP class.

20.1.32 `unsigned short vscp_getVSCPtypeFromCANid(const unsigned long id)`

Get the VSCP type from a a CANAL message id (CAN id).

id Extended 29-bit CAN id to get head from.

Return VSCP type.

20.1.33 `unsigned short vscp_getVSCPnicknameFromCANid(const unsigned long id)`

Get the VSCP nickname from a a CANAL message id (CAN id).

id Extended 29-bit CAN id to get head from.

Return VSCP nickname.

20.1.34 `unsigned long vscp_getCANidFromVSCPdata(const unsigned char priority, const unsigned short vscp_class, const unsigned short vscp_type)`

Construct a CANAL id (CAN id) from VSCP event.

priority VSCP event priority.

vscp_class VSCP Class.

vscp_type VSCP type.

Return CANAL (CAN) id.

20.1.35 `unsigned long vscp_getCANidFromVSCPevent(const vscpEvent *pEvent)`

Get CANAL id (CAN id) from VSCP event.

pEvent VSCP event.

Return CANAL (CAN) id.

20.1.36 `short vscp_calcCRC(vscpEvent *pEvent, short bSet)`

Calculate VSCP CRC and optionally set it.

pEvent VSCP event.

bSet Set the CRC in event.

Return VSCP CRC.

20.1.37 `bool vscp_getGuidFromString(vscpEvent *pEvent, const char * pGUID)`

Write GUID into VSCP event from string.

pEvent VSCP event.

pGUID Pointer to GUID in string form.

Return True if success, false if not.

20.1.38 `bool vscp_getGuidFromStringToArray(uint8_t *pGUID,
const char * pStr)`

Write GUID from string into array.

pGUID Pointer to VSCP GUID array to fill.

pStr Pointer to GUID in string form.

Return True if success, false if not.

20.1.39 `bool vscp_writeGuidToString(const vscpEvent *pEvent,
char * pStr)`

Write GUID from VSCP event to string.

pEvent VSCP event.

pStr Pointer to GUID in string form.

Return True if success, false if not.

20.1.40 `bool vscp_writeGuidToString4Rows(const vscpEvent *pEvent,
wxString& strGUID)`

Write GUID from VSCP event to string with four bytes on each row separated by “\r\n”.

pEvent VSCP event.

strGUID String that get formatted GUID.

Return True if success, false if not.

20.1.41 `bool vscp_writeGuidArrayToString(const unsigned char *
pGUID, wxString& strGUID)`

Write GUID from byte array to string.

Return True if success, false if not.

20.1.42 `bool vscp_isGUIDEmpty(unsigned char *pGUID)`

Check if GUID is empty (all nulls).

Return True if success, false if not.

20.1.43 `bool vscp_isSameGUID(const unsigned char *pGUID1, const
unsigned char *pGUID2)`

Check if two GUID's is equal to each other.

Return True if equal, false if not.

20.1.44 `bool vscp_convertVSCPtoEx(vscpEventEx *pEventEx, const vscpEvent *pEvent)`

Convert VSCP standard event form to ex. form.

pEvent VSCP event ex to convert to

pEvent VSCP event to convert to.

Return True if success, false if not.

20.1.45 `bool vscp_convertVSCPfromEx(vscpEvent *pEvent, const vscpEventEx *pEventEx)`

Convert VSCP ex. event form to standard form.

pEvent VSCP event to convert to.

pEvent VSCP event ex to convert from.

Return True if success, false if not.

20.1.46 `void vscp_deleteVSCPevent(vscpEvent *pEvent)`

Delete VSCP event.

pEvent VSCP event.

20.1.47 `void vscp_deleteVSCPeventEx(vscpEventEx *pEventEx)`

Delete VSCP event ex.

pEvent VSCP event ex.

20.1.48 `void vscp_clearVSCPFilter(vscpEventFilter *pFilter)`

Clear VSCP filter.

pFilter Pointer to VSCP filter.

20.1.49 `bool vscp_readFilterFromString(vscpEventFilter *pFilter, wxString& strFilter)`

Read a filter from a string

pFilter Filter structure to write filter to.

strFilter Filter in string form filter-priority, filter-class, filter-type, filter-GUID

Return True if success, false if not.

20.1.50 `bool readMaskFromString(vscpEventFilter *pFilter, wxString& strMask)`

Read a mask from a string

pFilter Filter structure to write mask to.

strMask Mask in string form mask-priority, mask-class, mask-type, mask-GUID

Return True if success, false if not.

20.1.51 `bool vscp_doLevel2Filter(const vscpEvent *pEvent, const vscpEventFilter *pFilter)`

Check VSCP filter condition.

Return True if success, false if not.

20.1.52 `bool vscp_convertCanalToEvent(vscpEvent *pvscpEvent, const canalMsg *pcanalMsg, unsigned char *pGUID, bool bCAN)`

Convert CANAL message to VSCP event.

Return True if success, false if not.

20.1.53 `bool vscp_convertEventToCanal(canalMsg *pcanalMsg, const vscpEvent *pvscpEvent)`

Convert VSCP event to CANAL message.

Return True if success, false if not.

20.1.54 `bool vscp_convertEventExToCanal(canalMsg *pcanalMsg, const vscpEventEx *pvscpEventEx)`

Convert VSCP event ex. to CANAL message.

Return True if success, false if not.

20.1.55 `unsigned long vscp_getTimeStamp(void)`

Get VSCP timestamp.

Return Timestamp

20.2 Variable handling

The following methods are for Level II drivers as a way to get configuration data and it's own persistent data.

20.2.1 `bool vscp_copyVSCPEvent(vscpEvent *pEventTo, const vscpEvent *pEventFrom)`

Copy VSCP event.

Return True if success, false if not.

20.2.2 `bool vscp_writeVscpDataToString(const vscpEvent *pEvent, wxString& str, bool bUseHtmlBreak)`

Write VSCP data in readable form to a (multi line) string.

Return True if success, false if not.

20.2.3 `bool vscp_getVscpDataFromString(vscpEvent *pEvent, const wxString& str)`

Set data in VSCP event from a string.

Return True if success, false if not.

20.2.4 `bool vscp_writeVscpEventToString(vscpEvent *pEvent, char *p)`

Write VSCP data to a string.

Return True if success, false if not.

20.2.5 `bool vscp_getVscpEventFromString(vscpEvent *pEvent, const char *p)`

Get VSCP event from string.

Return True if success, false if not.

20.2.6 `bool vscp_getVariableString(const char *pName, char *pValue)`

Get variable value from string variable

pName name of variable

pValue pointer to string that get the value of the string variable.

Return true if the variable is of type string.

20.2.7 `bool vscp_setVariableString(const char *pName, char *pValue)`

Set string variable from a pointer to a string

pName Name of variable

pValue Pointer to string that contains the string.

Return True if the variable is of type string.

20.2.8 `bool vscp_getVariableBool(const char *pName, bool *bValue)`

Get variable value from boolean variable

pName Name of variable

bValue Pointer to boolean variable that get the value of the string variable.

Return True if the variable is of type string.

20.2.9 `bool vscp_setVariableBool(const char *pName, bool bValue)`

Get variable value from boolean variable

pName Name of variable

bValue Pointer to boolean variable that get the value of the string variable.

Return true if the variable is of type string.

20.2.10 `bool vscp_getVariableInt(const char *pName, int *value)`

Get variable value from integer variable

pName Name of variable

value Pointer to integer variable that get the value of the string variable.

Return true if the variable is of type string.

20.2.11 `bool vscp_setVariableInt(const char *pName, int value)`

Get variable value from integer variable

pName Name of variable

value Pointer to integer variable that get the value of the string variable.

Return true if the variable is of type string.

20.2.12 `bool vscp_getVariableLong(const char *pName, long *value)`

Get variable value from long variable

pName Name of variable

value Pointer to long variable that get the value of the string variable.

Return true if the variable is of type string.

20.2.13 `bool vscp_setVariableLong(const char *pName, long value)`

Get variable value from long variable

pName Name of variable

value Pointer to long variable that get the value of the string variable.

Return true if the variable is of type string.

20.2.14 `bool vscp_getVariableDouble(const char *pName, double *value)`

Get variable value from double variable

pName Name of variable

value Pointer to double variable that get the value of the string variable.

Return true if the variable is of type string.

20.2.15 `bool vscp_setVariableDouble(const char *pName, double value)`

Get variable value from double variable

pName Name of variable

value Pointer to double variable that get the value of the string variable.

Return true if the variable is of type string.

20.2.16 `bool vscp_getVariableMeasurement(const char *pName, char *pValue)`

Get variable value from measurement variable

pName Name of variable

pValue String that get that value of the measurement.

Return true if the variable is of type string.

20.2.17 `bool vscp_setVariableMeasurement(const char *pName,
char *pValue)`

Get variable value from measurement variable

pName Name of variable

pValue String that get that get the value of the measurement.

Return true if the variable is of type string.

20.2.18 `bool vscp_getVariableEvent(const char *pName, vscpEvent
*pEvent)`

Get variable value from event variable

pName Name of variable

pEvent Pointer to event variable that get the value of the string variable.

Return true if the variable is of type string.

20.2.19 `bool vscp_setVariableEvent(const char *pName, vscpEvent
*pEvent)`

Get variable value from event variable

pName Name of variable

pEvent Pointer to event variable that get the value of the string variable.

Return true if the variable is of type string.

20.2.20 `bool vscp_getVariableEventEx(const char *pName, vscpEven-
tEx *pEvent)`

Get variable value from event variable

pName Name of variable

pEvent Pointer to event variable that get the value of the string variable.

Return true if the variable is of type string.

20.2.21 `bool vscp_setVariableEventEx(const char *pName, vscpEven-
tEx *pEvent)`

Get variable value from event variable

pName Name of variable

pEvent Pointer to event variable that get the value of the string variable.

Return true if the variable is of type string.

20.2.22 `bool vscp_getVariableGUID(const char *pName, cguid& GUID)`

Get variable value from GUID variable

pName Name of variable

GUID Pointer to event variable that get the value of the GUID variable.

Return true if the variable is of type string.

20.2.23 `bool vscp_setVariableGUID(const char *pName, cguid& GUID)`

Get variable value from GUID variable

pName Name of variable

GUID Pointer to event variable that get the value of the GUID variable.

Return true if the variable is of type string.

20.2.24 `bool vscp_getVariableVSCPdata(const char *pName, uint16_t *psizeData, uint8_t *pData)`

Get variable value from VSCP data variable

pName Name of variable

psizeData Pointer to variable that will hold the size of the data array

pData Pointer to VSCP data array variable (unsigned char [8]) that get the value of the string variable.

Return true if the variable is of type string.

20.2.25 `bool vscp_setVariableVSCPdata(const char *pName, uint16_t sizeData, uint8_t *pData)`

Get variable value from VSCP data variable

pName Name of variable

sizeData Pointer to variable that will hold the size of the data array

pData Pointer to VSCP data array variable (unsigned char [8]) that get the value of the string variable.

Return true if the variable is of type string.

20.2.26 `bool vscp_getVariableVSCPclass(const char *pName, uint16_t *vscp_class)`

Get variable value from class variable

pName Name of variable

vscp_class Pointer to int that get the value of the class variable.

Return true if the variable is of type string.

20.2.27 `bool vscp_setVariableVSCPclass(const char *pName, uint16_t vscp_class)`

Get variable value from class variable

pName Name of variable

vscp_class Pointer to int that get the value of the class variable.

Return true if the variable is of type string.

20.2.28 `bool vscp_getVariableVSCPtype(const char *pName, uint8_t *vscp_type)`

Get variable value from type variable

pName Name of variable

vscp_type Pointer to int that get the value of the type variable.

Return true if the variable is of type string.

20.2.29 `bool vscp_setVariableVSCPtype(const char *pName, uint8_t vscp_type)`

Get variable value from type variable

pName Name of variable

vscp_type Pointer to int that get the value of the type variable.

Return true if the variable is of type string.

20.3 Examples

Currently the best example code is the source code for VSCP Works.

21 Canalsuperwrapper

```
//////////  
// CanalSuperWrapper.h: interface for the CCanalSuperWrapper class  
//  
// This file is part is part of CANAL (CAN Abstraction Layer)  
// http://www.vscp.org)  
//  
// Copyright (C) 2000-2012 Ake Hedman, Grodans Paradis AB, <akhe@grodansparadis.com>  
//  
// This library is free software; you can redistribute it and/or  
// modify it under the terms of the GNU Lesser General Public  
// License as published by the Free Software Foundation; either  
// version 2.1 of the License, or (at your option) any later version.  
//  
// This library is distributed in the hope that it will be useful,  
// but WITHOUT ANY WARRANTY; without even the implied warranty  
of  
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  
See the GNU  
// Lesser General Public License for more details.  
//  
// You should have received a copy of the GNU Lesser General Public  
// License along with this library; if not, write to the Free Software  
// Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307  
USA  
//  
// $RCSfile: canalsuperwrapper.h,v $  
// $Date: 2005/08/30 11:00:04 $  
// $Author: akhe $  
// $Revision: 1.4 $  
//////////  
/*!  
\file canalsuperwrapper.h  
\brief Wrapping class for VSCP interfaces.  
\details This class encapsulates both the CANAL dll and the VSCP TCP/IP  
interface and  
makes it possible to use the same code to read/write both CANAL messages  
and VSCP  
events.  
\author Ake Hedman <akhe@grodansparadis.com>, Grodans Paradis AB,  
Sweden  
*/  
#include <string.h>  
#include "devitem.h"  
#include "dllwrapper.h"
```

```

#include "vscptcpif.h"
#include "guid.h"
#include "../common/dllist.h"
#ifndef AFX_CANALSUPERWRAPPER_H__A908F21A_317D_4E74_9308_18D7DD6B7D49_
#define AFX_CANALSUPERWRAPPER_H__A908F21A_317D_4E74_9308_18D7DD6B7D49__INC
#include "canal.h"
#include "devitem.h"
/// Build swig code if defined
#define BUILD_SWIG // Must be here for SWIG builds, Can be commented out for C/C++
/*!{@
Constants for possible interfaces
@}*/
/// Constant that defines the CANAL interface
#define USE_DLL_INTERFACE 0
/// Con-stat that defines the TCP/IP interface
#define USE_TCPIP_INTERFACE 1
/// Maximum path length (if not defined by the OS if any.
#ifndef MAX_PATH
#define MAX_PATH 255
#endif
/*!
\class CCanalSuperWrapper
\brief Encapsulates the CANAL interface and the VSCP tcp/ip interface
*/
class CCanalSuperWrapper
{
public:
/*!
Default Constructor
Use one of the SetInterface methods to specify
device/host data.
*/
CCanalSuperWrapper( void );
/*!
Set Interface CANAL
Use this method if the default constructor is used to construct the object
pItem->name is the user name for the driver.
pItem->path is the location of the driver on the disc. Path can be set to
"TCPIP"
in which case the config string holds connect information.
pItem->config is the configuration string for the driver. If path = "TCPIP"
the
this string is searched for host connect information on the form
"username;password;host;port"
pItem->flags is the configuration flags for the driver. Not used for TCP/IP

```

```

interface.
*/
CCanalSuperWrapper( devItem *pItem );
/// Destructor
virtual ~CCanalSuperWrapper();
/*!
Set Interface TCP/IP
Use this method if the default constructor is used to construct the object
@param host to connect to (defaults to localhost if not given).
@param port on host to connect to. Default to 9598 if not given.
@param username Username to login to service.
@param password to login to service.
*/
void setInterface( const wxString& host,
const short port,
const wxString& username,
const wxString& password );
/*!
Set Interface CANAL
Use this method if the default constructor is used to construct the object
@param name is the user name for the driver.
@param path is the location of the driver on the disc.
@param parameters is the configuration string for the driver.
@param flags is the configuration flags for the driver.
@param filter Filer to set.
@param mask Mask to set.
*/
void setInterface( const wxString& name,
const wxString& path,
const wxString& parameters,
const unsigned long flags,
const unsigned long filter,
const unsigned long mask );
/*!
Open communication channel.
Data can be supplied to override the pdev structure information
that was given on construction.
@param strInterface is name of channel.
@param flags Interface flags
@return true if channel is open or false if error or the channel is
already opened.
*/
long doCmdOpen( const wxString& strInterface = (_T("")), unsigned long
flags = 0L );
/*!
Close communication channel

```

```

@return true if the close was successful
*/
int doCmdClose( void );
/*!
Do a no operation test command.
@return true if success false if not.
*/
int doCmdNOOP( void );
/*!
Get the CANAL protocol level
@return true on success
*/
unsigned long doCmdGetLevel( void );
/*!
Send a CANAL message.
@return true if success false if not.
*/
int doCmdSend( canalMsg *pMsg );
/*!
Send a VSCP event.
@return true if success false if not.
*/
int doCmdSend( const vscpEvent *pEvent );
/*!
Send a VSCP Ex event.
@return true if success false if not.
*/
int doCmdSend( const vscpEventEx *pEventEx );
/*!
Receive a CANAL message.
@return true if success false if not.
*/
int doCmdReceive( canalMsg *pMsg );
/*!
Receive a VSCP event.
@return true if success false if not.
*/
int doCmdReceive( vscpEvent *pEvent );
/*!
Receive a VSCP Ex event.
@return true if success false if not.
*/
int doCmdReceive( vscpEventEx *pEventEx );
/*!
Get the number of messages in the input queue
@return the number of messages available or if negative

```

```

an error code.
*/
int doCmdDataAvailable( void );
/*!
Receive CANAL status.
@return true if success false if not.
*/
int doCmdStatus( canalStatus *pStatus );
/*!
Receive CANAL statistics through the pipe.
@return true if success false if not.
*/
int doCmdStatistics( canalStatistics *pStatistics );
/*!
Set/Reset a filter through the pipe.
@return true if success false if not.
*/
int doCmdFilter( unsigned long filter );
/*!
Set/Reset a mask through the pipe.
@return true if success false if not.
*/
int doCmdMask( unsigned long mask );
/*!
Set/Reset the VSCP filter for a channel.
@return true if success false if not.
*/
int doCmdVscpFilter( const vscpEventFilter *pFilter );
/*!
Set baudrate.
@return true if success false if not.
*/
int doCmdBaudrate( unsigned long baudrate );
/*!
Get i/f version.
@return true if success false if not.
*/
unsigned long doCmdVersion( void );
/*!
Get dll version.
@return true if success false if not.
*/
unsigned long doCmdDLLVersion( void );
/*!
Get vendorstring.
@return Pointer to vendor string.

```

```

*/
const char *doCmdVendorString( void );
/*!
Get device information string.
@return pointer to driver information string.
*/
const char *doCmdGetDriverInfo( void );
/*!
Get the type of interface that is active.
@return USE_DLL_INTERFACE if direct DLL interface active.
@return USE_TCPIP_INTERFACE if TCP/IP interface active.
*/
int getDeviceType( void ) { return m_itemDevice.id; }
/*!
Check if interface is open
@return Return true if open, false if closed.
*/
bool isOpen( void ) { return ( m_devid ? true : false ); }
/*!
Shutdown the VSCP daemon
@return CANAL_ERROR_SUCCESS if success CANAL_ERROR_GENERIC
on failure.
*/
int doCmdShutDown( void );
/*!
Get a pointer to the TCP/IP interface
@return A pointer to the TCP/IP interface
*/
VscpTcpIf *getTcpIpInterface( void ) { return &m_vscptcpif; }
/*!
Get variable value from string variable
\param name of variable
\param strValue pointer to string that get the value of the string variable.
\return true if the variable is of type string.
*/
bool getVariableString( wxString& name, wxString *strValue )
{ return m_vscptcpif.getVariableString( name, strValue ); }
/*!
Set variable value from string variable
\param name of variable
\param strValue to string that get the value of the string variable.
\return true if the variable is of type string.
*/
bool setVariableString( wxString& name, wxString& strValue )
{ return m_vscptcpif.setVariableString( name, strValue ); }
*/

```

```

Get variable value from boolean variable
\param name of variable
\param bValue pointer to boolean variable that get the value of the string
variable.
\return true if the variable is of type string.
*/
bool getVariableBool( wxString& name, bool *bValue )
{ return m_vscptcpif.getVariableBool( name, bValue ); };
/*!
Set variable value from boolean variable
\param name of variable
\param bValue boolean variable that get the value of the string variable.
\return true if the variable is of type string.
*/
bool setVariableBool( wxString& name, bool bValue )
{ return m_vscptcpif.setVariableBool( name, bValue ); };
/*!
Get variable value from integer variable
\param name of variable
\param value pointer to integer variable that get the value of the string
variable.
\return true if the variable is of type string.
*/
bool getVariableInt( wxString& name, int *value )
{ return m_vscptcpif.getVariableInt( name, value ); };
/*!
set variable value from integer variable
\param name of variable
\param value integer variable that get the value of the string variable.
\return true if the variable is of type string.
*/
bool setVariableInt( wxString& name, int value )
{ return m_vscptcpif.setVariableInt( name, value ); };
/*!
Get variable value from long variable
\param name of variable
\param value pointer to long variable that get the value of the string variable.
\return true if the variable is of type string.
*/
bool getVariableLong( wxString& name, long *value )
{ return m_vscptcpif.getVariableLong( name, value ); };
/*!
Set variable value from long variable
\param name of variable
\param value long variable that get the value of the string variable.
\return true if the variable is of type string.
*/

```

```

*/
bool setVariableLong( wxString& name, long value )
{ return m_vscptcpif.setVariableLong( name, value ); };
*/
Get variable value from double variable
\param name of variable
\param value pointer to double variable that get the value of the string
variable.
\return true if the variable is of type string.
*/
bool getVariableDouble( wxString& name, double *value )
{ return m_vscptcpif.getVariableDouble( name, value ); };
*/
Set variable value from double variable
\param name of variable
\param value pointer to double variable that get the value of the string
variable.
\return true if the variable is of type string.
*/
bool setVariableDouble( wxString& name, double value )
{ return m_vscptcpif.setVariableDouble( name, value ); };
*/
Get variable value from measurement variable
\param name of variable
\param strValue String that get that get the
value of the measurement.
\return true if the variable is of type string.
*/
bool getVariableMeasurement( wxString& name, wxString& strValue )
{ return m_vscptcpif.getVariableMeasurement( name, strValue ); };
*/
Set variable value from measurement variable
\param name of variable
\param strValue String that get that get the
value of the measurement.
\return true if the variable is of type string.
*/
bool setVariableMeasurement( wxString& name, wxString& strValue )
{ return m_vscptcpif.setVariableMeasurement( name, strValue ); };
*/
Get variable value from event variable
\param name of variable
\param pEvent pointer to event variable that get the value of the string
variable.
\return true if the variable is of type string.
*/

```

```

bool getVariableEvent( wxString& name, vscpEvent *pEvent )
{ return m_vscptcpif.getVariableEvent( name, pEvent ); };
<*/
Set variable value from event variable
\param name of variable
\param pEvent pointer to event variable that get the value of the string
variable.
\return true if the variable is of type string.
*/
bool setVariableEvent( wxString& name, vscpEvent *pEvent )
{ return m_vscptcpif.setVariableEvent( name, pEvent ); };
<*/
Get variable value from event variable
\param name of variable
\param pEvent pointer to event variable that get the value of the string
variable.
\return true if the variable is of type string.
*/
bool getVariableEventEx( wxString& name, vscpEventEx *pEvent )
{ return m_vscptcpif.getVariableEventEx( name, pEvent ); };
<*/
Set variable value from event variable
\param name of variable
\param pEvent pointer to event variable that get the value of the string
variable.
\return true if the variable is of type string.
*/
bool setVariableEventEx( wxString& name, vscpEventEx *pEvent )
{ return m_vscptcpif.setVariableEventEx( name, pEvent ); };
<*/
Get variable value from GUID variable
\param name of variable
\param GUID variable that get the value of the GUID variable.
\return true if the variable is of type string.
*/
bool getVariableGUID( wxString& name, cguid& GUID )
{ return m_vscptcpif.getVariableGUID( name, GUID ); };
<*/
Set variable value from GUID variable
\param name of variable
\param GUID variable that get the value of the GUID variable.
\return true if the variable is of type string.
*/
bool setVariableGUID( wxString& name, cguid& GUID )
{ return m_vscptcpif.setVariableGUID( name, GUID ); };
<*/

```

```

Get variable value from VSCP data variable
\param name of variable
\param psizeData Pointer to variable that will hold the size of the data array
\param pData pointer to VSCP data array variable (unsigned char [8] ) that
get the
    value of the string variable.
    \return true if the variable is of type string.
*/
bool getVariableVSCPdata( wxString& name, uint16_t *psizeData, uint8_t
*pData )
{
    { return m_vscptcpif.getVariableVSCPdata( name, psizeData, pData ); };
/*!
Set variable value from VSCP data variable
\param name of variable
\param sizeData to variable that will hold the size of the data array
\param pData pointer to VSCP data array variable (unsigned char [8] ) that
get the
    value of the string variable.
    \return true if the variable is of type string.
*/
bool setVariableVSCPdata( wxString& name, uint16_t sizeData, uint8_t
*pData )
{
    { return m_vscptcpif.setVariableVSCPdata( name, sizeData, pData ); };
/*!
Get variable value from class variable
\param name of variable
\param vscp_class pointer to int that get the value of the class variable.
\return true if the variable is of type string.
*/
bool getVariableVSCPclass( wxString& name, uint16_t *vscp_class )
{
    { return m_vscptcpif.getVariableVSCPclass( name, vscp_class ); };
/*!
Set variable value from class variable
\param name of variable
\param vscp_class int that get the value of the class variable.
\return true if the variable is of type string.
*/
bool setVariableVSCPclass( wxString& name, uint16_t vscp_class )
{
    { return m_vscptcpif.setVariableVSCPclass( name, vscp_class ); };
/*!
Get variable value from type variable
\param name of variable
\param vscp_type pointer to int that get the value of the type variable.
\return true if the variable is of type string.
*/
bool getVariableVSCPtype( wxString& name, uint8_t *vscp_type )

```

```

{ return m_vscptcpif.getVariableVSCPtype( name, vscp_type ); }
/*!
Set variable value from type variable
\param name of variable
\param vscp_type to int that get the value of the type variable.
\return true if the variable is of type string.
*/
bool setVariableVSCPtype( wxString& name, uint8_t vscp_type )
{ return m_vscptcpif.setVariableVSCPtype( name, vscp_type ); }

protected:
/*!
ID for open device
*/
long m_devid;
/*!
Timer for canal interaction
*/
unsigned int m_uTimerID;
/*!
Data for the selected device
id holds type of i/f (DLL, TCP/IP
*/
devItem m_itemDevice;
/*!
CANAL TCP/IP interface
*/
VscpTcpIf m_vscptcpif;
/*!
GUID
*/
cguid m_GUID;
/*!
The CANAL dll wrapper
*/
CDllWrapper m_canalDll;
/*!
Flag for open device
*/
bool m_bOpen;
};

#endif // !defined(AFX_CANALSUPERWRAPPER_H__A908F21A_317D_4E74_9308_18D7DD6B7I

```

22 dllwrapper

||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||

```

// dllwrapper.h: interface for the CDllWrapper class
//
// This file is part is part of CANAL (CAN Abstraction Layer)
// http://www.vscp.org)
//
// Copyright (C) 2000-2012 Ake Hedman, Grodans Paradis AB, <akhe@grodansparadis.com>
//
// This library is free software; you can redistribute it and/or
// modify it under the terms of the GNU Lesser General Public
// License as published by the Free Software Foundation; either
// version 2.1 of the License, or (at your option) any later version.
//
// This library is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty
of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
See the GNU
// Lesser General Public License for more details.
//
// You should have received a copy of the GNU Lesser General Public
// License along with this library; if not, write to the Free Software
// Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307
USA.
//
// RCSfile: dllwrapper.h,v $
// $Date: 2005/08/30 11:00:04 $
// $Author: akhe $
// $Revision: 1.10 $
///////////////////////////////$INCLUDE
#ifndef AFX_DLLWRAPPER_H_66E4FA3F_1CA1_405D_AAF1_5F9B1272D75A__INCLUDED_
#define AFX_DLLWRAPPER_H_66E4FA3F_1CA1_405D_AAF1_5F9B1272D75A__INCLUDED_
#include "wx/wx.h"
#include <wx/dynlib.h>
#include "canaldef.h" // Holds the function declarations
/*
\file dllwrapper.h
\brief This class encapsulates the CANAL dll/dl(so) interface.
\details The class have all
methods of the CANAL interface specification implemented in an easy to
handle form.

Use the class if you want to talk directly to CANAL dll/dl(so) drivers. A
better
choice may be to use CanalSuperWrapper which can talk to both a dll/dl(so)
and to a tvp/ip
interface.
*/

```

```

/*!
\class CDllWrapper
\brief Encapsulates the CANAL dll/dl(so) interface.
\details The class have all
methods of the CANAL interface specification implemented in an easy to
handle form.

Use the class if you want to talk directly to CANAL dll/dl(so) drivers. A
better
choice may be to use CanalSuperWrapper which can talk to both a dll/dl(so)
and to a tvp/ip
interface.

*/
class CDllWrapper
{
public:
/// Constructor
CDllWrapper();
/// Destructor
virtual ~CDllWrapper();
/*!
initialize the dll wrapper
@param strPath to the canal dll
@return true on success
*/
int initialize( wxString& strPath );
/*!
Open communication channel.
@param strConfiguration is name of channel.
@param flags CANAL flags for the channel.
@return Channel handler on success. -1 on error.
*/
long doCmdOpen( const wxString& strConfiguration = (_("")), unsigned
long flags = 0L );
/*!
Close communication channel
@return CANAL_ERROR_SUCCESS on success or CANAL error if failure.
*/
int doCmdClose( void );
/*!
Do a no operation test command.
@return CANAL_ERROR_SUCCESS on success or CANAL error if failure.
*/
bool doCmdNOOP( void ) { return true; };
/*!
Get the Canal protocol level
@return CANAL_ERROR_SUCCESS on success or CANAL error if failure.

```

```

*/
unsigned long doCmdGetLevel( void );
/*!
Send a CAN message
@param pMsg Pointer to CAN message to send
@return CANAL_ERROR_SUCCESS on success or CANAL error if failure.
*/
int doCmdSend( canalMsg *pMsg );
/*!
Send a CAN message and block if it can't
be sent right away.
@param pMsg Pointer to CAN message to send
@param timeout Time to wait in milliseconds or zero to wait forever.
@return CANAL_ERROR_SUCCESS on success or CANAL error if failure.
Return CANAL_ERROR_NOT_SUPPORTED if blocking operations is
not
supported.
*/
int doCmdBlockingSend( canalMsg *pMsg, unsigned long timeout );
/*!
Receive a CAN message.
@param pMsg Pointer to CAN message that receive received message.
@return CANAL_ERROR_SUCCESS on success or CANAL error if failure.
*/
int doCmdReceive( canalMsg *pMsg );
/*!
Receive a CAN message
@param pMsg Pointer to CAN message that receive received message.
@param timeout Time to wait in milliseconds or zero to wait forever.
@return CANAL_ERROR_SUCCESS on success or CANAL error if failure.
Return CANAL_ERROR_NOT_SUPPORTED if blocking operations is
not
supported.
*/
int doCmdBlockingReceive( canalMsg *pMsg, unsigned long timeout );
/*!
Get the number of messages in the input queue
@return the number of messages available or if negative
an error code.
*/
int doCmdDataAvailable( void );
/*!
Receive CAN status.
@return Return number of messages in in queue.
*/
int doCmdStatus( canalStatus *pStatus );

```

```

/*!
Receive CAN statistics.
@return CANAL_ERROR_SUCCESS on success or CANAL error if failure.
*/
int doCmdStatistics( canalStatistics *pStatistics );
/*!!
Set/Reset a filter.
@return CANAL_ERROR_SUCCESS on success or CANAL error if failure.
*/
int doCmdFilter( const unsigned long filter );
/*!!
Set/Reset a mask.
@return CANAL_ERROR_SUCCESS on success or CANAL error if failure.
*/
int doCmdMask( const unsigned long mask );
/*!!
Set baudrate.
@return CANAL_ERROR_SUCCESS on success or CANAL error if failure.
*/
int doCmdBaudrate( const unsigned long baudrate );
/*!!
Get interface version.
@return Driver version.
*/
unsigned long doCmdVersion( void );
/*!!
Get dll version.
@return DLL version.
*/
unsigned long doCmdDLLVersion( void );
/*!!
Get vendorstring.
@return Pointer to vendor string..
*/
const char *doCmdVendorString( void );
/*!!
Get driver information
@return Pointer to driver information string or NULL
if no driver info is available.
*/
const char *doCmdGetDriverInfo( void );
// Extended functionality
/*!!
Check if blocking is supported
@param bRead Set to true to check receive blocking
@param bWrite Set to true to check send blocking

```

```

@return Returns true if blocking is supported.
*/
bool isBlockingSupported( bool bRead = true, bool bWrite = true );
protected:
/*!
Path to CANAL driver
*/
wxString m_strPath;
/// dl/dll handler
wxDynamicLibrary m_wxdll;
/// device id from open call
long m_devid;
/*!
Flag that indicates that a successful initialization
has been performed.
*/
bool m_bInit;
//{@{
/// CANAL methods
LPFNDLL_CANALOPEN m_proc_CanalOpen;
LPFNDLL_CANALCLOSE m_proc_CanalClose;
LPFNDLL_CANALGETLEVEL m_proc_CanalGetLevel;
LPFNDLL_CANALSEND m_proc_CanalSend;
LPFNDLL_CANALRECEIVE m_proc_CanalReceive;
LPFNDLL_CANALDATAAVAILABLE m_proc_CanalDataAvailable;
LPFNDLL_CANALGETSTATUS m_proc_CanalGetStatus;
LPFNDLL_CANALGETSTATISTICS m_proc_CanalGetStatistics;
LPFNDLL_CANALSETFILTER m_proc_CanalSetFilter;
LPFNDLL_CANALSETMASK m_proc_CanalSetMask;
LPFNDLL_CANALSETBAUDRATE m_proc_CanalSetBaudrate;
LPFNDLL_CANALGETVERSION m_proc_CanalGetVersion;
LPFNDLL_CANALGETDLLVERSION m_proc_CanalGetDllVersion;
LPFNDLL_CANALGETVENDORSTRING m_proc_CanalGetVendorString;
// Generation 2
LPFNDLL_CANALBLOCKINGSEND m_proc_CanalBlockingSend;
LPFNDLL_CANALBLOCKINGRECEIVE m_proc_CanalBlockingReceive;
LPFNDLL_CANALGETDRIVERINFO m_proc_CanalGetdriverInfo;
//@}
};

#endif // !defined(AFX_DLLWRAPPER_H__66E4FA3F_1CA1_405D_AAF1_5F9B1272D75A__INC

```

23 vscptcpipif

```
///////////////////////////////
// vscptcpif.h:
```

```

// This file is part of CANAL (CAN Abstraction Layer)
// http://www.vscp.org)
//
// Copyright (C) 2000-2012 Ake Hedman, Grodans Paradis AB, <akhe@grodansparadis.com>
//
// This library is free software; you can redistribute it and/or
// modify it under the terms of the GNU Lesser General Public
// License as published by the Free Software Foundation; either
// version 2.1 of the License, or (at your option) any later version.
//
// This library is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty
of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
See the GNU
// Lesser General Public License for more details.
//
// You should have received a copy of the GNU Lesser General Public
// License along with this library; if not, write to the Free Software
// Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307
USA
//
//!/*****\*!\file vscptcpif.h
\brief The VscpTcpIf class encapsulates the VSCP tcp/ip interface.
\details The VscpTcpIf class holds functionality to talk to the tcp/ip
of the VSCP daemon and on client that implement the interface. All api's
of the interface are supported by the class.
*/
#ifndef !defined(AFX_VSCPTCPIF_H__C2A773AD_8886_40F0_96C4_4DCA663402B2__INCLUDED_)
#define AFX_VSCPTCPIF_H__C2A773AD_8886_40F0_96C4_4DCA663402B2__INCLUDED_
#include "canal.h"
#include "vscp.h"
#include "guid.h"
#include "vscphelper.h"
#include "wx/socket.h"
#include "wx/datetime.h"
//-----
/*!
\def DEFAULT_RESPONSE_TIMEOUT
Default response timeout for communication with the
tcp/ip interface
*/
#define DEFAULT_RESPONSE_TIMEOUT 2

```

```

/*!
\def TCPIP_DLL_VERSION
Pseudo version string
*/
#define TCPIP_DLL_VERSION 0x00000001
/*!
\def TCPIP_VENDOR_STRING
Pseudo vendor string
*/
#define TCPIP_VENDOR_STRING (_("Grodans Paradis AB, Sweden"))
/// Receive queue
WX_DECLARE_LIST( vscpEvent, EVENT_RX_QUEUE );
/// Transmit queue
WX_DECLARE_LIST( vscpEvent, EVENT_TX_QUEUE );
/*!
@brief Class for VSCP daemon tcp/ip interface
*/
class VscpTcpIf
{
public:
/// Constructor
VscpTcpIf();
/// Destructor
virtual ~VscpTcpIf();
public:
/*!
checkReturnValue
\return Return false for "-OK" and true for "+OK"
*/
bool checkReturnValue( void );
/*!
Do command
*/
bool doCommand( uint8_t cmd );
/*!
Open communication interface.
\param strInterface should contain "username;password;ip-addr;port" if used.
All including
port are optional and defaults to no username/password, server as "localhost" and "9598"
as default port.
\param flags are not used at the moment.
\return CANAL_ERROR_SUCCESS if channel is open or CANAL error
code if error
or the channel is already opened or other error occur.
*/

```

```

long doCmdOpen( const wxString& strInterface = (_("")), uint32_t flags
= 0L );
/*!
Open communication interface.
\param strHostname Host to connect to.
\param port TCP/IP port to use.
\param strUsername Username.
\param strPassword Username.
\return CANAL_ERROR_SUCCESS if channel is open or CANAL error
code if error
or the channel is already opened or other error occur.
*/
long doCmdOpen( const wxString& strHostname,
const short port,
const wxString& strUsername,
const wxString& strPassword );
/*!
Close communication interface
\return CANAL_ERROR_SUCCESS on success and error code if failure.
*/
int doCmdClose( void );
/*!
Write NOOP message through pipe.
\return CANAL_ERROR_SUCCESS on success and error code if failure.
*/
int doCmdNOOP( void );
/*!
Get the Canal protocol level
\return VSCP Level
*/
unsigned long doCmdGetLevel( void ) { return CANAL_LEVEL_STANDARD;
}
/*!
Send a VSCP Level II event through interface.
If a GUID sent with all items set to zero the GUID of the
interface ("") will be used.
\param pEvent VSCP Level II event to send.
\return CANAL_ERROR_SUCCESS on success and error code if failure.
*/
int doCmdSend( const vscpEvent *pEvent );
/*!
Send a VSCP ex event through interface.
\param pEvent VSCP Level II ex event to send.
\return CANAL_ERROR_SUCCESS on success and error code if failure.
*/
int doCmdSendEx( const vscpEventEx *pEvent );

```

```

/*!
Send a Level I event through interface.
\param pMsg VSCP Level I event (==canalMsg).
\return CANAL_ERROR_SUCCESS on success and error code if failure.
*/
int doCmdSendLevel1( const canalMsg *pMsg );
/*!
Receive a VSCP event through the interface.
\return CANAL_ERROR_SUCCESS on success and error code if failure.
*/
int doCmdReceive( vscpEvent *pEvent );
/*!
Receive an VSCP ex event through the interface.
\return true if success false if not.
*/
int doCmdReceiveEx( vscpEventEx *pEvent );
/*!
Receive an VSCP Level I event through the interface.
For the extended and the RTR bit to be handled the
CAN message bit in the head byte must be set.
Low eight bits of the CAN id is fetched from GUID[15]
that is LSB of GUID.
\return CANAL_ERROR_SUCCESS if success false if not.
*/
int doCmdReceiveLevel1( canalMsg *pCanalMsg );
/*!
This command sets an open interface in the receive loop (RCVLOOP).
It does nothing other then putting the interface in the loop mode and
checking that it went there.
Note! The only way to terminate this receive loop is to close the session
with
doCmdClose (or just close the socket).
\return CANAL_ERROR_SUCCESS if success CANAL_ERROR_GENERIC
if not.
*/
int doCmdEnterReceiveLoop( void );
/*!
Receive an event
The receiveloop command must have been issued for this method to work as
it sets up the ringbuffer used for the blocking receive.
\param pEvent Pointer to VSCP event.
\param timeout Tim out to wait for data. Default = 500 milliseconds.
\return CANAL_ERROR_SUCCESS when event is received. CANAL_ERROR_FIFO_EMPTY
is
returned if no event was available. CANAL_ERROR_TIMEOUT on time-
out.CANAL_ERROR_COMMUNICATION

```

```

is returned if a socket error is detected.
*/
int doCmdBlockReceive( vscpEvent *pEvent, uint32_t timeout = 500 );
/*
Get the number of events in the input queue of this interface
\return the number of events available or if negative
an error code.
*/
int doCmdDataAvailable( void );
/*
Receive CAN status through the interface.
\return CANAL_ERROR_SUCCESS on success and error code if failure.
*/
int doCmdStatus( canalStatus *pStatus );
/*
Receive CAN statistics through the interface.
\return CANAL_ERROR_SUCCESS on success and error code if failure.
*/
int doCmdStatistics( canalStatistics *pStatistics );
/*
Set/Reset a filter through the interface.
\return CANAL_ERROR_SUCCESS on success and error code if failure.
*/
int doCmdFilter( const vscpEventFilter *pFilter );
/*
Set/Reset filter through the interface.
\param filter Filter on string form (priority,class,type,guid).
\param mask Mask on string form (priority,class,type,guid).
\return CANAL_ERROR_SUCCESS on success and error code if failure.
*/
int doCmdFilter( const wxString& filter, const wxString& mask );
/*
Get i/f version through the interface.
\return version number.
*/
unsigned long doCmdVersion( void );
/*
Get interface version
*/
unsigned long doCmdDLLVersion( void );
/*
Get vendor string
*/
const char * doCmdVendorString( void );
/*
Get Driver information string.

```

```

*/
const char * doCmdGetDriverInfo( void );
/*!
Get GUID for this interface.
\return CANAL_ERROR_SUCCESS on success and error code if failure.
*/
int doCmdGetGUID( char *pGUID );
/*!
Set GUID for this interface.
\param pGUID Array with uint8_t GUID's. (Note not a string).
\return CANAL_ERROR_SUCCESS on success and error code if failure.
*/
int doCmdSetGUID( const char *pGUID );
/*!
Get information about a channel.
\param pChannelInfo The structure that will get the information
\return CANAL_ERROR_SUCCESS on success and error code if failure.
*/
int doCmdGetChannelInfo( VSCPChannelInfo *pChannelInfo );
/*!
Get Channel ID for the open channel
\param pChannelID Channel ID for channel
\return CANAL_ERROR_SUCCESS on success and error code if failure.
*/
int doCmdGetChannelID( uint32_t *pChannelID );
/*!
Get available interfaces
\param array A string array that will get interface list
\return CANAL_ERROR_SUCCESS on success and error code if failure.
*/
int doCmdInterfaceList( wxArrayString& array );
/*!
Dummy for Baudrate setting
*/
int doCmdSetBaudrate( uint32_t baudrate )
{ baudrate = baudrate; return CANAL_ERROR_SUCCESS; };
/*!
Dummy for filter setting
*/
int doCmdFilter( uint32_t filter )
{ filter= filter; return CANAL_ERROR_SUCCESS; };
/*!
Dummy for mask setting
*/
int doCmdMask( uint32_t mask )
{ mask = mask; return CANAL_ERROR_SUCCESS; };

```

```

// -----
// V A R I A B L E H A N D L I N G
// -----
/*!
Shutdown the VSCP daemon
*/
int doCmdShutDown( void );
/*!
Get variable value from string variable
\param name of variable
\param strValue pointer to string that get the value of the string variable.
\return true if the variable is of type string and the operation
was successful.
*/
bool getVariableString( wxString& name, wxString *strValue );
/*!
Set variable value from string variable
\param name of variable
\param strValue value to set string variable to.
\return true if the variable is of type string and the operation
was successful.
*/
bool setVariableString( wxString& name, const wxString& strValue );
/*!
Get variable value from boolean variable
\param name of variable
\param bValue pointer to boolean variable that get the value of the string
variable.
\return true if the variable is of type bool and the operation
was successful.
*/
bool getVariableBool( wxString& name, bool *bValue );
/*!
Set variable value from boolean variable
\param name of variable
\param bValue boolean variable with the value to set.
\return true if the variable is of type bool and the operation
was successful.
*/
bool setVariableBool( wxString& name, const bool bValue );
/*!
Get variable value from integer variable
\param name of variable
\param value pointer to integer variable that get the value of the string
variable.
\return true if the variable is of type integer and the operation

```

```

was successful.
*/
bool getVariableInt( wxString& name, int *value );
/*
Set variable value from integer variable
\param name of variable
\param value integer variable with the value to set.
\return true if the variable is of type integer and the operation
was successful.
*/
bool setVariableInt( wxString& name, int value );
/*
Get variable value from long variable
\param name of variable
\param value pointer to long variable that get the value of the string variable.
\return true if the variable is of type long and the operation
was successful.
*/
bool getVariableLong( wxString& name, long *value );
/*
Set variable value from long variable
\param name of variable
\param value long variable with the value to set.
\return true if the variable is of type long and the operation
was successful.
*/
bool setVariableLong( wxString& name, long value );
/*
Get variable value from double variable
\param name of variable
\param value double variable with the value to set.
\return true if the variable is of type double and the operation
was successful.
*/
bool getVariableDouble( wxString& name, double *value );
/*
Set variable value from double variable
\param name of variable
\param value The double value to set.
\return true if the variable is of type double and the operation
was successful.
*/
bool setVariableDouble( wxString& name, double value );
/*
Get variable value from measurement variable
\param name of variable

```

```

\param strValue String that get that get the
value of the measurement.
\return true if the variable is of type measurement and the operation
was successful.
*/
bool getVariableMeasurement( wxString& name, wxString& strValue );
/*!
set variable value from double variable
\param name of variable
\param strValue pointer to double variable that get the value of the string
variable.
\return true if the variable is of type double and the operation
was successful.
*/
bool setVariableMeasurement( wxString& name, wxString& strValue );
/*!
Get variable value from event variable
\param name of variable
\param pEvent pointer to event variable that get the value of the string
variable.
\return true if the variable is of type VSCP event and the operation
was successful.
*/
bool getVariableEvent( wxString& name, vscpEvent *pEvent );
/*!
set variable value from VSCP event
\param name of variable
\param pEvent pointer to event that is used to set the variable.
\return true if the operation was successful.
*/
bool setVariableEvent( wxString& name, vscpEvent *pEvent );
/*!
Get variable value from event variable
\param name of variable
\param pEvent pointer to event variable that get the value of the string
variable.
\return true if the variable is of type VSCP event and the operation
was successful.
*/
bool getVariableEventEx( wxString& name, vscpEventEx *pEvent );
/*!
set variable value from VSCP event
\param name of variable
\param pEvent pointer to event that is used to set the variable.
\return true if the operation was successful.
*/

```

```

bool setVariableEventEx( wxString& name, vscpEventEx *pEvent );
/*!
Get variable value from GUID variable
\param name of variable
\param pGUID pointer to event variable that get the value of the GUID
variable.
\return true if the variable is of type VSCP GUID and the operation
was successful.
*/
bool getVariableGUID( wxString& name, cguid& pGUID );
/*!
set variable value from GUID
\param name of variable
\param pGUID pointer to GUID that is used to set the variable.
\return true if the operation was successful.
*/
bool setVariableGUID( wxString& name, cguid& pGUID );
/*!
Get variable value from VSCP data variable
\param name of variable
\param psizeData pointer to variable that will hold the size of the data array
\param pData pointer to VSCP data array variable (unsigned char [8] ) that
get the
value of the string variable.
\return true if the variable is of type VSCP data and the operation
was successful.
*/
bool getVariableVSCPdata( wxString& name, uint16_t *psizeData, uint8_t
*pData );
/*!
set variable value from VSCP data
\param name of variable.
\param sizeData Size of data.
\param pData Pointer to data array to set data from.
\return true if the operation was successful.
*/
bool setVariableVSCPdata( wxString& name, uint16_t sizeData, uint8_t
*pData );
/*!
Get variable value from class variable
\param name of variable
\param vscp_class pointer to int that get the value of the class variable.
\return true if the variable is of type VSCP class and the operation
was successful.
*/
bool getVariableVSCPclass( wxString& name, uint16_t *vscp_class );

```

```

/*!
set variable value from vscp_class.
\param name of variable.
\param vscp_class to write to variable.
\return true if the operation was successful.
*/
bool setVariableVSCPclass( wxString& name, uint16_t vscp_class );
/*!
Get variable value from type variable
\param name of variable
\param vscp_type pointer to int that get the value of the type variable.
\return true if the variable is of type VSCP type and the operation
was successful.
*/
bool getVariableVSCPtype( wxString& name, uint8_t *vscp_type );
/*!
set variable value from vscp_type.
\param name of variable.
\param vscp_type to write to variable.
\return true if the operation was successful.
*/
bool setVariableVSCPtype( wxString& name, uint16_t vscp_type );
/*!
Get a VSCP event from a line of data from the server
\param strLine String with server event data line
\param pEvent Pointer to VSCP event.
\return true on success.
*/
bool getEventFromLine( const wxString& strLine, vscpEvent *pEvent );
/*!
Set response timeout
\param to Timeout value in seconds. (Default = 2 seconds.)
*/
void setResponseTimeout( uint8_t to ) { if ( to ) m_responseTimeOut =
to; }
// _____
// T H R E A D E D I N T E R F A C E
// _____
// _____
protected:
/// Socket
wxSocketClient* m_psock;
/// Server address
wxIPV4address m_addr;
/// Response string
wxString m_strReply;

```

```

/// Error storage
uint32_t m_err;
/// Last binary error
uint8_t m_lastBinaryError;
/// Flag for active receive loop
bool m_bModeReceiveLoop;
/// Server response timeout
uint8_t m_responseTimeOut;
};

// *****
//
// The following is helper code to make a threaded interface to the above
// function calls.
// The thread should be created JOINABLE and be terminated by setting
// to true. Before starting the thread the m_pCtrlObject must be set up
// to point to an initialized ctrlObjVscpTcpIf
//
// Receive tread states
/*
\def RX_TREAD_STATE_NONE
Idle state
*/
#define RX_TREAD_STATE_NONE 0
/*
\def RX_TREAD_STATE_CONNECTED
Connected state
*/
#define RX_TREAD_STATE_CONNECTED 1
/*
\def RX_TREAD_STATE_FAIL_DISCONNECTED
Fail state
*/
#define RX_TREAD_STATE_FAIL_DISCONNECTED 2
/*
\def RX_TREAD_STATE_DISCONNECTED
Disconnected state
*/
#define RX_TREAD_STATE_DISCONNECTED 3
/// Maximum number of events in receive queue
#define MAX_TREAD_RECEIVE_EVENTS 8192
/*
\class ctrlObjVscpTcpIf
\brief Shared VSCP tcp/ip object among worker threads.
*/
class ctrlObjVscpTcpIf
{

```

```

public:
    /// Constructor
    ctrlObjVscpTcpIf();
    /// Destructor
    ~ctrlObjVscpTcpIf();
    /// Username for VSCP serer
    wxString m_strUsername;
    /// Password for VSCP server
    wxString m_strPassword;
    /// Host address for VSCP server
    wxString m_strHost;
    /// Port for VSCP server
    int m_port;
    /// Thread run control (set to true to terminate thread)
    bool m_bQuit;
    /// Thread error return code
    long m_error;
    /// RX Thread run state
    uint8_t m_rxState;
    /*!
     * Channel id for receive channel
     */
    uint32_t m_rxChannelID;
    /*!
     * Channel id for transmit channel
     * If bFilterOwnTx is true events
     * with this obid will not be added
     * to the queue.
     */
    uint32_t m_txChannelID;
    /*!
     * Filter our own tx events identified by
     * obid set in m_txChannelID
     */
    bool m_bFilterOwnTx;
    /*!
     * Send wxWidgets receive event instead of writing
     * event to RX queue.
     * m_pWnd must point to a valid window and
     * the id for that window must be in m_wndID
     * Also TX events will be activated if thread is
     * started.
     */
    bool m_bUseRXTXEvents;
    /*!
     * Pointer to window that receive events.

```

```

NULL if not used.
*/
wxWindow *m_pWnd;
/// Id for owner window (0 if not used).
uint32_t m_wndID;
/// Event Input queue
EVENT_RX_QUEUE m_rxQueue;
// @{
/// Protection for input queue
wxMutex m_mutexRxQueue;
wxSemaphore m_semRxQueue;
// @}
/// Max events in queue
uint32_t m_maxRXqueue;
/// Event output queue
EVENT_TX_QUEUE m_txQueue;
// @{
/// Protection for output queue
wxMutex m_mutexTxQueue;
wxSemaphore m_semTxQueue;
// @}
};

/*
\class VSCPTCPIP_RX_WorkerThread
\brief This class implement a thread that handles
receive events for a link to the VSCP daemon.
*/
class VSCPTCPIP_RX_WorkerThread : public wxThread
{
public:
    /// Constructor
    VSCPTCPIP_RX_WorkerThread();
    /// Destructor
    virtual ~VSCPTCPIP_RX_WorkerThread();
/*
Thread code entry point
*/
    virtual void *Entry();
/*
called when the thread exits - whether it terminates normally or is
stopped with Delete() (but not when it is Kill()ed!)
*/
    virtual void OnExit();
/*
Pointer to control object.
*/

```

```

ctrlObjVscpTcpIf *m_pCtrlObject;
};

<*/
\class VSCPTCPIP_TX_WorkerThread
\brief This class implement a thread that handles
transmit events for a link to the VSCP daemon.
*/
class VSCPTCPIP_TX_WorkerThread : public wxThread
{
public:
    /// Constructor
    VSCPTCPIP_TX_WorkerThread();
    /// Destructor
    virtual ~VSCPTCPIP_TX_WorkerThread();
<*/
    Thread code entry point
*/
    virtual void *Entry();
<*/
    called when the thread exits - whether it terminates normally or is
    stopped with Delete() (but not when it is Kill(ed!))
*/
    virtual void OnExit();
<*/
    Pointer to the daemon main control object.
*/
    ctrlObjVscpTcpIf *m_pCtrlObject;
};

#endif // !defined(AFX_VSCPTCPIF_H__C2A773AD_8886_40F0_96C4_4DCA663402B2__INCLUDED)

```

Part VI

Firmware

The firmware code is located under the firmware folder. Here are files for different hardware architectures and code common for all of them in the *common* folder.

24 How to port VSCP to new platform

To port VSCP to a new platform is not very hard even if it involves some steps.

- Download the source distribution or even better check out the svn repository. Instructions on how to do this is in our download section.
- Locate the firmware/common folder in the source. Here the two file *vscp_firmware.c* and *vscp_firmware.h* is located which actually implement a full VSCP node. The folder src/vscp/common holds other important files which are aimed for higher level systems but the files *vscp_class.h* which defines all classes and the *vscp_type.h* which defines types is needed.
- inttypes.h is needed by the VSCP code. Most systems have this code already available if your system does not there is an inttype.h in the common folder. Check that it is correct for your firmware.
- Now think of how the registers should be organized for your device. Most nodes need a zone and at least one sub-zone or maybe several. All registers are 8-bit wide so if you have larger values they need to be split. In this case place the MSB first i.e. at the lowest register position. The VSCP Kelvin node (found here http://www.vscp.org/wiki/doku.php/kelvin_smart_ii_-temperature_times_2_module) can be used as a guide.
- Now define the events the node should send. Also determine if it should respond to certain events. All CLASS1.PROTOCOL such as register read/writes are taken care of by the *vscp_firmware.c* code
- Now start to code on your platform. Get the system up and running and construct a one millisecond timebase. Update the vscp_timer here.
- Add another timing variable that just like the vscp_timer is updated every millisecond.
- In the free flowing main loop of your program check if the timing variable of above is > 1000 that is if one second has elapsed. Now set the value to zero and call *vscp_doOneSecondWork()*
- Each VSCP node should have a init button. Typically the init button should be pressed for two seconds before an init condition occurs. A

variable *vscp_initbtncnt* hold this information and the buttons should be checked in the timebase and set to zero if the button is not pressed or increased by one as long as it is pressed.

- Each VSCP node should have a status LED. This LED should blink when the node is initializing. That is searching for a free nickname address. It should light steady when the node is active and be turned off if init error. Typical code in the timebase can look like this for a PIC processor

```

1  // Status LED
2  vscp_statuscnt++;
3  if ( ( VSCP_LED_BLINK1 == vscp_initledfunc )
4      && ( vscp_statuscnt > 100 ) ) {
5      if ( PORTCbits.RC1 ) {
6          PORTCbits.RC1 = 0;
7      }
8      else {
9          PORTCbits.RC1 = 1;
10     }
11
12     vscp_statuscnt = 0;
13
14 } else if ( VSCP_LED_ON == vscp_initledfunc ) {
15     PORTCbits.RC1 = 1;
16     vscp_statuscnt = 0;
17 } else if ( VSCP_LED_OFF == vscp_initledfunc ) {
18     PORTCbits.RC1 = 0;
19     vscp_statuscnt = 0;
20 }
```

- A typical main loop can look something like this

```

1 //*****
2 // Main Routine
3 //*****
4
5 void main()
6 {
7     init();
8     // Initialize Microcontroller
9     // Check VSCP persistent storage and
10    // restore if needed
11    if ( !vscp_check_pstorage() ) {
12        init_app_eeprom(); // Initialize the application EEPROM
13    }
14
15    vscp_init(); // Initialize the VSCP functionality
16
17    while ( 1 ) { // Loop Forever
18        ClrWdt(); // Feed the dog
19        if ( ( vscp_initbtncnt > 500 ) &&
20             ( VSCP_STATE_INIT != vscp_node_state ) ) {
```

```

21          // Init button pressed
22          vscp_nickname = VSCP_ADDRESS_FREE;
23          writeEEPROM( VSCP_EEPROM_NICKNAME, VSCP_ADDRESS_FREE );
24          vscp_init();
25      }
26
27      // Check for a valid event
28      vscp_imsg.flags = 0;
29      vscp_getEvent();
30
31      // do a measurement if needed
32      if ( measurement_clock > 1000 ) {
33          measurement_clock = 0;
34          // Do VSCP one second jobs
35          vscp_doOneSecondWork();
36
37          switch ( vscp_node_state ) {
38              case VSCP_STATE_STARTUP: // Cold/warm reset
39
40                  // Get nickname from EEPROM
41                  if ( VSCP_ADDRESS_FREE == vscp_nickname ) {
42                      // new on segment need a nickname
43                      vscp_node_state = VSCP_STATE_INIT;
44                  }
45                  else { // been here before - go on
46                      vscp_node_state = VSCP_STATE_ACTIVE;
47                      vscp_goActiveState();
48                  }
49                  break;
50
51              case VSCP_STATE_INIT: // Assigning nickname
52                  vscp_handleProbeState();
53                  break;
54
55              case VSCP_STATE_PREACTIVE: // Waiting for host initialisation
56                  vscp_goActiveState();
57                  break;
58
59              case VSCP_STATE_ACTIVE: // The normal state
60                  if ( vscp_imsg.flags & VSCP_VALID_MSG ) { // event?
61                      vscp_handleProtocolEvent();
62                  }
63                  break;
64
65              case VSCP_STATE_ERROR: // Everything is *very* *very* bad.
66                  vscp_error();
67                  break;
68
69              default: // Should not be here...
70                  vscp_node_state = VSCP_STATE_STARTUP;
71                  break;
72          }
73
74          doWork(); // Do general application work
75
76      } // while

```

- Now construct a MDF file that describes your module and upload it to a server where it can be reached by application software.
- This is it. Now just code your application.
- A good example to look at is the Kelvin Smart II module. The code is available in *firmware/pic/Kelvin/Smart2/project*

25 Firmware common code documentation

The most important files in the common folder is vscp_firmware.h ad vscp_firmware.c which implements the VSCP system on a specific hardware. Today this code has been tested and compiles on common development tools for PIC (Microchip), AVR(Atmel), ARM(Str, Texas, Atmel, Luminary, NXP etc) and other hardware platforms.

25.1 vscp_firmware.h

```
1 //////////////////////////////////////////////////////////////////
2 // File: vscp_firmware.h
3 /**
4  * ****
5 * VSCP (Very Simple Control Protocol)
6 * http://www.vscp.org
7 *
8 * akhe@eurosource.se
9 *
10 * Copyright (C) 1995–2010 Ake Hedman,
11 * eurosource, <akhe@eurosource.se>
12 *
13 * 080702 Cleaned up even more.
14 * 060508 Cleaned up.
15 * 060323 Added DM struct.
16 *
17 * This software is provided 'as-is', without any express or implied
18 * warranty. In no event will the authors be held liable for any damages
19 * arising from the use of this software.
20 *
21 * Permission is granted to anyone to use this software for any purpose,
22 * including commercial applications, and to alter it and redistribute it
23 * freely, subject to the following restrictions:
24 *
25 * 1. The origin of this software must not be misrepresented; you must
26 * not
27 * claim that you wrote the original software. If you use this software
28 * in a product, an acknowledgment in the product documentation would be
29 * appreciated but is not required.
30 * 2. Altered source versions must be plainly marked as such, and must
31 * not be
32 * misrepresented as being the original software.
33 * 3. This notice may not be removed or altered from any source
34 * distribution.
35 *
36 *
37 */
38 #ifndef VSCP_H
39 #define VSCP_H
40 /**
41 * RCSfile: vscp.h,v $
42 * Revision: 1.6 $
43 /**
44 */!
```

```

45  | file firmware_vscp.h
46  \brief VSCP firmware stack
47  This file contains the firmware needed to implement VSCP in a
48  low end hardware device.
49  Notes about defines
50
51  Normally make the following defines in the prjcfg.h file
52  ENABLE_WRITE_2PROTECTED_LOCATIONS
53
54  to make it possible to write GUID,
55  manufacturer ID,
56  manufacturer sub device id
57  page_low (0x92) and page_high (0x93) should both contain 0xFF
58  NOTE the storage must be in EEPROM also for it to work.
59  */
60 #include <vscp_compiler.h> // This file should be in your project folder
61 #include <vscp_projdefs.h> // This file should be in your project folder
62 #include <inttypes.h>
63 //
***** ****
64 // VSCP Constants
65 //
***** ****
66 #define VSCP_MAJOR_VERSION 1 //< VSCP Major version
67 #define VSCP_MINOR_VERSION 5 //< VSCP Minor Version
68 #define VSCP_ADDRESS_MASTER 0x00
69 #define VSCP_ADDRESS_FREE 0xFF
70 #define VSCP_SIZE_GUID 16 //< # GUID bytes
71 #define VSCP_SIZE_DEVURL 32 //< # of device URL bytes
72 #define VSCP_SIZE_STD_DM_ROW 8 //< Size for level I decision matrix row
73 #define VSCP_BOOT_FLAG 0xFF // Boot flag is stored in persistent storage
74 // and if it is there the bootloader will be
75 // activated.
76 // Bootloaders
77 #define VSCP_BOOTLOADER_VSCP 0x00 // VSCP bootloader algorithm
78 #define VSCP_BOOTLOADER_PIC1 0x01 // PIC algorithm 0
79 #define VSCP_BOOTLOADER_AVR1 0x10 // AVR algorithm 0
80 #define VSCP_BOOTLOADER_LPC1 0x20 // NXP/Philips LPC algorithm 0
81 #define VSCP_BOOTLOADER_ST 0x30 // ST STR algorithm 0
82 #define VSCP_BOOTLOADER_NONE 0xFF
83 #define VSCP_LEVEL1_COMMON_REGISTER_START 0x80
84 // State machine states
85 #define VSCP_STATE_STARTUP 0x00 // Cold/warm reset
86 #define VSCP_STATE_INIT 0x01 // Assigning nickname
87 #define VSCP_STATE_PREACTIVE 0x02 // Waiting for host initialisation
88 #define VSCP_STATE_ACTIVE 0x03 // The normal state
89 #define VSCP_STATE_ERROR 0x04 // error state. Big problems.
90 // State machine sub states
91 #define VSCP_SUBSTATE_NONE 0x00 // No state
92 #define VSCP_SUBSTATE_INIT_PROBE_SENT 0x01 // probe sent
93 #define VSCP_SUBSTATE_INIT_PROBE_ACK 0x02 // probe ACK received
94 // Helper consts and
95 #define VSCP_VALID_MSG 0x80 // Bit 7 set in flags
96 #define VSCP_PRIORITY7 0x00
97 #define VSCP_PRIORITY_HIGH 0x00
98 #define VSCP_PRIORITY6 0x01
99 #define VSCP_PRIORITY5 0x02
100 #define VSCP_PRIORITY4 0x03
101 #define VSCP_PRIORITY_MEDIUM 0x03
102 #define VSCP_PRIORITY_NORMAL 0x03
103 #define VSCP_PRIORITY3 0x04
104 #define VSCP_PRIORITY2 0x05
105 #define VSCP_PRIORITY1 0x06
106 #define VSCP_PRIORITY0 0x07
107 #define VSCP_PRIORITY_LOW 0x07
108 #define VSCP_PROBE_TIMEOUT 1000 // ms - one second

```

```

109 #define VSCP_PROBE_TIMEOUT_COUNT 3 // Max # probe timeouts allowed
110 // ****
111 // VSCP Register - Logical positions
112 // ****
113 #define VSCP_REG_ALARMSTATUS 0x80
114 #define VSCP_REG_VSCP_MAJOR_VERSION 0x81
115 #define VSCP_REG_VSCP_MINOR_VERSION 0x82
116 #define VSCP_REG_NODE_CONTROL 0x83
117 #define VSCP_REG_USERID0 0x84
118 #define VSCP_REG_USERID1 0x85
119 #define VSCP_REG_USERID2 0x86
120 #define VSCP_REG_USERID3 0x87
121 #define VSCP_REG_USERID4 0x88
122 #define VSCP_REG_MANUFACTUR_ID0 0x89
123 #define VSCP_REG_MANUFACTUR_ID1 0x8A
124 #define VSCP_REG_MANUFACTUR_ID2 0x8B
125 #define VSCP_REG_MANUFACTUR_ID3 0x8C
126 #define VSCP_REG_MANUFACTUR_SUBID0 0x8D
127 #define VSCP_REG_MANUFACTUR_SUBID1 0x8E
128 #define VSCP_REG_MANUFACTUR_SUBID2 0x8F
129 #define VSCP_REG_MANUFACTUR_SUBID3 0x90
130 #define VSCP_REG_NICKNAME_ID 0x91
131 #define VSCP_REG_PAGE_SELECT_MSB 0x92
132 #define VSCP_REG_PAGE_SELECT_LSB 0x93
133 #define VSCP_REG_FIRMWARE_MAJOR_VERSION 0x94
134 #define VSCP_REG_FIRMWARE_MINOR_VERSION 0x95
135 #define VSCP_REG_FIRMWARE_SUB_MINOR_VERSION 0x96
136 #define VSCP_REG_BOOT_LOADER_ALGORITHM 0x97
137 #define VSCP_REG_BUFFER_SIZE 0x98
138 #define VSCP_REG_PAGES_USED 0x99
139 #define VSCP_REG_GUID 0xD0
140 #define VSCP_REG_DEVICE_URL 0xE0
141 // INIT LED function codes
142 #define VSCP_LED_OFF 0x00
143 #define VSCP_LED_ON 0x01
144 #define VSCP_LED_BLINK1 0x02
145 */
146 struct _imsg
147 Input message
148 */
149 struct _imsg {
150 /*
151 Input message flags\n
152 _____\n
153 Bit 7 - Set if message valid\n
154 Bit 6 - Reserved\n
155 Bit 5 - Hardcoded (will never be set)\n
156 Bit 3 - Number of data bytes MSB\n
157 Bit 2 - Number of data bytes\n
158 Bit 1 - Number of data bytes\n
159 Bit 0 - Number of data bytes LSB\n
160 */
161 uint8_t flags; ///< Input message flags
162 uint8_t priority; ///< Priority for the message 0-7
163 uint16_t class; ///< VSCP class
164 uint8_t type; ///< VSCP type
165 uint8_t oaddr; ///< Packet originating address
166 uint8_t data[8]; ///< data bytes
167 };
168 */
169 struct _omsg
170 Output message
171 */
172 struct _omsg {

```

```

173 /*!
174 Output message flags ( Message to send )\n
175 =====\n
176 Bit 7 - Set if message should be sent (cleard when sent)\n
177 Bit 6 - Reserved\n
178 Bit 5 - Reserved\n
179 Bit 2 - Number of data bytes MSB\n
180 Bit 2 - Number of data bytes \n
181 Bit 1 - Number of data bytes\n
182 Bit 0 - Number of data bytes LSB\n
183 */
184 uint8_t flags; ///< Output message flags
185 uint8_t priority; ///< Priority for the message 0-7
186 uint16_t class; ///< VSCP class
187 uint8_t type; ///< VSCP type
188 //> Originating address is always *this* node
189 uint8_t data[8]; ///< data bytes
190 };
191 */
192 Decision Matrix - definitions
193 A matrix row consist of 8 bytes and have the following format
194 / oaddr / flags / class-mask / class-filter / type-mask / type-filter /
195 action / action-param /
196 oaddr is the originating address.
197 flag
198 =====
199 bit 7 - Enabled (==1).
200 bit 6 - oaddr should be checked (==1) or not checked (==0)
201 bit 5 - Reserved
202 bit 4 - Reserved
203 bit 3 - Reserved
204 bit 2 - Reserved
205 bit 1 - Classmask bit 8
206 bit 0 - Classfilter bit 8
207 Action = 0 is always NOOP, "no operation".
208 */
209 #define VSCP_DM_POS_OADDR 0
210 #define VSCP_DM_POS_FLAGS 1
211 #define VSCP_DM_POS_CLASSMASK 2
212 #define VSCP_DM_POS_CLASSFILTER 3
213 #define VSCP_DM_POS_TYPEMASK 4
214 #define VSCP_DM_POS_TYPEFILTER 5
215 #define VSCP_DM_POS_ACTION 6
216 #define VSCP_DM_POS_ACTIONPARAM 7
217 #define VSCP_DM_FLAG_ENABLED 0x80
218 #define VSCP_DM_FLAG_CHECK_OADDR 0x40
219 #define VSCP_DM_FLAG_HARDCODED 0x20
220 #define VSCP_DM_FLAG_CHECK_ZONE 0x10
221 #define VSCP_DM_FLAG_CHECK_SUBZONE 0x08
222 #define VSCP_DM_FLAG_CLASS_MASK 0x02
223 #define VSCP_DM_FLAG_CLASS_FILTER 0x01
224 */
225 struct _dmrow {
226 Decision matrix row element (for RAM storage)
227 Each DM row consist of a structure of this type.
228 */
229 struct _dmrow {
230 uint8_t oaddr; ///< Originating address
231 uint8_t flags; ///< Decion matrix row flags
232 uint8_t class_mask; ///< Mask for class (lower eight bits)
233 uint8_t class_filter; ///< Filter for class (lower eight bits)
234 uint8_t type_mask; ///< Mask for type
235 uint8_t type_filter; ///< Filter for type
236 uint8_t action; ///< Action code
237 uint8_t action_param; ///< Action parameter
238 };

```

```

239  /// External - VSCP Data
240  extern uint8_t vscp_nickname; ///< Assigned node nickname
241  extern uint8_t vscp_errorcnt; ///< VSCP error counter
242  extern uint8_t vscp_alarmstatus; ///< VSCP alarm status register
243  extern uint8_t vscp_node_state; ///< VSCP state machine main state
244  extern uint8_t vscp_node_substate; ///< VSCP state machine sub state
245  extern uint8_t vscp_initledfunc; ///<
246  /// The following are defined in vscp_firmware.c
247  extern struct _imsg vscp_imsg; ///< Current input event
248  extern struct _omsg vscp_omsg; ///< Current outgoing event
249  extern volatile uint16_t vscp_timer; ///< 1 ms timer counter
250  extern uint8_t vscp_probe_address; ///< Probe address for nickname
251  discovery
252  extern volatile uint8_t vscp_initbtncnt; ///< init button counter
253  extern volatile uint8_t vscp_statuscnt; ///< status LED counter
254  extern uint16_t vscp_page_select; ///< Selected Register Page
255  // Prototypes
256  */
257  \fn vscp_init
258  Init the VSCP firmware.
259  Call this before entering the mainloop.
260  */
261  void vscp_init( void );
262  /*
263  Set VSCP error state
264  */
265  void vscp_error( void );
266  /*
267  Handle nickname probing
268  This routine should be called periodically while
269  in state VSCP_STATE_INIT
270  */
271  void vscp_handleProbeState( void );
272  /*
273  Handle the preactive state
274  This state is entered if a nod with nickname=0 answers the
275  probe. Zero is reserved for a segment controller and if it
276  is available and acknowledge this way that it is the node should
277  wait for the segment controller to set its nickname. This should
278  happen before a preset timeout (1 second) and if this time is
279  exceeded the init process is started again to give the segment controller
280  more chances to do its job.
281  */
282  void vscp_handlePeActiveState( void );
283  /*
284  Handle incoming CLASS1.PROTOCOL event
285  The event should be in the vscp_imsg buffer on entry.
286  */
287  void vscp_handleProtocolEvent( void );
288  /*
289  Go to the active state
290  This mean the node sends new node online and informs other nodes
291  about its aquired nickname.
292  */
293  void vscp_goActiveState( void );
294  /*
295  Send periodic heartbeat
296  This event should be called by the application perodic
297  to inform the world about its existance and tell it's
298  alive.
299  @param zone Zone the module belongs to or zero if no zone.
300  @param subzone Sub-zone the module belongs to or zero i no zubzone.
301  */
302  void vscp_sendHeartBeat( uint8_t zone, uint8_t subzone );
303  /*
304  Handle received VSCP segment controller heartbeat
*/

```

```

305 void vscp_handleHeartbeat( void );
306 /*!
307 Handle Set Nickname event
308 */
309 void vscp_handleSetNickname( void );
310 /*!
311 Handle Drop Nickname
312 */
313 void vscp_handleDropNickname( void );
314 /*!
315 Report a new node on the bus
316 */
317 void vscp_newNodeOnline( void );
318 /*!
319 Read a VSCP register
320 @param reg Register to read.
321 @return Content of register.
322 */
323 uint8_t vscp_readRegister( uint8_t reg );
324 /*!
325 Read standard register (upper part)
326 @param reg Register to read (>=0x80)
327 @return Register content or 0xFF for nn valid register
328 */
329 uint8_t vscp_readStdReg( uint8_t reg );
330 /*!
331 Write VSCP register
332 @param reg Register to write to
333 @param value Value to write
334 @return Content of register after write.
335 */
336 uint8_t vscp_writeRegister( uint8_t reg, uint8_t value );
337 /*!
338 Write standard register (upper part)
339 @param reg Register to write to
340 @param value Value to write
341 @return Content of register after write.
342 */
343 uint8_t vscp_writeStdReg( uint8_t reg, uint8_t value );
344 /*!
345 Do One second work
346 This routine should be called once a second by the
347 application.
348 */
349 void vscp_doOneSecondWork( void );
350 /*!
351 */
352 int8_t vscp_check_pstorage( void );
353 /*!
354 Send VSCP event in the out-buffer.
355 @return TRUE on success.
356 */
357 int8_t vscp_sendEvent( void );
358 /*!
359 Get VSCP Event if there is no message in the input buffer.
360 @return TRUE if a valid event is placed in the in-buffer.
361 */
362 int8_t vscp_getEvent( void );
363 // _____ External Functions
364 //
365 // All functions below should be implemented by the application
366 //
367 // _____ External Functions
368 /*!
369 Get a VSCP frame frame
370 @param pvsycopclass Pointer to variable that will get VSCP class.

```

```

371  @param pVscptype Pointer to variable which will get VSCP type.
372  @param pNodeId Pointer to variable which will get node-ID.
373  @param pPriority Pointer to variable which will get priority (0-7).
374  @param pSize Pointer to variable that will get datasize.
375  @param pData pointer to array that will get event data.
376  @return TRUE on success.
377  */
378 int8_t getVSCPFrame( uint16_t *pVscptype ,
379  uint8_t *pNodeId ,
380  uint8_t *pPriority ,
381  uint8_t *pSize ,
382  uint8_t *pData );
383 /*!
384 Send a VSCP frame
385 @param vscptype VSCP class for event.
386 @param vscptype VSCP type for event.
387 @param nodeid Nodeid for originating node.
388 @param priority Priority for event.
389 @param size Size of data portion.
390 @param pData Pointer to event data.
391 @return TRUE on success.
392 */
393 int8_t sendVSCPFrame( uint16_t vscptype ,
394  uint8_t vscptype ,
395  uint8_t nodeid ,
396  uint8_t priority ,
397  uint8_t size ,
398  uint8_t *pData );
399 /*!
400 The following methods must be defined
401 in the application and should return firmware version
402 information
403 */
404 uint8_t vscp_getMajorVersion( void );
405 uint8_t vscp_getMinorVersion( void );
406 uint8_t vscp_getSubMinorVersion( void );
407 /*!
408 Get GUID from permanent storage
409 */
410 uint8_t vscp_getGUID( uint8_t idx );
411 void vscp_setGUID( uint8_t idx , uint8_t data );
412 /*!
413 User ID 0 idx=0
414 User ID 1 idx=1
415 User ID 2 idx=2
416 User ID 3 idx=3
417 */
418 uint8_t vscp_getUserID( uint8_t idx );
419 void vscp_setUserID( uint8_t idx , uint8_t data );
420 /*!
421 Handle manufacturer ID.
422 Not that both main and sub ID are fetched here
423 Manufacturer device ID byte 0 - idx=0
424 Manufacturer device ID byte 1 - idx=1
425 Manufacturer device ID byte 2 - idx=2
426 Manufacturer device ID byte 3 - idx=3
427 Manufacturer device sub ID byte 0 - idx=4
428 Manufacturer device sub ID byte 1 - idx=5
429 Manufacturer device sub ID byte 2 - idx=6
430 Manufacturer device sub ID byte 3 - idx=7
431 */
432 uint8_t vscp_getManufacturerId( uint8_t idx );
433 void vscp_setManufacturerId( uint8_t idx , uint8_t data );
434 /*!
435 Get bootloader algorithm from permanent storage
436 */
437 uint8_t vscp_getBootLoaderAlgorithm( void );

```

```

439  /*!
440   Get buffer size
441  */
442 uint8_t vscp_getBufferSize( void );
443 /*!
444   Get number of register pages used by app.
445  */
446 uint8_t vscp_getRegisterPagesUsed( void );
447 /*!
448   Get URL from device from permanent storage
449   index 0-15
450  */
451 uint8_t vscp_getMDF_URL( uint8_t idx );
452 /*!
453   Fetch nickname from permanent storage
454   @return read nickname.
455  */
456 uint8_t vscp_readNicknamePermanent( void );
457 /*!
458   Write nickname to permanent storage
459   @param nickname to write
460  */
461 void vscp_writeNicknamePermanent( uint8_t nickname );
462 /*!
463   Fetch segment CRC from permanent storage
464  */
465 uint8_t vscp_getSegmentCRC( void );
466 /*!
467   Write segment CRC to permanent storage
468  */
469 void vscp_setSegmentCRC( uint8_t crc );
470 /*!
471   Write control byte permanent storage
472  */
473 void vscp_setControlByte( uint8_t ctrl );
474 /*!
475   Fetch control byte from permanent storage
476  */
477 uint8_t vscp_getControlByte( void );
478 /*!
479   Get page select bytes
480   idx=0 - byte 0 MSB
481   idx=1 - byte 1 LSB
482  */
483 uint8_t vscp_getPageSelect( uint8_t idx );
484 /*!
485   Set page select registers
486   @param idx 0 for LSB, 1 for MSB
487   @param data Byte to set of page select registers
488  */
489 void vscp_setPageSelect( uint8_t idx, uint8_t data );
490 /*!
491   Read application register (lower part)
492   @param reg Register to read (<0x80)
493   @return Register content or 0xFF for non valid register
494  */
495 uint8_t vscp_readAppReg( uint8_t reg );
496 /*!
497   Write application register (lower part)
498   @param reg Register to read (<0x80)
499   @param value Value to write to register.
500   @return Register content or 0xFF for non valid register
501  */
502 uint8_t vscp_writeAppReg( uint8_t reg, uint8_t value );
503 /*!
504   Get DM matrix info
505   The output message data structure should be filled with
506   the following data by this routine.

```

```

507 byte 0 - Number of DM rows. 0 if none.
508 byte 1 - offset in register space.
509 byte 2 - start page MSB
510 byte 3 - start page LSB
511 byte 4 - End page MSB
512 byte 5 - End page LSB
513 byte 6 - Level II size of DM row (Just for Level II nodes).
514 */
515 void vscp_getMatrixInfo( char *pData );
516 /*!
517 Get embedded MDF info
518 If available this routine sends an embedded MDF file
519 in several events. See specification CLASS1.PROTOCOL
520 Type=35/36
521 */
522 void vscp_getEmbeddedMdflInfo( void );
523 /*!
524 Go bootloader mode
525 This routine force the system into bootloader mode according
526 to the selected protocol.
527 */
528 void vscp_goBootloaderMode( void );
529 /*!
530 Get Zone for device
531 Just return zero if not used.
532 */
533 uint8_t vscp_getZone( void );
534 /*!
535 Get Sub-zone for device
536 Just return zero if not used.
537 */
538 uint8_t vscp_getSubzone( void );
539 #endif

```

25.2 vscp_firmware.c

```
1 ///////////////////////////////////////////////////////////////////
2 // File: vscp_firmware.c
3 /**
4 */
*****  

5 * VSCP (Very Simple Control Protocol)
6 * http://www.vscp.org
7 *
8 * akhe@eurosource.se
9 *
10 * Copyright (C) 1995–2010 Ake Hedman,
11 * eurosource, <akhe@eurosource.se>
12 *
13 * This software is provided 'as-is', without any express or implied
14 * warranty. In no event will the authors be held liable for any damages
15 * arising from the use of this software.
16 *
17 * Permission is granted to anyone to use this software for any purpose,
18 * including commercial applications, and to alter it and redistribute it
19 * freely, subject to the following restrictions:
20 *
21 * 1. The origin of this software must not be misrepresented; you must
22 * not
23 * claim that you wrote the original software. If you use this software
24 * in a product, an acknowledgment in the product documentation would be
25 * appreciated but is not required.
26 * 2. Altered source versions must be plainly marked as such, and must
27 * not be
28 * misrepresented as being the original software.
29 * 3. This notice may not be removed or altered from any source
30 * distribution.
31 *
32 *
*****  

33 */
34 /**
35 * RCSfile: vscp.c,v $
36 * $Revision: 1.6 $
37 */
38 #include <string.h>
39 #include <stdlib.h>
40 #include "firmware_vscp.h"
41 #include "../src/vscp/common/vscp_class.h"
42 #include "../src/vscp/common/vscp_type.h"
43 #ifndef FALSE
44 #define FALSE 0
45 #endif
46 #ifndef TRUE
47 #define TRUE !FALSE
48 #endif
49 #ifndef ON
50 #define ON !FALSE
51 #endif
52 #ifndef OFF
53 #define OFF FALSE
54 #endif
55 // Globals
56 // VSCP Data
```

```

57 uint8_t vscp_nickname; // < Node nickname
58 uint8_t vscp_errorcnt; // VSCP/CAN errors
59 uint8_t vscp_alarmstatus; // VSCP Alarm Status
60 uint8_t vscp_node_state; // State machine state
61 uint8_t vscp_node_substate; // State machine substate
62 uint8_t vscp_probe_cnt; // Number of timeout probes
63 // Incoming message
64 struct _imsg vscp_imsg;
65 // Outgoing message
66 struct _omsg vscp_omsg;
67 uint8_t vscp_probe_address; // Address used during initialization
68 uint8_t vscp_initledfunc; // Init LED functionality
69 volatile uint16_t vscp_timer; // 1 ms timer counter
70 // Should be externally updated.
71 volatile uint8_t vscp_initbtncnt; // init button counter
72 // increase this value by one each millisecond
73 // the initbutton is pressed. Set to zero when button
74 // is released.
75 volatile uint8_t vscp_statususcnt; // status LED counter
76 // increase bye one every millisecond
77 // page selector
78 uint16_t vscp_page_select;
79 // The GUID reset is used when the VSCP_TYPE_PROTOCOL_RESET DEVICE
80 // is received. Bit 4,5,6,7 is set for each received frame with
81 // GUID data. Bit 4 is for index = 0, bit 5 is for index = 1 etc.
82 // This means that a bit is set if a frame with correct GUID is
83 // received.
84 uint8_t vscp_guid_reset;
85 // Timekeeping
86 uint8_t vscp_second;
87 uint8_t vscp_minute;
88 uint8_t vscp_hour;
89 // /////////////////////////////////
90 // vscp_init
91 //
92 void vscp_init( void )
93 {
94 vscp_initledfunc = VSCP_LED_BLINK1;
95 // read the nickname-ID
96 vscp_nickname = vscp_readNicknamePermanent();
97 // if zero set to uninitialized
98 if ( !vscp_nickname ) vscp_nickname = VSCP_ADDRESS_FREE;
99 // Init incoming message
100 vscp_imsg.flags = 0;
101 vscp_imsg.priority = 0;
102 vscp_imsg.class = 0;
103 vscp_imsg.type = 0;
104 // Init outgoing message
105 vscp_omsg.flags = 0;
106 vscp_omsg.priority = 0;
107 vscp_omsg.class = 0;
108 vscp_omsg.type = 0;
109 vscp_errorcnt = 0; // No errors yet
110 vscp_alarmstatus = 0; // No alarmstatus
111 vscp_probe_address = 0;
112 // Initial state
113 vscp_node_state = VSCP_STATE_STARTUP;
114 vscp_node_substate = VSCP_SUBSTATE_NONE;
115 vscp_probe_cnt = 0;
116 vscp_page_select = 0;
117 // Initialize time keeping
118 vscp_timer = 0;
119 vscp_second = 0;
120 vscp_minute = 0;
121 vscp_hour = 0;
122 }

```

```

123 //      /////////////////////////////////
124 // vscp_check_pstorage
125 // Check control position integrity and restore EEPROM
126 // if invalid.
127 //
128 int8_t vscp_check_pstorage( void )
129 {
130     // controlbyte == 01xxxxxx means initialized
131     // everything else is uninitialized
132     if ( ( vscp_getSegmentCRC() & 0xC0 ) == 0x40 ) {
133         return TRUE;
134     }
135     // No nickname yet
136     vscp_writeNicknamePermanent( 0xFF );
137     // No segment CRC yet.
138     vscp_setSegmentCRC( 0x00 );
139     // Initial startup
140     // write allowed
141     vscp_setControlByte( 0x90 );
142     return FALSE;
143 }
144 //
145 //      /////////////////////////////////
146 // vscp_error
147 //
148 void vscp_error( void )
149 {
150     vscp_initledfunc = VSCP_LED_OFF;
151 }
152 //
153 // vscp_handleProbe
154 //
155 void vscp_handleProbeState( void )
156 {
157     switch ( vscp_node_substate ) {
158     case VSCP_SUBSTATE_NONE:
159         if ( VSCP_ADDRESS_FREE != vscp_probe_address ) {
160             vscp_omsg.flags = VSCP_VALID_MSG + 1; // one databyte
161             vscp_omsg.priority = VSCP_PRIORITY_HIGH;
162             vscp_omsg.class = VSCP_CLASS1_PROTOCOL;
163             vscp_omsg.type = VSCP_TYPE_PROTOCOL_NEW_NODE_ONLINE;
164             vscp_omsg.data[ 0 ] = vscp_probe_address;
165             // send the probe
166             vscp_sendEvent();
167             vscp_node_substate = VSCP_SUBSTATE_INIT_PROBE_SENT;
168             vscp_timer = 0;
169         }
170     else {
171         // No free address -> error
172         vscp_node_state = VSCP_STATE_ERROR;
173         // Tell system we are giving up
174         vscp_omsg.flags = VSCP_VALID_MSG + 1; // one databyte
175         vscp_omsg.data[ 0 ] = 0xFF; // we are unassigned
176         vscp_omsg.priority = VSCP_PRIORITY_LOW;
177         vscp_omsg.class = VSCP_CLASS1_PROTOCOL;
178         vscp_omsg.type = VSCP_TYPE_PROTOCOL_PROBE_ACK;
179         // send the error event
180         vscp_sendEvent();
181     }
182     break;
183     case VSCP_SUBSTATE_INIT_PROBE_SENT:
184     if ( vscp_imsg.flags & VSCP_VALID_MSG ) { // incoming message?

```

```

185 // Yes, incoming message
186 if ( ( VSCP_CLASS1_PROTOCOL == vscp_imsg.class ) &&
187 ( VSCP_TYPE_PROTOCOL_PROBE_ACK == vscp_imsg.type ) ) {
188 // Yes it was an ack from the segment master or a node
189 if ( 0 == vscp_probe_address ) {
190 // Master controller answered
191 // wait for address
192 vscp_node_state = VSCP_STATE_PREACTIVE;
193 vscp_timer = 0; // reset timer
194 }
195 else {
196 // node answered, try next address
197 vscp_probe_address++;
198 vscp_node_substate = VSCP_SUBSTATE_NONE;
199 vscp_probe_cnt = 0;
200 }
201 }
202 }
203 else {
204 if ( vscp_timer > VSCP_PROBE_TIMEOUT ) { // Check for timeout
205 vscp_probe_cnt++; // Another timeout
206 if ( vscp_probe_cnt > VSCP_PROBE_TIMEOUT_COUNT ) {
207 // Yes we have a timeout
208 if ( 0 == vscp_probe_address ) { // master controller probe?
209 // No master controller on segment, try next node
210 vscp_probe_address++;
211 vscp_node_substate = VSCP_SUBSTATE_NONE;
212 vscp_timer = 0;
213 }
214 else {
215 // We have found a free address - use it
216 vscp_nickname = vscp_probe_address;
217 vscp_node_state = VSCP_STATE_ACTIVE;
218 vscp_node_substate = VSCP_SUBSTATE_NONE;
219 vscp_writeNicknamePermanent( vscp_nickname );
220 vscp_setSegmentCRC( 0x40 ); // segment code (non server segment )
221 // Report success
222 vscp_probe_cnt = 0;
223 vscp_goActiveState();
224 }
225 }
226 else {
227 vscp_node_substate = VSCP_SUBSTATE_NONE;
228 }
229 } // Timeout
230 }
231 break;
232 case VSCP_SUBSTATE_INIT_PROBE_ACK:
233 break;
234 default :
235 vscp_node_substate = VSCP_SUBSTATE_NONE;
236 break;
237 }
238 vscp_imsg.flags = 0;
239 }
240 ///////////////////////////////////////////////////////////////////
241 // vscp_handlePreActiveState
242 //
243 void vscp_handlePreActiveState( void )
244 {
245 if ( vscp_imsg.flags & VSCP_VALID_MSG ) { // incoming message?
246 if ( ( VSCP_CLASS1_PROTOCOL == vscp_imsg.class ) &&
247 ( VSCP_TYPE_PROTOCOL_SET_NICKNAME == vscp_imsg.type ) &&
248 ( VSCP_ADDRESS_FREE == vscp_imsg.data[ 0 ] ) ) {
249 // Assign nickname
250 vscp_nickname = vscp_imsg.data[ 2 ];

```

```

251 vscp_writeNicknamePermanent( vscp_nickname );
252 vscp_setSegmentCRC( 0x40 );
253 // Go active state
254 vscp_node_state = VSCP_STATE_ACTIVE;
255 }
256 }
257 else {
258 // Check for time out
259 if ( vscp_timer > VSCP_PROBE_TIMEOUT ) {
260 // Yes, we have a timeout
261 vscp_nickname = VSCP_ADDRESS_FREE;
262 vscp_writeNicknamePermanent( VSCP_ADDRESS_FREE );
263 vscp_init();
264 }
265 }
266 }
267 ///////////////////////////////////////////////////////////////////
268 // vscp_goActiveState
269 //
270 void vscp_goActiveState( void )
271 {
272 vscp_omsg.flags = VSCP_VALID_MSG + 1; // one databyte
273 vscp_omsg.priority = VSCP_PRIORITY_HIGH;
274 vscp_omsg.class = VSCP_CLASS1_PROTOCOL;
275 vscp_omsg.type = VSCP_TYPE_PROTOCOL_NEW_NODE_ONLINE;
276 vscp_omsg.data[ 0 ] = vscp_nickname;
277 // send the message
278 vscp_sendEvent();
279 vscp_initledfunc = VSCP_LED_ON;
280 }
281 ///////////////////////////////////////////////////////////////////
282 // vscp_sendHeartbeat
283 //
284 void vscp_sendHeartBeat( uint8_t zone, uint8_t subzone )
285 {
286 vscp_omsg.flags = VSCP_VALID_MSG + 3; // three databyte
287 vscp_omsg.priority = VSCP_PRIORITY_LOW;
288 vscp_omsg.class = VSCP_CLASS1_INFORMATION;
289 vscp_omsg.type = VSCP_TYPE_INFORMATION_NODE_HEARTBEAT;
290 vscp_omsg.data[ 0 ] = 0;
291 vscp_omsg.data[ 1 ] = zone;
292 vscp_omsg.data[ 2 ] = subzone;
293 // send the message
294 vscp_sendEvent();
295 }
296 ///////////////////////////////////////////////////////////////////
297 // vscp_handleHeartbeat
298 //
299 void vscp_handleHeartbeat( void )
300 {
301 if ( ( 5 == ( vscp_imsg.flags & 0x0F ) ) &&
302 ( vscp_getSegmentCRC() != vscp_imsg.data[ 0 ] ) ) {
303 // Stored CRC are different than received
304 // We must be on a different segment
305 vscp_setSegmentCRC( vscp_imsg.data[ 0 ] );
306 // Introduce ourself in the proper way and start from the beginning
307 vscp_nickname = VSCP_ADDRESS_FREE;
308 vscp_writeNicknamePermanent( VSCP_ADDRESS_FREE );
309 vscp_node_state = VSCP_STATE_INIT;
310 }
311 }

```

```

312 ///////////////////////////////////////////////////////////////////
313 // vscp_handleSetName
314 //
315 void vscp_handleSetName( void )
316 {
317 if ( ( 2 == ( vscp_imsg.flags & 0x0F ) ) &&
318 ( vscp_nickname == vscp_imsg.data[ 0 ] ) ) {
319 // Yes, we are addressed
320 vscp_nickname = vscp_imsg.data[ 1 ];
321 vscp_writeNicknamePermanent( vscp_nickname );
322 vscp_setSegmentCRC( 0x40 );
323 }
324 }
325 ///////////////////////////////////////////////////////////////////
326 // vscp_handleDropNickname
327 //
328 void vscp_handleDropNickname( void )
329 {
330 if ( ( 1 == ( vscp_imsg.flags & 0x0F ) ) &&
331 ( vscp_nickname == vscp_imsg.data[ 0 ] ) ) {
332 // Yes, we are addressed
333 vscp_nickname = VSCP_ADDRESS_FREE;
334 vscp_writeNicknamePermanent( VSCP_ADDRESS_FREE );
335 vscp_init();
336 }
337 }
338 ///////////////////////////////////////////////////////////////////
339 // vscp_newNodeOnline
340 //
341 void vscp_newNodeOnline( void )
342 {
343 if ( ( 1 == ( vscp_imsg.flags & 0x0F ) ) &&
344 ( vscp_nickname == vscp_imsg.data[ 0 ] ) ) {
345 // This is a probe which use our nickname
346 // we have to respond to tell the new node that
347 // this nickname is in use
348 vscp_omsg.flags = VSCP_VALID_MSG;
349 vscp_omsg.priority = VSCP_PRIORITY_HIGH;
350 vscp_omsg.class = VSCP_CLASS1_PROTOCOL;
351 vscp_omsg.type = VSCP_TYPE_PROTOCOL_PROBE_ACK;
352 vscp_sendEvent();
353 }
354 }
355 ///////////////////////////////////////////////////////////////////
356 // vscp_doOneSecondWork
357 //
358 void vscp_doOneSecondWork( void )
359 {
360 if ( vscp_second > 59 ) {
361 vscp_second = 0;
362 vscp_minute++;
363 // send periodic heartbeat
364 if ( VSCP_STATE_ACTIVE == vscp_node_state ) {
365 vscp_sendHeartBeat( vscp_getZone(),
366 vscp_getSubzone() );
367 }
368 }
369 if ( vscp_minute > 59 ) {
370 vscp_minute = 0;
371 vscp_hour++;

```

```

372 }
373 if ( vscp_hour > 23 ) vscp_hour = 0;
374 if ( VSCP_STATE_ACTIVE == vscp_node_state ) {
375 vscp_guid_reset++;
376 if ( (vscp_guid_reset & 0x0F) >= 2 ) {
377 vscp_guid_reset = 0;
378 }
379 }
380 else {
381 }
382 }
383 ///////////////////////////////////////////////////////////////////
384 // vscp_readRegister
385 ///////////////////////////////////////////////////////////////////
386 uint8_t vscp_readRegister( uint8_t reg )
387 {
388 if ( reg >= 0x80 ) {
389 return vscp_readStdReg( reg );
390 }
391 else {
392 return vscp_readAppReg( reg );
393 }
394 }
395 ///////////////////////////////////////////////////////////////////
396 // vscp_readStdReg
397 ///////////////////////////////////////////////////////////////////
398 uint8_t vscp_readStdReg( uint8_t reg )
399 {
400 uint8_t rv;
401 if ( VSCP_REG_ALARMSTATUS == vscp_imsg.data[ 1 ] ) {
402 // * * * Read alarm status register * * *
403 rv = vscp_alarmstatus;
404 vscp_alarmstatus = 0x00; // Reset alarm status
405 }
406 else if ( VSCP_REG_VSCP_MAJOR_VERSION == vscp_imsg.data[ 1 ] ) {
407 // * * * VSCP Protocol Major Version * * *
408 rv = VSCP_MAJOR_VERSION;
409 }
410 else if ( VSCP_REG_VSCP_MINOR_VERSION == vscp_imsg.data[ 1 ] ) {
411 // * * * VSCP Protocol Minor Version * * *
412 rv = VSCP_MINOR_VERSION;
413 }
414 else if ( VSCP_REG_NODE_CONTROL == vscp_imsg.data[ 1 ] ) {
415 // * * * Reserved * * *
416 rv = 0;
417 }
418 else if ( VSCP_REG_FIRMWARE_MAJOR_VERSION == vscp_imsg.data[ 1 ] ) {
419 // * * * Get firmware Major version * * *
420 rv = vscp_getMajorVersion();
421 }
422 else if ( VSCP_REG_FIRMWARE_MINOR_VERSION == vscp_imsg.data[ 1 ] ) {
423 // * * * Get firmware Minor version * * *
424 rv = vscp_getMinorVersion();
425 }
426 else if ( VSCP_REG_FIRMWARE_SUB_MINOR_VERSION == vscp_imsg.data[ 1 ] ) {
427 // * * * Get firmware Sub Minor version * * *
428 rv = vscp_getSubMinorVersion();
429 }
430 else if ( vscp_imsg.data[ 1 ] < VSCP_REG_MANUFACTUR_ID0 ) {
431 // * * * Read from persistent locations * * *
432 rv = vscp_getUserID( vscp_imsg.data[ 1 ] - VSCP_REG_USERID0 );
433 }
434 else if ( ( vscp_imsg.data[ 1 ] > VSCP_REG_USERID4 ) &&
435 ( vscp_imsg.data[ 1 ] < VSCP_REG_NICKNAME_ID ) ) {

```

```

436 // * * * Manufacturer ID information * * *
437 rv = vscp_getManufacturerId( vscp_imsg.data[ 1 ] -
438     VSCP_REG_MANUFACTUR_ID0 );
439 } else if ( VSCP_REG_NICKNAME_ID == vscp_imsg.data[ 1 ] ) {
440 // * * * nickname-ID * * *
441 rv = vscp_nickname;
442 }
443 else if ( VSCP_REG_PAGE_SELECT_LSB == vscp_imsg.data[ 1 ] ) {
444 // * * * Page select LSB * * *
445 rv = ( vscp_page_select & 0xFF );
446 }
447 else if ( VSCP_REG_PAGE_SELECT_MSB == vscp_imsg.data[ 1 ] ) {
448 // * * * Page select MSB * * *
449 rv = ( vscp_page_select >> 8 ) & 0xFF;
450 }
451 else if ( VSCP_REG_BOOT_LOADER_ALGORITHM == vscp_imsg.data[ 1 ] ) {
452 // * * * Boot loader algorithm * * *
453 rv = vscp_getBootLoaderAlgorithm();
454 }
455 else if ( VSCP_REG_BUFFER_SIZE == vscp_imsg.data[ 1 ] ) {
456 // * * * Buffer size * * *
457 rv = vscp_getBufferSize();
458 }
459 else if ( VSCP_REG_PAGES_USED == vscp_imsg.data[ 1 ] ) {
460 // * * * Register Pages Used * * *
461 rv = vscp_getRegisterPagesUsed();
462 }
463 else if ( ( vscp_imsg.data[ 1 ] > ( VSCP_REG_GUID - 1 ) ) &&
464 ( vscp_imsg.data[ 1 ] < VSCP_REG_DEVICE_URL ) ) {
465 // * * * GUID * * *
466 rv = vscp_getGUID( vscp_imsg.data[ 1 ] - VSCP_REG_GUID );
467 }
468 else {
469 // * * * The device URL * * *
470 rv = vscp_getMDF_URL( vscp_imsg.data[ 1 ] - VSCP_REG_DEVICE_URL );
471 }
472 return rv;
473 }
474 ///////////////////////////////////////////////////////////////////
475 // vscp_writeRegister
476 //
477 uint8_t vscp_writeRegister( uint8_t reg, uint8_t value )
478 {
479 if ( reg >= 0x80 ) {
480 return vscp_writeStdReg( reg, value );
481 }
482 else {
483 return vscp_writeAppReg( reg, value );
484 }
485 }
486 ///////////////////////////////////////////////////////////////////
487 // vscp_writeStdReg
488 //
489 uint8_t vscp_writeStdReg( uint8_t reg, uint8_t value )
490 {
491 uint8_t rv = ~value;
492 if ( ( vscp_imsg.data[ 1 ] > ( VSCP_REG_VSCP_MINOR_VERSION + 1 ) ) &&
493 ( vscp_imsg.data[ 1 ] < VSCP_REG_MANUFACTUR_ID0 ) ) {
494 // * * * User Client ID * * *
495 vscp_setUserID( ( vscp_imsg.data[ 1 ] - VSCP_REG_USERID0 ), vscp_imsg.
496     data[ 2 ] );
497 rv = vscp_getUserID( ( vscp_imsg.data[ 1 ] - VSCP_REG_USERID0 ) );
498 }

```

```

498 else if ( VSCP_REG_PAGE_SELECT_MSB == vscp_imsg.data[ 1 ] ) {
499 // * * * Page select register MSB * * *
500 vscp_page_select = ( vscp_page_select & 0xFF00 ) | ( ( uint16_t )vscp_imsg
501 .data[ 2 ] << 8 );
502 rv = ( vscp_page_select >> 8 ) & 0xFF;
503 }
504 else if ( VSCP_REG_PAGE_SELECT_LSB == vscp_imsg.data[ 1 ] ) {
505 // * * * Page select register LSB * * *
506 vscp_page_select = ( vscp_page_select & 0xFF ) | vscp_imsg.data[ 2 ];
507 rv = ( vscp_page_select & 0xFF );
508 }
509 #ifdef ENABLE_WRITE_2PROTECTED_LOCATIONS
510 // Write manufacturer ID configuration information
511 else if ( ( vscp_imsg.data[ 1 ] > VSCP_REG_USERID4 ) && ( vscp_imsg.data
512 [ 1 ] < VSCP_REG_NICKNAME_ID ) ) {
513 // page select must be 0xFFFF for writes to be possible
514 if ( ( 0xFF != ( ( vscp_page_select >> 8 ) & 0xFF ) ) || ( 0xFF != ( vscp_page_select & 0xFF ) ) ) {
515 // return complement to indicate error
516 rv = ~vscp_imsg.data[ 2 ];
517 }
518 // Write
519 vscp_setManufacturerId( vscp_imsg.data[ 1 ] - VSCP_REG_MANUFACTUR_ID0,
520 vscp_imsg.data[ 2 ] );
521 rv = vscp_getManufacturerId( vscp_imsg.data[ 1 ] -
522 VSCP_REG_MANUFACTUR_ID0 );
523 }
524 // Write GUID configuration information
525 else if ( ( vscp_imsg.data[ 1 ] > ( VSCP_REG_GUID - 1 ) ) && ( vscp_imsg
526 .data[ 1 ] < VSCP_REG_DEVICE_URL ) ) {
527 // page must be 0xFFFF for writes to be possible
528 if ( ( 0xFF != ( ( vscp_page_select >> 8 ) & 0xFF ) ) || ( 0xFF != ( vscp_page_select & 0xFF ) ) ) {
529 // return complement to indicate error
530 rv = ~vscp_imsg.data[ 2 ];
531 }
532 vscp_setGUID( vscp_imsg.data[ 1 ] - VSCP_REG_GUID, vscp_imsg.data[ 2 ] )
533 ;
534 rv = vscp_getGUID( vscp_imsg.data[ 1 ] - VSCP_REG_GUID );
535 }
536 #endif
537 else {
538 // return complement to indicate error
539 rv = ~value;
540 }
541 return rv;
542 }
543 ///////////////////////////////////////////////////////////////////
544 // vscp_writeStdReg
545 //
546 void vscp_handleProtocolEvent( void )
547 {
548 if ( VSCP_CLASS1_PROTOCOL == vscp_imsg.class ) {
549 switch( vscp_imsg.type ) {
550 case VSCP_TYPE_PROTOCOL_SEGCTRL_HEARTBEAT:
551 vscp_handleHeartbeat();
552 break;
553 case VSCP_TYPE_PROTOCOL_NEW_NODE_ONLINE:
554 vscp_newNodeOnline();
555 break;
556 case VSCP_TYPE_PROTOCOL_SET_NICKNAME:
557 vscp_handleSetNickname();

```

```

558     break;
559   case VSCP_TYPE_PROTOCOL_DROP_NICKNAME:
560     vscp_handleDropNickname();
561     break;
562   case VSCP_TYPE_PROTOCOL_READ_REGISTER:
563     if ( ( 2 == ( vscp_imsg.flags & 0x0F ) ) &&
564       ( vscp_nickname == vscp_imsg.data[ 0 ] ) ) {
565       if ( vscp_imsg.data[ 1 ] < 0x80 ) {
566         vscp_omsg.priority = VSCP_PRIORITY_MEDIUM;
567         vscp_omsg.flags = VSCP_VALID_MSG + 2;
568         vscp_omsg.class = VSCP_CLASS1_PROTOCOL;
569         vscp_omsg.type = VSCP_TYPE_PROTOCOL_RW_RESPONSE;
570         // Register to read
571         vscp_omsg.data[ 0 ] = vscp_imsg.data[ 1 ];
572         // Read application specific register
573         vscp_omsg.data[ 1 ] = vscp_readAppReg( vscp_imsg.data[ 1 ] );
574         // Send reply data
575         vscp_sendEvent();
576       }
577     } else {
578       vscp_omsg.priority = VSCP_PRIORITY_NORMAL;
579       vscp_omsg.flags = VSCP_VALID_MSG + 2;
580       vscp_omsg.class = VSCP_CLASS1_PROTOCOL;
581       vscp_omsg.type = VSCP_TYPE_PROTOCOL_RW_RESPONSE;
582       // Register to read
583       vscp_omsg.data[ 0 ] = vscp_imsg.data[ 1 ];
584       // Read VSCP register
585       vscp_omsg.data[ 1 ] =
586       vscp_readStdReg( vscp_imsg.data[ 1 ] );
587       // Send event
588       vscp_sendEvent();
589     }
590   }
591   break;
592 case VSCP_TYPE_PROTOCOL_WRITE_REGISTER:
593   if ( ( 3 == ( vscp_imsg.flags & 0x0F ) ) &&
594     ( vscp_nickname == vscp_imsg.data[ 0 ] ) ) {
595     if ( vscp_imsg.data[ 1 ] < 0x80 ) {
596       vscp_omsg.priority = VSCP_PRIORITY_MEDIUM;
597       vscp_omsg.flags = VSCP_VALID_MSG + 2;
598       vscp_omsg.class = VSCP_CLASS1_PROTOCOL;
599       vscp_omsg.type = VSCP_TYPE_PROTOCOL_RW_RESPONSE;
600       // Register read
601       vscp_omsg.data[ 0 ] = vscp_imsg.data[ 1 ];
602       // Write application specific register
603       vscp_omsg.data[ 1 ] =
604       vscp_writeAppReg( vscp_imsg.data[ 1 ], vscp_imsg.data[ 2 ] );
605       // Send reply
606       vscp_sendEvent();
607     }
608   } else {
609     vscp_omsg.priority = VSCP_PRIORITY_MEDIUM;
610     vscp_omsg.flags = VSCP_VALID_MSG + 2;
611     vscp_omsg.class = VSCP_CLASS1_PROTOCOL;
612     vscp_omsg.type = VSCP_TYPE_PROTOCOL_RW_RESPONSE;
613     vscp_omsg.data[ 0 ] = vscp_nickname;
614     // Register read
615     vscp_omsg.data[ 0 ] = vscp_imsg.data[ 1 ];
616     // Write VSCP register
617     vscp_omsg.data[ 1 ] =
618     vscp_writeStdReg( vscp_imsg.data[ 1 ], vscp_imsg.data[ 2 ] );
619     // Write event
620     vscp_sendEvent();
621   }
622 }
623 break;
624 case VSCP_TYPE_PROTOCOL_ENTER_BOOT_LOADER:
625   if ( ( vscp_nickname == vscp_imsg.data[ 0 ] ) &&

```

```

626 ( 1 == vscp_imsg.data[ 1 ] ) &&
627 ( vscp_getGUID( 0 ) == vscp_imsg.data[ 2 ] ) &&
628 ( vscp_getGUID( 3 ) == vscp_imsg.data[ 3 ] ) &&
629 ( vscp_getGUID( 5 ) == vscp_imsg.data[ 4 ] ) &&
630 ( vscp_getGUID( 7 ) == vscp_imsg.data[ 5 ] ) &&
631 ( ( vscp_page_select >> 8 ) == vscp_imsg.data[ 6 ] ) &&
632 ( ( vscp_page_select & 0xFF ) == vscp_imsg.data[ 7 ] ) ) {
633 vscp_goBootloaderMode();
634 }
635 break;
636 case VSCP_TYPE_PROTOCOL_RESET_DEVICE:
637 switch ( vscp_imsg.data[ 0 ] >> 4 ) {
638 case 0:
639 if ( ( vscp_getGUID( 0 ) == vscp_imsg.data[ 1 ] ) &&
640 ( vscp_getGUID( 1 ) == vscp_imsg.data[ 2 ] ) &&
641 ( vscp_getGUID( 2 ) == vscp_imsg.data[ 3 ] ) &&
642 ( vscp_getGUID( 3 ) == vscp_imsg.data[ 4 ] ) ) {
643 vscp_guid_reset |= 0x10;
644 }
645 break;
646 case 1:
647 if ( ( vscp_getGUID( 4 ) == vscp_imsg.data[ 1 ] ) &&
648 ( vscp_getGUID( 5 ) == vscp_imsg.data[ 2 ] ) &&
649 ( vscp_getGUID( 6 ) == vscp_imsg.data[ 3 ] ) &&
650 ( vscp_getGUID( 7 ) == vscp_imsg.data[ 4 ] ) ) {
651 vscp_guid_reset |= 0x20;
652 }
653 break;
654 case 2:
655 if ( ( vscp_getGUID( 8 ) == vscp_imsg.data[ 1 ] ) &&
656 ( vscp_getGUID( 9 ) == vscp_imsg.data[ 2 ] ) &&
657 ( vscp_getGUID( 10 ) == vscp_imsg.data[ 3 ] ) &&
658 ( vscp_getGUID( 11 ) == vscp_imsg.data[ 4 ] ) ) {
659 vscp_guid_reset |= 0x40;
660 }
661 break;
662 case 3:
663 if ( ( vscp_getGUID( 12 ) == vscp_imsg.data[ 1 ] ) &&
664 ( vscp_getGUID( 13 ) == vscp_imsg.data[ 2 ] ) &&
665 ( vscp_getGUID( 14 ) == vscp_imsg.data[ 3 ] ) &&
666 ( vscp_getGUID( 15 ) == vscp_imsg.data[ 4 ] ) ) {
667 vscp_guid_reset |= 0x80;
668 }
669 break;
670 default:
671 vscp_guid_reset = 0;
672 break;
673 }
674 if ( 0xF0 == ( vscp_guid_reset & 0xF0 ) ) {
675 // Do a reset
676 vscp_init();
677 }
678 break;
679 case VSCP_TYPE_PROTOCOL_PAGE_READ:
680 if ( vscp_nickname == vscp_imsg.data[ 0 ] ) {
681 uint8_t i;
682 uint8_t pos = 0;
683 uint8_t bSent = TRUE;
684 for ( i = vscp_imsg.data[ 1 ];
685 i < ( vscp_imsg.data[ 1 ] + vscp_imsg.data[ 2 ] + 1 );
686 i++ ) {
687 vscp_omsg.flags = VSCP_VALID_MSG + ( pos % 7 ) + 1;
688 vscp_omsg.data[ ( pos % 7 ) + 1 ] =
689 vscp_readAppReg( i );
690 bSent = FALSE;
691 if ( pos && ( 0 == ( pos % 7 ) ) ) {
692 vscp_omsg.priority = VSCP_PRIORITY_NORMAL;
693 vscp_omsg.class = VSCP_CLASS1_PROTOCOL;

```

```

694 vscp_omsg.type = VSCP_TYPE_PROTOCOL_RW_PAGE_RESPONSE;
695 vscp_omsg.data[ 0 ] = pos/7; // index
696 vscp_omsg.flags = VSCP_VALID_MSG + 7; // Count = 7
697 // send the event
698 vscp_sendEvent();
699 bSent = TRUE;
700 }
701 pos++;
702 }
703 // Send any pending event
704 if ( bSent ) {
705 vscp_omsg.priority = VSCP_PRIORITY_NORMAL;
706 vscp_omsg.class = VSCP_CLASS1_PROTOCOL;
707 vscp_omsg.type = VSCP_TYPE_PROTOCOL_RW_PAGE_RESPONSE;
708 vscp_omsg.data[ 0 ] = pos/7; // index
709 // send the event
710 vscp_sendEvent();
711 }
712 }
713 break;
714 case VSCP_TYPE_PROTOCOL_PAGE_WRITE:
715 if ( vscp_nickname == vscp_imsg.data[ 0 ] ) {
716 uint8_t i;
717 uint8_t pos = vscp_imsg.data[ 1 ];
718 for ( i = 0; ( i < ( vscp_imsg.flags & 0x0F ) - 2 ); i++ ) {
719 // Write VSCP register
720 vscp_omsg.data[ i + 1 ] =
721 vscp_writeStdReg( vscp_imsg.data[ 1 ] + i,
722 vscp_imsg.data[ 2 + i ] );
723 }
724 vscp_omsg.priority = VSCP_PRIORITY_NORMAL;
725 vscp_omsg.class = VSCP_CLASS1_PROTOCOL;
726 vscp_omsg.type = VSCP_TYPE_PROTOCOL_RW_PAGE_RESPONSE;
727 vscp_omsg.data[ 0 ] = 0; // index
728 vscp_omsg.flags = VSCP_VALID_MSG +
729 ( vscp_imsg.flags & 0x0F ) - 2;
730 // send the event
731 vscp_sendEvent();
732 }
733 break;
734 case VSCP_TYPE_PROTOCOL_INCREMENT_REGISTER:
735 if ( vscp_nickname == vscp_imsg.data[ 0 ] ) {
736 vscp_omsg.priority = VSCP_PRIORITY_NORMAL;
737 vscp_omsg.flags = VSCP_VALID_MSG + 2;
738 vscp_omsg.class = VSCP_CLASS1_PROTOCOL;
739 vscp_omsg.type = VSCP_TYPE_PROTOCOL_RW_RESPONSE;
740 vscp_omsg.data[ 0 ] = vscp_imsg.data[ 1 ];
741 vscp_omsg.data[ 1 ] = vscp_writeAppReg(
742 vscp_imsg.data[ 1 ],
743 vscp_readAppReg( vscp_imsg.data[ 1 ] ) + 1 );
744 // send the event
745 vscp_sendEvent();
746 }
747 break;
748 case VSCP_TYPE_PROTOCOL_DECREMENT_REGISTER:
749 if ( vscp_nickname == vscp_imsg.data[ 0 ] ) {
750 vscp_omsg.priority = VSCP_PRIORITY_NORMAL;
751 vscp_omsg.flags = VSCP_VALID_MSG + 2;
752 vscp_omsg.class = VSCP_CLASS1_PROTOCOL;
753 vscp_omsg.type = VSCP_TYPE_PROTOCOL_RW_RESPONSE;
754 vscp_omsg.data[ 0 ] = vscp_imsg.data[ 1 ];
755 vscp_omsg.data[ 1 ] = vscp_writeAppReg(
756 vscp_imsg.data[ 1 ],
757 vscp_readAppReg( vscp_imsg.data[ 1 ] ) - 1 );
758 // send the message
759 vscp_sendEvent();
760 }
761 break;

```

```

762 case VSCP_TYPE_PROTOCOL_WHO_IS_THERE:
763 if ( ( vscp_nickname == vscp_imsg.data[ 0 ] ) ||
764 ( 0xFF == vscp_imsg.data[ 0 ] ) ) {
765 uint8_t i,j;
766 // Send data
767 for ( j = 0; j < 7; j++ ) {
768 vscp_omsg.priority = VSCP_PRIORITY_NORMAL;
769 vscp_omsg.flags = VSCP_VALID_MSG + 8;
770 vscp_omsg.class = VSCP_CLASSI_PROTOCOL;
771 vscp_omsg.type = VSCP_TYPE_PROTOCOL_WHO_IS_THERE_RESPONSE;
772 vscp_omsg.data[ 0 ] = j;
773 for ( i = 1; i < 8; i++ ) {
774 vscp_omsg.data[ i ] = vscp_readStdReg( 0xD0 + 7*j + ( i - 1 ) );
775 }
776 // Correct read error for last run
777 if ( 6 == j ) {
778 vscp_omsg.data[ 7 ] = 0;
779 }
780 // send the event
781 vscp_sendEvent();
782 }
783 }
784 break;
785 case VSCP_TYPE_PROTOCOL_GET_MATRIX_INFO:
786 if ( vscp_nickname == vscp_imsg.data[ 0 ] ) {
787 vscp_omsg.priority = VSCP_PRIORITY_NORMAL;
788 vscp_omsg.flags = VSCP_VALID_MSG + 7;
789 vscp_omsg.class = VSCP_CLASSI_PROTOCOL;
790 vscp_omsg.type = VSCP_TYPE_PROTOCOL_GET_MATRIX_INFO_RESPONSE;
791 vscp_getMatrixInfo( (char *)vscp_omsg.data );
792 // send the event
793 vscp_sendEvent();
794 }
795 break;
796 case VSCP_TYPE_PROTOCOL_GET_EMBEDDED_MDF:
797 vscp_getEmbeddedMdfInfo();
798 break;
799 case VSCP_TYPE_PROTOCOL_EXTENDED_PAGE_READ:
800 if ( vscp_nickname == vscp_imsg.data[ 0 ] ) {
801 uint8_t i;
802 uint16_t page_save;
803 uint8_t page_msb = vscp_imsg.data[ 1 ];
804 uint8_t page_lsb = vscp_imsg.data[ 2 ];
805 // Save the current page
806 page_save = vscp_page_select;
807 // Check for valid count
808 if ( vscp_imsg.data[ 3 ] > 6 ) vscp_imsg.data[ 3 ] = 6;
809 vscp_omsg.priority = VSCP_PRIORITY_NORMAL;
810 vscp_omsg.flags = VSCP_VALID_MSG + 3 + vscp_imsg.data[ 3 ];
811 vscp_omsg.class = VSCP_CLASSI_PROTOCOL;
812 vscp_omsg.type = VSCP_TYPE_PROTOCOL_EXTENDED_PAGE_RESPONSE;
813 for ( i=vscp_imsg.data[ 2 ]; i < ( vscp_imsg.data[ 2 ] + vscp_imsg.data[ 3 ] ); i++ ) {
814 vscp_omsg.data[ 3 + ( i - vscp_imsg.data[ 2 ] ) ] =
815 vscp_readRegister( i );
816 }
817 // Restore the saved page
818 vscp_page_select = page_save;
819 // send the event
820 vscp_sendEvent();
821 }
822 break;
823 case VSCP_TYPE_PROTOCOL_EXTENDED_PAGE_WRITE:
824 if ( vscp_nickname == vscp_imsg.data[ 0 ] ) {
825 uint8_t i;
826 uint16_t page_save;
827 uint8_t page_msb = vscp_imsg.data[ 1 ];

```

```

830     uint8_t page_lsb = vscp_imsg.data[ 2 ];
831     // Save the current page
832     page_save = vscp_page_select;
833     vscp_omsg.priority = VSCP_PRIORITY_NORMAL;
834     vscp_omsg.flags = VSCP_VALID_MSG + 3 + vscp_imsg.data[ 3 ];
835     vscp_omsg.class = VSCP_CLASS1_PROTOCOL;
836     vscp_omsg.type = VSCP_TYPE_PROTOCOL_EXTENDED_PAGE_RESPONSE;
837     for ( i=vscp_imsg.data[ 2 ]; i < ( vscp_imsg.data[ 2 ] + vscp_imsg.data[ 3 ] ); i++ ) {
838         vscp_omsg.data[ 3 +
839             ( i - vscp_imsg.data[ 2 ] ) ] =
840         vscp_writeRegister( i,
841             vscp_imsg.data[ 4 +
842                 ( i - vscp_imsg.data[ 2 ] ) ] );
843     }
844     // Restore the saved page
845     vscp_page_select = page_save;
846     // send the event
847     vscp_sendEvent();
848 }
849 break;
850 default:
851     // Do work load
852     break;
853 } // switch
854 } // CLASS1.PROTOCOL event
855 }
856 }
857 }
858 ///////////////////////////////////////////////////////////////////
859 // vscp_sendEvent
860 //
861 int8_t vscp_sendEvent( void )
862 {
863     int8_t rv;
864     if ( !( rv = sendVSCPFrame( vscp_omsg.class,
865         vscp_omsg.type,
866         vscp_nickname,
867         vscp_omsg.priority,
868         ( vscp_omsg.flags & 0x0F ),
869         vscp_omsg.data ) ) ) {
870         vscp_errorcnt++;
871     }
872     return rv;
873 }
874 //
875 // vscp_getEvent
876 //
877 int8_t vscp_getEvent( void )
878 {
879     int8_t rv;
880     // Don't read in new message if there already is a message
881     // in the input buffer. We return TRUE though to indicate there is
882     // a valid event.
883     if ( vscp_imsg.flags & VSCP_VALID_MSG ) return TRUE;
884     if ( ( rv = getVSCPFrame( &vscp_imsg.class,
885         &vscp_imsg.type,
886         &vscp_imsg.oaddr,
887         &vscp_imsg.priority,
888         &vscp_imsg.flags,
889         vscp_imsg.data ) ) ) {
890         vscp_imsg.flags |= VSCP_VALID_MSG;
891     }
892     return rv;
893 }

```

Part VII

CANAL - CAN Abstraction Layer

Abstract

CANAL is originally designed to be a hardware interface to CAN hardware. Now it is mostly used as a least common denominator communication interface for VSCP low end devices. CANAL is a very simplistic and pragmatic interface and therefore also very easy to implement.

Version 1.14 (2008-10-28)
Åke Hedmanakhe@eurosource.se
<http://www.vscp.org>

26 Introduction to CANAL? What is it?

CANAL (CAN Abstraction Layer) is a very simple approach to interfacing a CAN card or some other CAN hardware. It consists mostly of open, close, read, write and filter/mask handling. If you need to do more this is not the model for you and we recommend CanPie <http://sourceforge.net/projects/canpie> or VCA (Virtual CAN API) used in the OCERA project above <http://www.ocera.org> which are much more advanced, and capable implementations.

CANAL is the lower layer for a home automation protocol VSCP (Very Simple Control Protocol) which uses (or rather can use) CAN based nodes. It is built as two separate solutions just to make the CANAL stuff useful also for other CAN tasks.

On top of this some drivers have been built. The can232 driver for the Lawicel adapter is a typical example but there will be others for , IXXAT, Zanthic Technologies Inc., Ferraris Elettronica, Vector and OMK (OCERA project) and others. There will also be some simulation devices using RS-232, UDP and TCP. The good thing here is that you now can use one programmatic interface to them all. You can therefore build applications that work for different hardware.

On top of this is a daemon/service built. This works much as an Ethernet hub. On one side there is a canal interface for clients and on the other side is a CANAL interface for drivers. You tell it which drivers to load at start up. A client now can connect to the daemon and send a event and it will be sent to all other clients and to all devices. The typical use is for simulation. Develop your nodes as clients in the early development stages and debug and simulate the behavior of your control situation. Deploy piece by piece (client by client) to the real architecture.

If that is to much use it as a logger for your system. The VSCP Works application let you view the traffic on your system. The footprint of the daemon is light and small.

The license for the daemon and the applications are GNU GPL meaning source will be available. For drivers, classes and interfaces the GNU LGPL

license is used. You can therefore use the later more freely in “protected” applications.

27 CANAL-API Specification

27.1 long CanalOpen(const char *pConfigStr, unsigned long flags)

Opens a CAN channel.

27.1.1 Params

pConfigStr Physical device to connect to. This is the place to add device specific parameters and filters/masks. This is a text string. It can be a name, some parameters or whatever the interface creator chooses.

flags Device specific flags with a meaning defined by the interface creator.

27.1.2 Returns

Handle for open physical interface or $<= 0$ on error. For an interface where there is only one channel the handle has no special meaning and can only be looked upon as a status return parameter.

27.2 int CanalClose(long handle)

Close the channel and free all allocated resources associated with the channel.

27.2.1 Params

handle Handle for open physical interface.

Returns TRUE on success or FALSE if failure.

27.3 unsigned long CanalGetLevel(long handle)

27.3.1 Params

handle

Handle for open physical interface.

>Returns

Returns the canal level this interface can handle. An error is indicated by a zero return value. At the moment the following levels is defined

CANAL_LEVEL_STANDARD	A standard CANAL driver as of this specification.
CANAL_LEVELUSES_TCPIP	This driver does not respond to any of the methods, even if they must be implemented. The driver will instead use the TCP/IP interface of the VSCP daemon for its data transfer. This driver type is called a VSCP Level II driver because it is constructed to work just with VSCP Level II events which does not fit in standard CAN frames. This type of driver does not need any buffers as no data is exchanged through the interface

Table 11: CANAL Levels

27.4 int CanalSend(long handle, const PCANALMSG pCanMsg)

27.4.1 Params

handle Handle for open physical interface.

pCanMsg Message to send.

27.4.2 Returns

CANAL_ERROR_SUCCESS on success or an error code on failure.

27.5 int CanalBlockingSend(long handle, const PCANALMSG pCanMsg, unsigned long timeout)

27.5.1 Params

handle Handle for open physical interface.

pCanMsg Message to send.

timeout Timeout in milliseconds. 0 to wait forever.

27.5.2 Returns

CANAL_ERROR_SUCCESS, CANAL_ERROR_TIMEOUT on timeout on success or an error code on failure.

27.6 int CanalReceive(long handle, PCANALMSG pCanMsg)

27.6.1 Params

handle Handle for open physical interface.

pCanMsg Pointer to message to receive.

27.6.2 Returns

CANAL_ERROR_SUCCESS on success or an error code on failure. *Note that if bit 2 in flags of the CANAL message is set and error status message is returned from the adapter.*

27.7 int CanalBlockingReceive(long handle, PCANALMSG pCanMsg, unsigned long timeout)

27.7.1 Params

handle Handle for open physical interface.

pCanMsg Message to receive.

timeout Timeout in milliseconds. 0 to wait forever.

27.7.2 Returns

CANAL_ERROR_SUCCESS on success, CANAL_ERROR_TIMEOUT on timeout or an error code on failure. *Note that if bit 2 in flags of the CANAL message is set and error status message is returned from the adapter.*

27.8 int CanalDataAvailable(long handle)

Check if there is data available in the input queue for this channel that can be fetched with CanalReceive.

27.8.1 Params

handle – Handle for open physical interface.

27.8.2 Returns

Number of frames available to read.

27.9 int CanalGetStatus(long handle, PCANALSTATUS pCanStatus)

Returns a structure that gives some information about the state of the channel. How the information is interpreted is up to the interface designer. Typical use is for extended error information.

27.9.1 Params

handle Handle for open physical interface.

pCanStatus Status.

27.9.2 Returns

CANAL_ERROR_SUCCESS on success or an error code on failure.

27.10 int CanalGetStatistics (long handle, PCANALSTATISTICS pCanalStatistics)

Return some statistics about the interface. If not implemented for an interface FALSE should always be returned.

27.10.1 Params

handle Handle for open physical interface.

pCanalStatistics Statistics for the interface.

27.10.2 Returns

CANAL_ERROR_SUCCESS on success or an error code on failure.

27.11 int CanalSetFilter (long handle, unsigned long filter)

Set the filter for a channel. There is only one filter available. The CanalOpen call can be used to set multiple filters. If not implemented FALSE should always be returned.

Enable filter settings in the open call if possible. If available in the open method this method can be left unimplemented returning false.

27.11.1 Params

handle Handle for open physical interface.

filter filter for the interface.

27.11.2 Returns

CANAL_ERROR_SUCCESS on success or an error code on failure.

27.12 int CanalSetMask (long handle, unsigned long mask)

Set the mask for a channel. There is only one mask available for a channel. The CanalOpen call can be used to set multiple masks. If not implemented FALSE should always be returned.

Enable mask settings in the open call if possible. If available in the open method this method can be left unimplemented returning false.

27.12.1 Params

handle Handle for open physical interface.

mask filter for the interface.

27.12.2 Returns

CANAL_ERROR_SUCCESS on success or an error code on failure.

27.13 int CanalSetBaudrate (long handle, unsigned long baudrate)

Set the bus speed for a channel. The CanalOpen call may be a better place to do this. If not implemented FALSE should always be returned.

Enable baudrate settings in the open call if possible. If available in the open method this method can be left unimplemented returning false.

27.13.1 Params

handle Handle for open physical interface.

baudrate The bus speed for the interface.

27.13.2 Returns

CANAL_ERROR_SUCCESS on success or an error code on failure.

27.14 unsigned long CanalGetVersion (void)

Get the Canal version. This is the version derived from the document that has been used to implement the interface. Version is located on the front page of the document.

27.14.1 Returns

Canal version expressed as an unsigned long.

27.15 unsigned long CanalGetDllVersion (void)

Get the version of the interface implementation. This is the version of the code designed to implement Canal for some specific hardware.

27.15.1 Returns

Canal dll version expressed as an unsigned long.

27.16 const char * CanalGetVendorString (void)

Get a pointer to a null terminated vendor string for the maker of the interface implementation. This is a string that identifies the constructor of the interface implementation and can hold copyright and other valid information.

27.16.1 Returns

Pointer to a vendor string.

27.17 const char * CanalGetDriverInfo(void)

This call returns a documentation object in XML form of the configuration string for the driver. This string can be used to help users to enter the configuration data in an application which allows for this.

```
<?xml version = "1.0" encoding = "UTF-8" ?>
<!-- Version 0.0.2      2009-11-17
     "string"      Text string .
     "bool"        1 bit number specified as true or false .
     "char"        8 bit number. Hexadecimal if it starts with "0x" else decimal .
     "uchar"       Unsigned 8 bit number. Hexadecimal if it starts with "0x" else decimal .
     "short"       16 bit signed number. Hexadecimal if it starts with "0x" else decimal .
     "ushort"      16 bit unsigned number. Hexadecimal if it starts with "0x" else decimal .
     "int"         32 bit signed number. Hexadecimal if it starts with "0x" else decimal .
     "uint"        32 bit unsigned number. Hexadecimal if it starts with "0x" else decimal .
     "long"        64 bit signed number. Hexadecimal if it starts with "0x" else decimal .
     "ulong"       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 .
-->
<canaldriver>
    <drvname>aaaaaaaa</drvname>
    <platform>WIN32|WIN64|LINUX</platform>
    <model>bbbbbb</model>
    <version>cccccc</version>
    <description lang = "en">yyyyyyyyyyyyyyyyyyyyyyyy</description>
    <!-- Site with info about the product -->
    <infourl>http://www.somewhere.com</infourl>
```

```

<!-- Information about CANAL driver maker -->
<maker>
    <name>t t t t t t t t t t t t t t t t t t</name>
    <address>
        <street>t t t t t t t t t t t t t t t t t t</street>
        <town>l l l l l l l l l l l l</town>
        <city>London</city>
        <postcode>HH1234</postcode>
        <!-- Use region or state -->
        <state></state>
        <region></region>
        <country>t t t t </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>someone@somwhere.com</email>

    <!-- One or many -->
    <web>www.somewhere.com</web> </maker>

    <devicestring>

        <!-- Option example with selectable values -->
        <option pos="0" type="string">
            <name lang="en">aaaaa</name>
            <description lang="en">yyy</description>
            <valuelist>
                <item value="CANUSB" />
                    <name lang="en">USB Adapter</name>
                    <description lang="en">yyy</description>
                </item>
                <item value="CANPCI" />
                    <name lang="en">PCI Adapter</name>
                    <description lang="en">yyy</description>
                </item>
                <item value="CAN232" />
                    <name lang="en">Serial Adapter</name>
                    <description lang="en">yyy</description>
                </item>
            </valuelist>
        </option>

        <!-- Option example with numerical value -->
        <option pos="1" type="uchar" min="0" max="4">
            <name lang="en">aaaaa</name>
            <description lang="en">yyy</description>
        </option>
    </devicestring>

    <!-- Flags part (32-bit value) of the device information -->
    <flags>
        <description lang="en">yyy</description>
        <bit pos="0">
            <name lang="en">t t t t </name>
            <description lang="en">yyy</description>
        </bit>
    </flags>

```

```

<bit pos="1" width="4">
    <!-- example for bit groups , in this case bit 1,2,3,4 -->
    <name lang="en">tttt</name>
    <description lang="en">yyy</description>
</bit>
</flags>

<!-- Status return value (32-bit value) -->
<status>
    <bit pos="0">
        <name lang="en">tttt</name>
        <description lang="en">yyy</description>
    </bit>
    <bit pos="1" width="4">
        <name lang="en">tttt</name>
        <description lang="en">yyy</description>
    </bit>
</status>

</canaldriver>

```

27.17.1 Returns

Pointer to a configuration string or NULL if no configuration string is available.

27.17.2 Params

handle Handle for open physical interface.

pDriverInfo Pointer to unsigned long that holds driver information after call.
Bit 31 - Driver support blocking calls. all other bits undefined at the moment.

27.17.3 Returns

CANAL_ERROR_SUCCESS on success or an error code on failure.

27.18 CANAL - Data Structures

27.18.1 CANMSG

This is the general message structure

unsigned long flags Flags for the package.

- Bit 0 – if set indicates that an extended identifier (29-bit id) else standard identifier (11-bit) is used.
- Bit 1 – If set indicates a RTR (Remote Transfer) frame.
- Bit 2 – If set indicates that this is an error package. The data bytes holds the error information. ID is set to zero. For format see CANAL_IDFLAG_STATUS below.
- Bit 3 – Bit 30 Reserved.

- Bit 31 – This bit can be used as a direction indicator for application software. 0 is receive and 1 is transmit.

unsigned long obid Used by the driver or higher layer protocols.

unsigned long id The 11-bit or 29 bit message ID.

unsigned char data[8] Eight bytes of data.

unsigned char count Number of data bytes 0-8

unsigned long timestamp A time stamp on the message from the driver or the interface expressed in microseconds. Can be used for relative time measurements.

27.18.2 CANALSTATUS

unsigned long channel_status Current state for CAN channel

CANAL_STATUS_NONE	0x00000000
CANAL_STATUS_BUS_OFF	0x80000000
CANAL_STATUS_BUS_WARN	0x40000000
CANAL_STATUS_PASSIVE	0x20000000
CANAL_STATUS_ACTIVE	0x10000000
CANAL_STATUS_PHY_FAULT	0x08000000
CANAL_STATUS_PHY_H	0x04000000
CANAL_STATUS_PHY_L	0x02000000
CANAL_STATUS_SLEEPING	0x01000000
CANAL_STATUS_STOPPED	0x00800000
CANAL_STATUS_RECEIVE_FIFO_FULL	0x00400000
CANAL_STATUS_TRANSMIT_FIFO_FULL	0x00200000

Bits from 16-31 are reserved, bits from 0-15 are user defined and can be defined by the driver maker.

27.18.3 PCANALSTATISTICS

This is the general statistics structure

unsigned long cntReceiveFrames Number of received frames since the channel was opened.

unsigned long cntTransmittFrames Number of frames transmitted since the channel was opened.

unsigned long cntReceiveData Number of bytes received since the channel was opened.

unsigned long cntTransmittData Number of bytes transmitted since the channel was opened.

unsigned long cntOverruns Number of overruns since the channel was opened.

unsigned long cntBusWarnings Number of bus warnings since the channel was opened.

unsigned long cntBusOff Number of bus offs since the channel was opened.

27.18.4 Error codes

Codes up to 0xFFFF are reserved, codes from 0x10000 and up are user defined. Message ID Flags

Each message has some flags set to give information about the events. The flags are defined as follow

CANAL_IDFLAG_SEND may seem strange but can be very useful for software to use as a way to distinguish between sent and received frames.

CANAL_IDFLAG_STATUS can be used by CAN controllers to report status data back to a host or an application. At the moment the following error codes are defined. All status events consist of four data bytes where the first byte tell the status code, the second is the receive error counter, the third the transmit error counter and the fourth byte is reserved and should be set to zero.

27.19 CANAL Specification History

- 2009-07-04 - Changed format for CanalGetDriverInfo and added options.
- 2008-10-28 - Fixed error in status flags. Clarified error status return.
- 2008-04-15 - Added Message ID flags description. CANAL_IDFLAG_ERROR changed to CANAL_IDFLAG_STATUS
- 2008-04-04 - Driver description format final.
- 2007-12-08 - CanalGetDriverInfo does not need a handle.
- 2007-11-22 - CANAL_ERROR_INTERNAL and CANAL_ERROR_COMMUNICATION added.
- 2007-11-13 - getDriverConfigInfo added.
- 2007-11-12 - added const in front of send messages.

Constant	Error code	Description
CANAL_ERROR_SUCCESS	0	All is OK
CANAL_ERROR_BAUDRATE	1	Baudrate error.
CANAL_ERROR_BUS_OFF	2	Bus off error
CANAL_ERROR_BUS_PASSIVE	3	Bus Passive error
CANAL_ERROR_BUS_WARNING	4	Bus warning error
CANAL_ERROR_CAN_ID	5	Invalid CAN ID
CANAL_ERROR_CAN_MESSAGE	6	Invalid CAN message
CANAL_ERROR_CHANNEL	7	Invalid channel
CANAL_ERROR_FIFO_EMPTY	8	Noting available to read. FIFO is empty
CANAL_ERROR_FIFO_FULL	9	FIFO is full
CANAL_ERROR_FIFO_SIZE	10	FIFO size error
CANAL_ERROR_FIFO_WAIT	11	Blocking for FIFO room.
CANAL_ERROR_GENERIC	12	Generic error
CANAL_ERROR_HARDWARE	13	A hardware related fault.
CANAL_ERROR_INIT_FAIL	14	Initialization failed.
CANAL_ERROR_INIT_MISSING	15	Not available
CANAL_ERROR_INIT_READY	16	Ready.
CANAL_ERROR_NOT_SUPPORTED	17	Not supported.
CANAL_ERROR_OVERRUN	18	Overrun.
CANAL_ERROR_RCV_EMPTY	19	Receive buffer empty.
CANAL_ERROR_REGISTER	20	Register value error
CANAL_ERROR_TRM_FULL	21	Full buffer.
CANAL_ERROR_ERRFRM_STUFF	22	Error frame: stuff error detected.
CANAL_ERROR_ERRFRM_FORM	23	Error frame: form error detected .
CANAL_ERROR_ERRFRM_ACK	24	Error frame: acknowledge error.
CANAL_ERROR_ERRFRM_BIT1	25	Error frame: bit 1 error.
CANAL_ERROR_ERRFRM_BIT0	26	Error frame: bit 0 error.
CANAL_ERROR_ERRFRM_CRC	27	Error frame: CRC error.
CANAL_ERROR_LIBRARY	28	Unable to load library.
CANAL_ERROR_PROCADDRESS	29	Unable get library proc address.
CANAL_ERROR_ONLY_ONE_INSTANCE	30	Only one instance allowed.
CANAL_ERROR_SUB_DRIVER	31	Problem with sub driver call.
CANAL_ERROR_TIMEOUT	32	Blocking call timeout.
CANAL_ERROR_NOT_OPEN	33	The device is not open.
CANAL_ERROR_PARAMETER	44	A parameter is invalid.
CANAL_ERROR_MEMORY	35	Memory exhausted.
CANAL_ERROR_INTERNAL	36	Some kind of internal program error.
CANAL_ERROR_COMMUNICATION	37	Some kind of communication error.

Table 12: CANAL error codes

Flag	Value	Description
CANAL_IDFLAG_STANDARD	0x00000000	Standard message ID (11-bit)
CANAL_IDFLAG_EXTENDED	0x00000001	Extended message ID (29-bit)
CANAL_IDFLAG_RTR	0x00000002	RTR-Frame
CANAL_IDFLAG_STATUS	0x00000004	This package is an error indication (data holds error code,id=0).
CANAL_IDFLAG_SEND	0x80000000	Reserved for use by application software to indicate send.

Table 13: CANAL message ID flags

Flag	Value	Description
CANAL_STATUSMSG_OK	0x00	Normal condition.
CANAL_STATUSMSG_OVERRUN	0x01	Overrun occurred when sending data to CAN bus.
CANAL_STATUSMSG_BUSLIGHT	0x02	Error counter has reached 96.
CANAL_STATUSMSG_BUSHEAVY	0x03	Error counter has reached 128.
CANAL_STATUSMSG_BUSOFF	0x04	Device is in BUSOFF. CANAL_STATUSMSG_OK is sent when returning to operational mode.
CANAL_STATUSMSG_STUFF	0x20	Stuff Error.
CANAL_STATUSMSG_FORM	0x21	Form Error.
CANAL_STATUSMSG_ACK	0x23	Ack Error.
CANAL_STATUSMSG_BIT0	0x24	Bit1 Error.
CANAL_STATUSMSG_BIT1	0x25	Bit0 Error.
CANAL_STATUSMSG_CRC	0x26	CRC Error.

Table 14: CANAL status flags

- *2007-11-01* - CANAL_LEVELUSES_TCPIP CANAL driver Level introduced for use with the VSCP Daemon.
- *2007-10-30* - CanalGetDriverInfo call added.
- *2007-05-13* - CanalOpen device set to const char * instead of incorrect char *
- *2007-03-06* - CANALSTATUS table was wrong. Corrected.
- *2007-01-14* - Handle changed from int to long.
- *2005-07-26* - CanalBlockingOpen call removed. CanalBlockingSend and CanalBlockingReceive added. Fixed returned values that where wrong.
- *2005-03-17* - CanalBlockingOpen call added.
- *2004-07-08* – Bit 31 of the canmsg flag defined as direction bit for application software use.
- *2004-07-01* – Added some clarifications on the CanalSetMask, CanalSetFilter and CanalSetBaudrate methods
- *2004-06-07* – CANALASTATUS had a typo and was called CANALSTATE. Fixed.
- *2004-06-07* – Recovery from version lost in hard disk crash and realest as version 1.00
- *2003-02-18* – Initial version.

Part VIII

Level I (CANAL) drivers

27.20 Drivers for the Windows and Linux platform

27.20.1 APOX Driver

This is a device driver for the Apox Controls (<http://www.apoxcontrols.com/usbcan.htm>) USB adapter.



CANAL Driver apoxdrv.dll(win32), apoxdrv.so(Linux)

Include library for msvc apoxdrv.lib

DriverString The driver string has the following format (note that all values can be entered in either decimal or hexadecimal form (for hex preceded with 0x)).

serial;bus-speed

serial This is the serial number for the adapter in use.

buss_speed This is the speed for the CAN bus. It can be given as

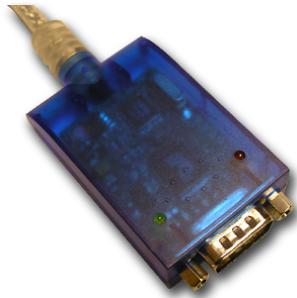
- 125 for 125Kbps
- 250 for 250Kbps
- 500 for 500Kbps
- 1000 for 1Mbps

Flags Not used set to 0.

Status return The CanalGetStatus call returns the status structure with the channel_status member having the following meaning:

- Bit 0-7 TX error counter.
- Bit 8-15 RX error counter.
- Bit 16 Overflow.
- Bit 17 RX Warning.
- Bit 18 TX Warning.
- Bit 19 TX bus passive.
- Bit 20 RX bus passive.
- Bit 21-28 Reserved.
- Bit 29 Bus Passive.
- Bit 30 Bus Warning status.
- Bit 31 Bus off status

27.20.2 Lawicel CANUSB Driver



This is a device driver for the Lawicel CANUSB adapter (<http://www.canusb.com>). It is the property of Lawicel AB and source is therefore not available. The driver can be downloaded from the above site or requested from Lawicel AB (<http://www.lawicel.se>)

Important! Note that before you use this driver the ftdi dxx libraries must be installed. Information about how to install them is available on the CANUSB site or on the ftdi site (<http://www.ftdichip.com>). We recommend using the combined VCI + DXX driver which also is available on the CANUSB site in the download section.

DriverString The driver string has the following format (note that all values can be entered in either decimal or hexadecimal form (for hex precede with 0x)).

adapter-code;bus-speed;filter;mask

To use default values just skip parameters. adapter-code is the only mandatory but can be set to “NULL” to use the first found adapter on the system. To find the adapter-code use the *adaptercheck.exe* program that is part of the CANUSB installation. A typical looking adapter code is “LWNQ06ES”.

bus_speed This is the speed for the CAN bus. It can be given as

- 5 for 5 Kbps
- 10 for 10 Kbps
- 20 for 20 Kbps
- 50 for 50 Kbps
- 100 for 100 Kbps
- 125 for 125 Kbps
- 250 for 250 Kbps
- 500 for 500 Kbps
- 800 for 800 Kbps
- 1000 for 1000 Mbs

filter Is the hardware dependent filter for this board hardware. Note that this filter work in a different way than the CANAL filter. Check the CANUSB documentation.

0x00000000 is the default. Let all messages true.

mask Is the hardware dependent mask for this board hardware. Note that this filter work in a different way than the CANAL filter. Check the CANUSB documentation.

0xFFFFFFFF is the default. Let all messages true.

example For the CANUSB adapter use

LWNQ06ES;125

which says to use the CANUSB adapter with adapter-ID LWNQ06ES and 125 kbps. You get the same result with

LWNQ06ES

which will use 125 kbps as default.

Flags

bit 0 CANUSB_FLAG_TIMESTAMP - Timestamp will be set by adapter. If not set timestamp will be set by the driver.

bit 1 CANUSB_FLAG_QUEUE_REPLACE Normally when the input queue is full new messages received are disregarded by setting this flag the first message in the queue is removed to make room for the new message.

bit 2 CANUSB_FLAG_BLOCK Can be set to make Read and Writes blocking.

::Important This bit must be set if the CANAL blocking methods should be used.

bit 3 CANUSB_FLAG_SLOW This flag can be used at slower transmission speeds where the data stream still can be high. Every effort is made to transfer the frame even if the adapter reports that the internal buffer is full. Normally a frame is trashed if the hardware buffer becomes full to promote speed.

Status return The CanalGetStatus call returns the status structure with the channel_status member having the following meaning:

- Bit 0-15 CANUSB Adapter specific.
- Bit 16-31 CANAL specified.

CANAL Driver canusbdrv.dll(win32)

Include library for msvc canusbdrv.lib

27.20.3 Lawicel CAN232 Driver



This driver interface is for the can232 adapter from Lawicel (<http://www.lawicel.com> or <http://www.can232.com>). This is a low cost CAN adapter that connects to one of the serial communication ports on a computer. The driver can handle

the adapter in both polled and non polled mode, which handled transparently to the user. It is recommended however that the following settings are made before real life use.

1. Set the baud rate for the device to 115200. You do this with the U1 command. This is the default baud rate used by this driver.
2. Set auto poll mode by issuing the X1 command.
3. Enable the time stamp by issuing the Z1 command.

CANAL Driver can232drv.dll(win32), can232drv.so(Linux)

Include library for msvc can232drv.lib

DriverString The driver string has the following format (note that all values can be entered in either decimal or hexadecimal form (for hex precede with 0x)).

comport;baudrate;mask;filter;bus-speed;btr0;btr1

comport is the serial communication port to use.(for example 1,2,3. . .).

baudrate is a valid baudrate for the serial interface (for example. 9600).

mask is the mask for the adapter. Read the Lawicel CAN232 manual on how to set this. It is not the same as for CANAL.

filter is the filter for the adapter. Read the Lawicel CAN232 manual on how to set this. It is not the same as for CANAL.

bus-speed is the speed or the CAN interface. Valid values are

- 10 10Kbps
- 20 20Kbps
- 50 50Kbps
- 100 100Kbps
- 125 125Kbps
- 250 250Kbps
- 500 500Kbps
- 800 800Kbps
- 1000 1Mbps

btr0/btr1 Optional. Instead of setting a bus-speed you can set the SJA1000 BTR0/BTR1 values directly. If both are set the bus_speed parameter is ignored.

This link can be a help for data http://www.port.de/engl/canprod/sv_req_form.html

If no device string is given COM1/ttyS0 will be used. Baud rate will be set to 115200 baud and the filter/mask to fully open. The CAN bit rate will be 500Kbps.

Flags Not used, set to 0.

Status return The CanalGetStatus call returns the status structure with the channel_status member having the following meaning:

- Bit 0-22 Reserved.
- Bit 23 Transmit buffer full.
- Bit 24 Receive Buffer Full.
- Bit 25-27 Reserved.
- Bit 28 Bus Active.
- Bit 29 Bus passive status.
- Bit 30 Bus Warning status.
- Bit 31 Bus off status.

Platforms WIN32, Linux

Example

`5;115200;0;0;1000`

Uses COM5 at 115200 with filters/masks open and with 1Mbps CAN bit rate.

`1;57600;0;0;0;0x09;0x1C`

Uses COM1 at 57600 with filters/masks open and with bit-rate set to 50Kbps using btr0/btr1

27.20.4 CCS CAN Driver

Device driver for diagnostic logging. It allows you to log CAN traffic to a text file. Several drivers can be loaded with different output files and using different filter/masks.

CANAL Driver ccsdrv.dll(win32), ccsdrv.so(Linux)

Include library for msvc ccsdrv.lib

DriverString Not used.

Flags Not used set to 0.

Status return Currently undefined.

27.20.5 Generic Serial/RS-232 interface Driver

This is a device driver that can be used to set up a serial link and exchange CAN packet data through this link.

CANAL Driver commdrv.dll(win32), commdrv.so(Linux)

Include library for msvc commdrv.lib

DriverString The driver string has the following format (note that all values can be entered in either decimal or hexadecimal form (for hex precede with 0x)).

comport;baudrate

comport is the serial communication port to use. (for example 1,2,3....).

baudrate is a valid baudrate for the serial interface (for example. 9600).

Flags Not used set to 0.

Status return Currently undefined

27.20.6 Tellstick Driver

This is a driver for the USB module available from <http://www.telldus.se>. This module can be used to control wireless switches etc from many vendors (<http://www.telldus.se/receivers.html>).

CANAL Driver tellstickdrv.dll(win32), tellstickdrv.so(Linux)

Include library for msvc tellstickdrv.lib

DriverString for WIN32/64

device-id;path-to-config

device-id This is the ID for the actual Tellstick adapter. If not present the first found adapter will be used.

path-to-config This is the path to the XML config file. If absent the driver will search for the file tellstick.xml in vscp folder the current users application data folder.

DeviceString for Linux

usb-device;device-id;path-to-config

usb-device Physical USB channel e.g. /dev/ttyUSB0. Mandatory.

device-id This is the ID for the actual tellstick adapter. If not present the first found adapter will be used.

path-to-config This is the path to the XML config file. If absent the driver will search for the file tellstick.xml in vscp folder the current users application data folder.

Flags Not used at the moment.

Status return The CanalGetStatus call returns the status structure with the channel_status member having the following meaning:

- Bit 0 - Reserved.
- Bit 1 - Reserved.
- Bit 2 - Reserved.
- Bit 3 - Reserved.
- Bit 4 - Reserved.
- Bit 5 - Reserved.
- Bit 6 - Reserved.
- Bit 7 - Reserved.
- Bit 8 - Reserved.
- Bit 9 - Reserved.
- Bit 10 - Reserved.
- Bit 11 - Reserved.
- Bit 12 - Reserved.

- Bit 13 - Reserved.
- Bit 14 - Reserved.
- Bit 15 - Reserved.
- Bit 16 - Reserved.
- Bit 17 - Reserved.
- Bit 18 - Reserved.
- Bit 19 - Reserved.
- Bit 20 - Reserved.
- Bit 21 - Reserved.
- Bit 22 - Reserved.
- Bit 23 - Reserved.
- Bit 24 - Reserved.
- Bit 25 - Reserved.
- Bit 26 - Reserved.
- Bit 27 - Reserved.
- Bit 28 - Reserved.
- Bit 29 - Reserved.
- Bit 30 - Reserved.
- Bit 31 - Reserved.

Configuration for the module is done with a XML file

```
<tellstick>

    <!-- Many devies can be set as receivers for one event -->
    <event type="on|off|dim|all" zone="n" subzone="n">
        <device protocol="NEXA |SARTANO | WAVEMAN | IKEA"
            systemcode="1-16" <!-- Needed for IKEA -->
            devicecode="1-10" <!-- Needed for IKEA -->
            dimlevel="0-10" <!-- If missing and dim event level/10 is
            dimstyle="0|1"
            housecode="A-P" <!-- Needed for NEXA, WAVEMAN -->
            channel="1-16 [SARTANO 0-255] "
```

```

<!-- Needed for NEXA, WAVEMAN and SARTANO -->
state = "0|1"
<!-- 0=off , 1=on if absent event state is used. -->
<!-- Needed for NEXA, WAVEMAN, SARTANO -->
/>
</event>
</tellstick >

```

The driver understands four events

"all" CLASS1.CONTROL Class=30, Type=2 ALL LAMPS ON/OFF

Reply: CLASS1.INFORMATION Class=30, Type=3 for ON or Type=4 for OFF for each effected device.

"on" CLASS1.CONTROL Class=30, Type=5 TURN ON

Reply: CLASS1.INFORMATION Class=30, Type=3 for ON for each effected device.

"off" CLASS1.CONTROL Class=30, Type=6 TURN OFF

Reply: CLASS1.INFORMATION Class=30, Type=4 for OFF for each effected device.

"dim" CLASS1.CONTROL Class=30, Type=20 DIM

Reply: CLASS1.INFORMATION Class=30, Type=40 for LEVEL CHANGED for each effected device.

Replies from the module are not really replies from the controlled nodes themselves as they don't issue a confirm. VSCP devices should ALWAYS reply back there state after a request to change it so here the driver take this responsibility.

27.20.7 USB2CAN CAN Driver



This is a driver for the low cost 8devices USB2CAN module available from <http://www.8devices.com>

CANAL Driver usb2can.dll(win32)

Include library for msvc usb2can.lib

DriverString WIN32/64

serial_number;baudrate;mask;filter

or if baudrate == 0

serial_number;baudrate;0;tseg2;tseg1;sjw;brp;mask;filter

note: mask and filter can be omitted

serial_numbers Serial number of the usb2can module. example: ED0000004

baudrate Baudrate for the serial interface

mask Mask to filter messages.

filter Filter to filter messages.

Flags

- 1.Enable status message
- 2.Disable auto retransmission.
- 3.Silent.
- 4.Loop-back.

Status return The CanalGetStatus call returns the status structure with the channel_status member having the following meaning:

- Bit 0 - TX Error Counter.
- Bit 1 - TX Error Counter.
- Bit 2 - TX Error Counter.
- Bit 3 - TX Error Counter.
- Bit 4 - TX Error Counter.
- Bit 5 - TX Error Counter.
- Bit 6 - TX Error Counter.
- Bit 7 - TX Error Counter.

- Bit 8 - RX Error Counter.
- Bit 9 - RX Error Counter.
- Bit 10 - RX Error Counter.
- Bit 11 - RX Error Counter.
- Bit 12 - RX Error Counter.
- Bit 13 - RX Error Counter.
- Bit 14 - RX Error Counter.
- Bit 15 - RX Error Counter.
- Bit 16 - Overflow.
- Bit 17 - RX Warning.
- Bit 18 - TX Warning.
- Bit 19 - TX bus passive.
- Bit 20 - RX bus passive.
- Bit 21 - Reserved.
- Bit 22 - Reserved.
- Bit 23 - Reserved.
- Bit 24 - Reserved.
- Bit 25 - Reserved.
- Bit 26 - Reserved.
- Bit 27 - Reserved.
- Bit 28 - Reserved.
- Bit 29 - Bus Passive.
- Bit 30 - Bus Warning status
- Bit 31 - Bus off status

27.20.8 CAN4VSCP Driver

This is a driver for the low cost CAN4VSCP-RS-232 module. The Module is described http://www.grodansparadis.com/can4vscp_rs232/can4vscp_rs232.html where there also is a description of its usage with VSCP Works and the VSCP daemon.



CANAL Driver can4vscdrv.dll(win32/64), can4vscdrv.so(Linux)

Include library for msvc can4vscdrv.lib

DriverString WIN32/64

port;baudrate

port The first parameter is the serial port to use (1=COM1, 2=COM2 and so on).

baudrate The second parameter is the baudrate for the serial interface. The baudrate is fixed at 57600 at so the second parameter is actually not used at the moment.

DriverString Linux

device;baudrate

device The first parameter is the device to use (/dev/ttyS0, /dev/ttyS1) etc.

baudrate The second parameter is the baudrate for the serial interface.

Flags Not used at the moment.

Status return The CanalGetStatus call returns the status structure with the channel_status member having the following meaning:

- Bit 0-7 - TX Error Counter.
- Bit 8-15 - RX Error Counter.
- Bit 16 - Overflow. Cleard by status read

- Bit 17 - RX Warning.
- Bit 18 - TX Warning.
- Bit 19 - TX bus passive.
- Bit 20 - RX bus passive..
- Bit 21 - Reserved.
- Bit 22 - Reserved.
- Bit 23 - Reserved.
- Bit 24 - Reserved.
- Bit 25 - Reserved.
- Bit 26 - Reserved.
- Bit 27 - Reserved.
- Bit 28 - Reserved.
- Bit 29 - Bus Passive.
- Bit 30 - Bus Warning status
- Bit 31 - Bus off status

Serial Protocol This is the description of the protocol used by the serial interface of the module.

A byte stuffing binary interface is used to talk to the module through the serial interface. Messages has the following format

```
[DLE]
[STX]
[flags]
[ID MSB]
[ID]
[ID]
[ID LSB]
[Databyte 0]
[Databyte 1]
[Databyte 2]
[Databyte 3]
[Databyte 4]
[Databyte 5]
[Databyte 6]
[Databyte 7]
[DLE]
[ETX]
```

Bit	Description	
7	Extended identifier	
6	RTR	
4,5	Code	Package type
	00	Data packet.
	01	Error packet/Command.
	10	Response ACK.
	11	Response NACK.
0,1,2,3	Number of databytes	

Table 15: CAN4VSCP protocol flags

Code	Description
0	Command response.
1	Overrun in CAN input buffer.
2	Bus Light - CAN error counter 96.
3	Bus Heavy - CAN error counter 128
4	Bus Off.
5-255	Reserved.

Table 16: CAN4VSCP Error packet code

There can be 0 to 8 databytes.

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

[DLE] = 0x10

[STX] = 0x02

[ETX] = 0x03

For an extended ID message only 29-bits of the 32-bit ID is valid. For a standard ID message only 11-bits of the 32-bit ID is valid.

flags are defined as Bits 4-5, **Error Packet/Command** have different meaning for receive and transmit. If an adapter sends this code it should be interpreted as if the packet is an error packet. If the driver software - such as the CANAL driver - use this bit combination it should be regarded as a command.

The first byte of the data field for error packets is a code that code the **Error Packet/Command**. This is coded in the following form for the CAN4VSCP and is always implementer specific.

Code 0 is reserved for command responses. How they are used are up to the implementer of the hardware. CAN4VSCP in the next firmware release will use

Command Code	Byte 0 of data field	Description
?	0	ACK
?	1	NACK
2	2	“CAN4VSCP\0” is returned in byte 3-11
3	3	Byte 3,4,5 will contain hardware version info as major version, minor version
4	4	Byte 3,4,5 will contain firmware version info as major version, minor version
5	5	Byte 3 will contain the error counter.

Table 17: CAN4VSCP command response codes

Note that the format for commands and responses does not follow the packet format.

When the adapter connects to the hardware (channel is opened) it should ask the adapter for its version information and by this verify that the adapter is present.

27.20.9 IXATT VCI interface Driver

This is a device driver for the IXXAT VCI Driver interface

IXXAT - Overview PC/CAN Interfaces (http://www.ixxat.com/download_vci_v3_en.html).

The driver uses proprietary include files and libraries from IXXAT which are not included in the distribution.

CANAL Driver ixxatvcidrv.dll(win32)

Include library for msvc ixxatvcidrv.lib

DriverString The driver string has the following format (note that all values can be entered in either decimal or hexadecimal form (for hex precede with 0x)).

board;channel;filter;mask;bus-speed;btr0;btr1

board board should be one of:

- VCI_IPCI165 or 0 for iPC-I 165, ISA slot
- VCI_IPCI320 or 1 for iPC-I 320, ISA slot
- VCI_CANDY or 2 for CANdy320, LPT port
- VCI_PCMCIA or 3 for tinCAN, pc card
- VCI_IPCI386 or 5 for iPC-I 386, ISA slot
- VCI_IPCI165_PCI or 6 for iPC-I 165, PCI slot
- VCI_IPCI320_PCI or 7 for iPC-I 320, PCI slot

- VCI_CP350_PCI or 8 for iPC-I 165, PCI slot
- VCI_PMC250_PCI or 9 for special hardware from PEP
- VCI_USB2CAN or 10 for USB2CAN, USB port
- VCI_CANDYLITE or 11 for CANdy lite, LPT port
- VCI_CANANET or 12 for CAN@net, Ethernet
- VCI_BFCARD or 13 for byteflight Card, pc card
- VCI_PCI04_PCI or 14 for PC-I 04, PCI slot, passive
- VCI_USB2CAN_COMPACT or 15 for USB2CAN compact, USB port
- VCI_PASSIV or 50 for PC-I 03, ISA slot, passive

channel channel Is a value from 0 an up indicating the CAN channel on the selected board.

filter filter Is the hardware dependent filter for this board hardware. Note that this filter may work in a different way then the CANAL filter.

mask mask Is the hardware dependent mask for this board hardware. Note that this filter may work in a different way then the CANAL filter.

bus-speed One of the predefined bit rates can be set here 10 for 10 Kbps 20 for 20 Kbps 50 for 50 Kbps 100 for 100 Kbps 125 for 125 Kbps 250 for 250 Kbps 500 for 500 Kbps 800 for 800 Kbps 1000 for 1000 Mobs

btr0/btr1 Value for bit rate register 0/1. If btr value pairs are given then the bus-speed parameter is ignored.

Flags

bit 0

- 0 11-bit identifier mode
- 1 29-bit identifier mode

bit 1

- 0 High speed
- 1 A low speed-bus connector is used (if provided by the hardware)

bit 2

- 0 Filter our own TX messages from our receive..
- 1 All sent CAN objects appear as received CAN objects in the rx queues.

bit 3

- 0 Active Mode.
- 1 Passive mode: CAN controller works as passive CAN node (only monitoring) therefore it does not send any acknowledge bit for CAN objects sent by another CAN node.

bit 4

- 0 No error report objects.
- 1 Error frames are detected and reported as CAN objects via the rx queues

Status return The CanalGetStatus call returns the status structure with the channel_status member having the following meaning:

- Bit 0-1 Reserved.
- Bit 2 RemoteQueOverrun
- Bit 3 CAN-TX pending
- Bit 4 CAN-Init-Mode.
- Bit 5 CAN-Data-Overrun.
- Bit 6 CAN-Error-Warning-Level.
- Bit 7 CAN_Bus_Off_status.
- Bit 8-15 Reserved.
- Bit 16 29-bit ID. Bit 17 Low speed-bus connector.
- Bit 18 All sent CAN objects appear as received CAN objects.
- Bit 19 Passive mode Bit 20 Error frames are detected and reported.
- Bit 21-29 Reserved. Bit 30 Bus Warning status (repeated from bit 6)
- Bit 31 Bus off status (repeated from bit 7)

Examples

VCI_USB2CAN_COMPACT;0;0;0;500

uses the USB2CAN compact USB adapter with a bus-speed off 500kbps and the first (and only) channel of the adapter with filter/mask fully open. This can also be set as

15;0;0;0;125

27.20.10 PEAK CAN Adapter Driver

This is a device driver for the PEAK family of cards (http://www.peak-system.com/index_gb.html).

The driver uses proprietary include files and libraries from PEAK which are not included in the distribution. Download driver

The driver is included in the installation of the VSCP and friends. Source for the driver is available in the source distribution or from CANAL/VSCP CVS vscp_cvs

Download area is <http://www.vscp.org/downloads.html>
or http://sourceforge.net/project/showfiles.php?group_id=53560

Important! Note that before you use this driver the PEAK original driver dll(s) for your particular adapter(s) must be in the dll search path. These DLL's are delivered from PEAK with your adapter. A good place to install them is in the windows system32 folder. The following table show the dll needed by the CANAL driver for each adapter Adapter PEAK DLL that is needed CANDONGLE pcan_dng.dll CANDONGLEPRO pcan_dnp.dll CANISA pcan_isa.dll CANPCI pcan_pci.dll CANPCI2 pcan_2pci.dll CANUSB pcan_usb.dll

CANAL Driver peakdrv.dll(win32)

Include library for msvc peakdrv.lib

DriverString The driver string has the following format (note that all values can be entered in either decimal or hexadecimal form (for hex precede with 0x)).

For non PnP boards/adapters

board;bus-speed;hwtype;port;irq;channel;filter;mask

For PnP boards/adapters

board;bus-speed;channel;filter;mask

To use default values just skip parameters. board is the only mandatory parameter.

board This is one of the following

- CANDONGLE or 0 for LPT port adapter
- CANDONGLEPRO or 1 for LPT port adapter (PRO version)
- CANISA or 2 for ISA adapter
- CANPCI or 3 for 1 channel PCI adapter
- CANPCI2 or 4 for 2 channel PCI adapter
- CANUSB or 5 for USB adapter

bus_speed This is the speed for the CAN bus. It can be given as

- 5 for 5 Kbps
- 10 for 10 Kbps
- 20 for 20 Kbps
- 50 for 50 Kbps
- 100 for 100 Kbps
- 125 for 125 Kbps
- 250 for 250 Kbps
- 500 for 500 Kbps
- 800 for 800 Kbps
- 1000 for 1000 Mbps

hwtype Only valid for non PNP cards

CANDONGLE

- 2 - dongle
- 3 - epp
- 5 - sja
- 6 - sja-epp

CANDONGLEPRO

- 7 - dongle pro
- 8 - epp

CANISA

- 1 - ISA
- 9 – SJA

port For ISA and parallel port adapters this is the hardware port address

irq This is the interrupt to use for non PNP devices.

channel Is a value from 0 an up indicating the CAN channel on the selected board.

filter Is the hardware dependent filter for this board hardware. Note that this filter may work in a different way than the CANAL filter.

mask Is the hardware dependent mask for this board hardware. Note that this filter may work in a different way than the CANAL filter.

example For the CANUSB adapter use

CANUSB;125

which says to use the CANUSB adapter at port 0 and 500 kbps.

If you have two adapters use

CANUSB;125;1

for the second adapter.

Flags

bit 0

- 0 11 bit identifier mode
- 1 11/29 bit identifier mode (Use for VSCP)

Status return The CanalGetStatus call returns the status structure with the channel_status member having the following meaning:

- Bit 0-15 PEAK Adapter specific (Taken from the CAN_Status method).
- Bit 16-28 Reserved.
- Bit 29 Reserved.
- Bit 30 Bus Warning status (repeated from bit 6) .
- Bit 31 Bus off status (repeat from bit 7).

27.20.11 Vector CAN Driver

This is a device driver for the Vector Informatik (<http://www.vector-informatik.com/english>) . The driver uses proprietary include files and libraries from Vector which are not included in the distribution.

CANAL Driver vectordrv.dll(win32)

Include library for msvc vectordrv.lib

DriverString The driver string has the following format (note that all values can be entered in either decimal or hexadecimal form (for hex precede with 0x)).

board;boardidx;channel;filter;mask;bus-speed

board board is one of

- VIRTUAL or 1 for virtual adapter
- CANCARDX or 2
- CANPARI or 3
- CANAC2 or 5
- CANAC2PCI or 6
- CANCARDY or 12
- CANCARDXL or 15 for PCI Card / PCMCIA
- CANCARD2 or 17
- EDICCARD or 19
- CANCASEXL or 21 for USB adapter
- CANBOARDXL or 25
- CANBOARDXL_COMPACT or 27

board index index for board if more then one of same type (0...n)

channel Is a value from 0 an up indicating the CAN channel on the selected board.

filter Is the hardware dependent filter for this board hardware. Note that this

filter may work in a different way then the CANAL filter.

mask Is the hardware dependent mask for this board hardware. Note that this filter may work in a different way then the CANAL filter.

bus_speed The actual bit rate is set here i.e. 125b is given as 125,000

Flags bit 0 Reserved

bit 1 Reserved

bit 2

- 0 Filter our own TX messages from our receive..
- 1 All sent CAN objects appear as received CAN objects in the rx queues.

bit 3

- 0 Active Mode.
- 1 Passive mode: CAN controller works as passive CAN node (only monitoring) and therefore it does not send any acknowledge bit for CAN objects sent by another CAN node.

bit 4

- 0 No error report objects.
- 1 Error frames are detected and reported as CAN objects via the rx queues.

Status return The CanalGetStatus call returns the status structure with the channel_status member having the following meaning:

- Bit 0-7 TX error counter
- Bit 8-15 RX error counter
- Bit 16-17 Reserved
- Bit 18 All sent CAN objects appear as received CAN objects.
- Bit 19 Passive mode.
- Bit 20-27 Reserved.
- Bit 28 Active.
- Bit 29 Bus Passive.
- Bit 30 Bus Warning status.
- Bit 31 Bus off status.

Platforms

WIN32

27.20.12 Zanthic CAN Driver

This is a device driver for the Zanthic Technologies (<http://www.zanthic.com/products.htm>) USB adapter.

CANAL Driver zanthicdrv.dll(win32)

Include library for msvc zanthicdrv.lib

DriverString The driver string has the following format (note that all values can be entered in either decimal or hexadecimal form (for hex preceded with 0x)).

bus-speed This is the speed for the CAN bus. It can be given as

- 10 for 10 Kbps
- 20 for 20 Kbps
- 50 for 50 Kbps
- 100 for 100 Kbps
- 125 for 125 Kbps
- 250 for 250 Kbps
- 500 for 500 Kbps
- 800 for 800 Kbps
- 1000 for 1000 Mbps

Flags Not used set to 0.

Status return The CanalGetStatus call returns the status structure with the channel_status member having the following meaning:

- Bit 0-31 Reserved.

27.20.13 LIRC Driver

Device driver for that connects to LIRC <http://www.lirc.org> (Unix) or WINLIRC <http://winlirc.sourceforge.net> (Windows) and receive data from IR remotes and sends IR data out.

LIRC/WINLIRC must be functioning on the computer that use this driver. Follow the information on respective site on how to set them up.

CANAL Driver lircdrv.dll(win32), lircdrv.so(Linux)

Include library for msvc lircdrv.lib

DriverString

"path to config file;lirchost;lircport"

path to config file The full path to the location for the configuration XML-file.

lirchost Hostname for the host the LIRC daemon is running on. If not given "localhost" is assumed.

lircport The port that the LIRC daemon is listening to. If not given the default-port "8765" is used.

Flags Not used.

Status return Not used. Platforms

Configuration file format In the configuration file it is possible to set what VSCP (Level I or Level II) events or generic CAN messages that should be sent (none, one or many) for a specif key pressed on the remote.

The format for this file is XML and is defined as follows:

See source files

27.20.14 Logger Driver

Device driver for diagnostic logging. It allows you to log CAN (VSCP Level I) traffic to a text file. Several drivers can be loaded with different output files and using different filter/masks.

CANAL Driver canallogger.dll(win32), canallogger.so(Linux)

Include library for msvc canallogger.lib

DriverString Windows

c:|logfile.log;0x0;0x0

Linux

/tmp/logfile;0x0;0x0

The absolute or relative path including the file name to the file that log data should be written to. The filename is followed by the optional 32-bit filter and mask. The default is all events logged.

Note that the filter/mask looks at the CAN ID. If you work with VSCP look at format of the 29 bit CAN identifier in VSCP Bit 32 is set to one for an extended frame (all VSCP frames) and bit 30 is set to one for RTR frames (never for VSCP).

Flags

- 0 – Append data to an existing file (Create if not available).
- 1 – Create a new file or rewrite data on an old file.

Status return The CanalGetStatus call returns the status structure with the channel_status member having the following meaning: 0 is always returned.

Log file format The log file have the following format and consist of the following parts

- Time when frame was received
- Timestamp
- Flags
- ID
- Number of databytes
- Databytes

27.20.15 TCP Driver

This is a device driver that can be used to set up a TCP link and exchange CAN packet data through this link.

CANAL Driver tcpdrv.dll(win32), tcpdrv.so(Linux)

Include library for msvc tcpdrv.lib

DriverString The driver string has the following format (note that all values can be entered in either decimal or hexadecimal form (for hex precede with 0x)).

remote_url;remote_port;timeout;local_ip;local_port

remote_url remote server to connect to. Either as an ip address or standard url.

remote_port The port to connect to at the remote URL.

timeout If no packages are received during this time the link will be closed. Set to zero for no timeout.

local_ip If local_ip is given then remote connections will be accepted on this interface.

local_port This is the port used for remote connections.

Flags Not used set to 0.

Status return The CanalGetStatus call returns the status structure with the channel_status member having the following meaning: *Currently undefined*.

27.20.16 UDP Driver

This is a device driver that can be used to set up a UDP link and exchange CAN packet data through this link. CANAL Driver udpdrv.dll(win32), udpdrv.so(Linux)

Include library for msvc udpdrv.lib

DriverString The driver string has the following format (note that all values can be entered in either decimal or hexadecimal form (for hex precede with 0x)).

remote_url;remote_port;timeout;local_ip;local_port

remote_url Remote server to connect to. Either as an ip address or standard url. **remote_port** the port to connect to at the remote URL.

timeout If no packages are received during this time the link will be closed. Set to zero for no timeout.

local_ip If local_ip is given then remote connections will be accepted on this interface.

local_port This is the port used for remote connections.

Flags Not used set to 0.

Status return The CanalGetStatus call returns the status structure with the channel_status member having the following meaning: *Currently undefined*

27.20.17 xAP Driver

This is a driver that make the VSCP/CANAL daemon behaves like a xAP server or client. If no other server is detected when the driver is loaded it will start up in server mode. If a server is detected it will start up in client mode and connect to that server.

As this is a driver only Level I events will be transferred between VSCP and xAP.

CANAL Driver xapdrv.dll(win32), xapdrv.so(Linux)

Include library for msvc xapdrv.lib

DriverString

uid;port

where

uid is the ID for this device. Use a different ID for every device. Defaults to 9598.

port Port is the xAP port and defaults to 3639.

Flags Not used at the moment.

Status return Not used at the moment.

28 Creating a Level I (CANAL) driver for new hardware

The CANAL interface is just a dll with a specific interface to the world that acts as a black box against the original driver software. It is of course also possible to build the original driver so that it will have the canal interface. When this is done, which is a very easy task, the device is usable by all software that adopts the canal specification. The canal daemon is important software in this case.

A specific piece of software can work against the canal daemon and thus all drivers the daemon serves or just add one specific driver. The program just uses the correct dll.

If you work on the Windows platform there is source in the generic dll that can be used as a template for a new driver. If you are making something that is close in functionality to one of the other available drivers then the one that is closest is the natural start for a new project.

Part IX

Level II Drivers

Level II drivers are drivers that can be used to extend the VSCP server capabilities. They can connect to different type of hardware. Level II drivers have two advantages over Level I drivers.

- They always use the Level II event format which means that the full GUID is used.
- They interface the daemon through the full TCP/IP interface giving a lot of possibilities for the driver to interact with the daemon.

The ability to use the full GUID is good as there is no need for translation schema's between the actual GUID and GUID's used in interfaces. The node-ID is unique all over the world.

Letting the driver talk to the daemon over the TCP/IP interface is favorable in that it can do many things that previously has been impossible. The most exciting is that it can read and write variables (even create new ones if needed). This is the recommended way to use for configurations of a Level II driver. It means that configuration of all drivers can be made in one place (the daemon variable file), it gives a possibility to change runtime values in real time etc.

The level II driver is, just as the Level I driver, a dynamic link library with a specific set of exported methods. The exported methods are four of the methods from the CANAL interface and uses identical calling parameters and return values. There are some differences however noted below.

The configuration for a typical VSCP driver in the vscpd.conf file looks like this

```
<vscpdriver> <!-- Information about VSCP Level II drivers -->
    <driver prefix="logger1" >
        <name>l2logger</name>
        <config>d:\vscplog.txt;1</config>
        <path>
            /us/local/lib/vscpl2logger.so
        </path>
        <guid>00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00</guid>
    </driver>
</vscpdriver>
```

The driver name is used when the driver is presented to the world such as in the tcp/ip interface. The name should be unique within the system. The prefix is added in front of variable names the driver fetch or write to. Typically variables have names like “_variable” and the actual variable fetches is thus “prefix_variable”. “prefix” is set to the driver name set in the VSCP daemon configuration file for this driver. If the same driver is used for several times this

name should be different for each driver if different configuration data should be read in for the drivers. *The prefix* is used in front of configuration variables and it will be used on the form *prefix_variablename* when the driver reads in additional driver configuration data.

In the parameter a list with semicolon separated values can be stored. Each entry represent a specific configuration value. Even if they are present the driver will try fetch the corresponding value from the daemon and if it is found it will replace the value set in the parameters list. The path is the location where the driver dynamic link library is located. Guid is the guid used for the interface. If absent or set to all zeros the daemon assigned guid will be used for the interface.

Events received by the daemon from a Level II driver use the GUID set in the event if not all nills. If it has all nills the device GUID will be used if it is not all nills and if not the interface GUID will be used.

29 VSCP driver API

29.1 VSCPOpen

```
long VSCPOpen( const char *pUsername, const char *pPassword,
const char *pHost, short port, const char *pPrefix, const char *pCon-
figuration, unsigned long flags );
```

Start the driver and give it some initial configuration data. Before the driver is started by the daemon one parameters is added in front of the configuration string by the daemon. *Prefix* is a user defined string that is used as a prefix for variables read/write/construct. This can be an empty string but it is recommended to set a value so that the same driver can be used and configured for different things at the same time. Typically this string has the form “mydriver” or something like that. Variables used to configure this particular driver then need to use the same prefix. It is up to the driver to decide if the prefix should be used or not.

pUsername and *pPassword* is either a random generated pair of credentials constructed by the daemon or set from the configuration file. *pHost* and *port* is either set from the configuration file or *pHost* is set to “localhost” by the daemon and *port* is set to the VSCP port 9598.

pPrefix is set to the driver name fetched from the daemon configuration file. *pConfiguration* is the normally semicolon separated list of configuration values supplied by the user. The *flags* parameter is not used at the moment.

If the driver can initialize and is started it will return a handle for open physical interface or $<= 0$ on error. For an interface where there is only one channel the handle has no special meaning and can only be looked upon as a status return parameter.

29.2 VSCPclose

```
int VSCPclose( long handle );
```

If the driver can close/stop the driver it will return TRUE (non zero) on success or FALSE on failure.

29.3 VSCPBlockingSend

```
int VSCPBlockingSend( long handle, const vscpEvent *pEvent,  
unsigned long timeout ); Send an event while blocking for timeout (forever  
if timeout=0).
```

29.4 VSCPBlockingReceive

```
int VSCPBlockingReceive( long handle, vscpEvent *pEvent, un-  
signed long timeout ); Blocking receive of an event. Blocks for timeout and  
forever if timeout=0.
```

CANAL_ERROR_SUCCESS is received if an event is received and CANAL_ERROR_TIMEOUT on timeout.

29.5 VSCPGetLevel

```
unsigned long VSCPGetLevel( long handle ) Will always return  
CANAL_LEVELUSES_TCPIP for a Level II driver. The constant is defined  
in canal.h May return something else in the future if the driver is extended.
```

29.6 VSCPGetWebPageTemplate

```
VSCPGetWebPageTemplate( long handle, const char *url, char  
*page ) With this method it is possible for a driver to add internal web  
server functionality. Pages with an address
```

/vscp/drivername/url

will be recognized as a driver page and therefore redirected to this method of the the driver expect a pointer to a webpage in return or NULL for a non recognized url. **Important!** If a pointer is returned it is the calling program that should deallocate the data returned.

If you want to make it possible to get information from or configure a driver this is a way to do it. Pages that posts data will have there data scanned by the internal web-server of the daemon and found data values is written as strings to server variables (see 14.6.17) which are constructed if they are not already defined.

Before a page is sent to a user escapes as defined in the table below are replaced with current values

Escape	Description
%server	Address for the server.
%date	ISO formated date.
%time	ISO formated time.

The standard menu will be shown on top of each page.

29.7 VSCPGetWebPageInfo

VSCPGetWebPageInfo(long handle, const struct vscpextwebpageinfo *info) This call can be used to get information about what pages the driver will recognize abd a description of what they do. This information is displayed in the web interface. The structure that should be returned is defined in *vscp/src/common/vscpdlldef.h*

29.8 VSCPWebPageupdate

VSCPWebPageupdate(long handle, const char *url) When a page is posted to a driver url variable data will be written to server variables by the server. If variables are undefined they will be created. After this has been done a call to this method will be carried out.

29.9 VSCPGetDllVersion

unsigned long VSCPGetDllVersion (void) Gets the version for the driver dll/dl.

29.10 VSCPGetVendorString

const char * VSCPGetVendorString (void) Get a string that identifies the creator of the driver.

29.11 VSCPGetDriverInfo

const char * VSCPGetDriverInfo(void) Get information about the interface. Either NULL if there is no information or a pointer to a string with XML information.

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

<!-- Version 0.0.2      2009-11-17
     "string"      Text string.

     "bool"        1 bit number specified as true or false.

     "char"        8 bit number. Hexadecimal if it starts with "0x" else decimal.

     "uchar"       Unsigned 8 bit number. Hexadecimal if it starts with "0x"
                   else decimal.

     "short"       16 bit signed number. Hexadecimal if it starts with "0x"
```

```

else decimal.

"ushort"      16 bit unsigned number. Hexadecimal if it starts with "0x"
else decimal.

"int"         32 bit signed number. Hexadecimal if it starts with "0x"
else decimal.

"uint"        32 bit unsigned number. Hexadecimal if it starts with "0x"
else decimal.

"long"        64 bit signed number. Hexadecimal if it starts with "0x"
else decimal.

"ulong"       64 bit unsigned number. Hexadecimal if it starts with "0x"
else decimal.

"decimal"     128 bit number. Hexadecimal if it starts with "0x" else deci
"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 cl
-->

<vacpdriver>
<drivername>aaaaaaaa</drivername>
<arch>WIN32|WIN64|LINUX</arch>
<model>bbbbbb</model>
<version>cccccc</version>
<description lang = "en">yyyyyyyyyyyyyyyyyyyyyyyyyyyy</description>

<!-- Site with info about the product -->
<infourl>http://www.somewhere.com</infourl>

<!-- Information about CANAL driver maker -->
<maker>
    <name>ttttttttttttttt</name>
    <address>
        <street>ttttttttttt</street>
        <town>111111111</town>
        <city>London</city>
        <postcode>HH1234</postcode>
        <!-- Use region or state -->
        <state></state>
        <region></region>
        <country>tttt</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>someone@somwhere.com</email>

<!-- One or many -->
<web>www. somewhere.com</web> </maker>

<configstring>

    <!-- Option example with selectable values -->
    <option pos="0" type="string">
        <name lang="en">aaaaaa</name>
        <description lang="en">yyy</description>
        <valuelist>
            <item value="CANUSB" />
                <name lang="en">USB Adapter</name>
                <description lang="en">yyy</description>
            </item>
            <item value="CANPCI" />
                <name lang="en">PCI Adapter</name>
                <description lang="en">yyy</description>
            </item>
            <item value="CAN232" />
                <name lang="en">Serial Adapter</name>
                <description lang="en">yyy</description>
            </item>
        </valuelist>
    </option>

    <!-- Option example with numerical value -->
    <option pos="1" type="uchar" min="0" max="4">
        <name lang="en">aaaaaa</name>
        <description lang="en">yyy</description>
    </option>

```

```

</configstring>

<!-- Flags part (32-bit value) of the device information -->
<flags>
    <description lang="en">yyy</description>
    <bit pos="0">
        <name lang="en">tttt</name>
        <description lang="en">yyy</description>
    </bit>
    <bit pos="1" width="4">
        <!-- example for bit groups, in this case bit 1,-->
        <name lang="en">tttt</name>
        <description lang="en">yyy</description>
    </bit>
</flags>

<!-- Status return value (32-bit value) -->
<status>
    <bit pos="0">
        <name lang="en">tttt</name>
        <description lang="en">yyy</description>
    </bit>
    <bit pos="1" width="4">
        <name lang="en">tttt</name>
        <description lang="en">yyy</description>
    </bit>
</status>

<!-- Description of variable the driver reads in -->
<variable>
    <name>ksdkjskjskdjksdjks</name>
    <type>bool|string|etc etc</type>
    <description>ksksjksjkdjs</description>
</variable>

</vscpdriver>

```

30 Level II drivers for the Windows and Linux platform

30.1 Logger driver

The logger driver can be used to log events to a file. Two formats of the log file is supported. Either standard log file with a standard text string for each

event on each line or in XML file format which can be read by VSCP works and further analyzed there.

Driver: *vscpl2_loggerdrv.so* The configuration string have the following format

path;rewrite;vscpworksfmt;filter;mask

See the table below for a description. Variables fetched from the daemon are used if they are available even if the configuration string specify a value.

Variables fetched from the VSCP demon configuration file.

Variable name	Type	Description
<i>_path</i>	string	Path for the logfile.
<i>_rewrite</i>	bool	Set to “ <i>true</i> ” to rewrite the file each time the driver is started. Set to “ <i>false</i> ” to append to file.
<i>_vscpworksfmt</i>	bool	If “ <i>true</i> ” VSCP works XML format will be used for the log file. This means that the file will be possible to read and further analyzed by VSCP Works. If “ <i>false</i> ” a standard text based format will be used.
<i>_filter</i>	string	Standard VSCP filter in string form. <i>1,0x0000,0x0006,ff:ff:ff:ff:ff:ff:01:00:00:00:00:00:00:00</i> as priority,class,type,UUID
<i>_mask</i>	string	Standard VSCP mask in string form. <i>1,0x0000,0x0006,ff:ff:ff:ff:ff:ff:01:00:00:00:00:00:00:00</i> as priority,class,type,UUID

Example of vscpd.conf entry for the logger driver.

```
<driver enable="true" >
    <name>VSCP Logger</name>
    <path>/usr/local/lib/vscpl2_loggerdrv.so</path>
    <config>/tmp/vscp_level2.log</config>
    <guid>00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00</guid>
</driver>
```

30.2 Raw Ethernet driver

From version *0.3.0 Carbon* VSCP support raw Ethernet. This driver gives support for nodes that communicate using the VSCP raw Ethernet format (see 2.4) and it makes it possible to build nodes that use the vscp protocol and behave like high level nodes without the extra burden of a TCP/IP stack.

To use this driver on Windows the winpcap package must be installed. This package can be found here <http://www.winpcap.org/install/default.htm>. To use it on Linux the daemon must be run as root and you need the libpaps-dev package installed.

Driver: vscp2_rawethernetdrv.so The configuration string have the following format

Device;LocalMac

Device is a string that identifies the Ethernet device that should be used when sending/receiving Ethernet frames. Typically it's in the form of eth0, eth1, eth2 etc on Linux devices and \Device\NPF_{986752B0-3C0E-46A2-AFD3-B593E180EC54} on Windows. The tool *iflist* that is distributed with VSCP & friends on Windows can be used to list available interfaces on a specific system.

local-mac is the mac address used as the outgoing mac address when sending raw Ethernet frames. Normally this should be the same as for the adapter. The form i typically 00:26:55:CA:1F:DA.

Variables fetched from the VSCP demon configuration file.

Variable name	Type	Description
interface	string	A string that identifies the Ethernet device that should be used when sending/receiving Ethernet frames. Typically it's in the form of eth0, eth1, eth2 etc on Linux devices and \Device\NPF{986752B0-3C0E-46A2-AFD3-B593E180EC54} on a Windows system.
_localmac	string	The mac address used as the outgoing mac address when sending raw Ethernet frames. Normally this should be the same as for the adapter. The form i typically 00:26:55:CA:1F:DA.
_filter	string	Standard VSCP filter in string form. 1,0x0000,0x0006,ff:ff:ff:ff:ff:ff:01:00:00:00:00:00:00:00 as priority,class,type,GUID Used to filter what is sent from VSCP out on Ethernet.
_mask	string	Standard VSCP mask in string form. 1,0x0000,0x0006,ff:ff:ff:ff:ff:ff:01:00:00:00:00:00:00:00 as priority,class,type,GUID Used to filter what is sent from VSCP out on Ethernet.

A typical configuration example from a Windows 7 machine

Example of vscpd.conf configuration for raw ethernet driver.

```
<vscpdriver>
    <driver enable="true">
        <name>rawethernet1</name>
        <config>eth0</config>
        <path>/usr/local/lib/vscp2drv_raweth.so</path>
    </driver>
</vscpdriver>
```

30.3 Lm-sensors driver

Lm-sensors is the effort of building and providing some essential tools for monitoring the hardware health of Linux systems containing hardware health monitoring hardware such as the LM78 and LM75. This ongoing project includes general libraries and hardware-specific software. Much of the work done could not have been done without the many knowledgeable programmers who laid down the foundation. The projects home page is here <http://www.lm-sensors.org/>. Examples of units that can be monitored is temperatures, voltage, and fans.

This driver is used to monitor sensors that are exported by the lm-sensors system or other systems and convert them to VSCP events that is sent out on user specified periodic intervals.

As lm-sensor data is exported in the proc file system this driver can be used also for other sensors that can export it's data using this method or use an ordinary file as long as the data is numerical and is located at a specific offset in the file.

A good description on how to set up the lm-sensors system is here https://wiki.archlinux.org/index.php/Lm_sensors and an example of how to use the lm-sensors driver is here http://www.vscp.org/wiki/doku.php/howto/driver_lm_sensors

Driver: vscp2_lmsensorsdrv.so The configuration string has the following format

NumberOfSensors

The parameter *NumberOfSensors* (*which is optional*) is the number of sensors the driver should report data from. This value can also be available as a variable and if both are present the variable will be used.

Variable name	Type	Description
<code>_numberofsensors</code>	integer	<i>NumberOfSensors</i> is the number of sensors the driver should report data from.
<code>_path[0..n]</code>	string	Path to the lm-sensor data file.
<code>_guid[0..n]</code>	guid	GUID to use when the data for the sensor is reported.
<code>_interval[0..n]</code>	integer	Sample interval in seconds for events.
<code>_vscptype[0..n]</code>	integer	VSCP Type to use for events. Currently the type must be one from CLASS1.MEASUREMENT (see 12.4)
<code>_vscpclass[0..n]</code>	integer	VSCP class to use. Currently CLASS1.MEASUREMENT CLASS1.MEASUREMENT64 CLASS1.MEASUREZONE CLASS1.SETVALUEZONE CLASS2.MEASUREMENT_STR is allowed.
<code>_vscpinde</code>	integer	Needed index parameter if CLASS2.MEASUREMENT_STR, CLASS1.MEASUREZONE or CLASS1.SETVALUEZONE class is used
<code>_vscpzon</code>	integer	Needed zone parameter if CLASS2.MEASUREMENT_STR, CLASS1.MEASUREZONE or CLASS1.SETVALUEZONE class is used.
<code>_vscpsubzon</code>	integer	Needed subzone parameter if CLASS2.MEASUREMENT_STR, CLASS1.MEASUREZONE or CLASS1.SETVALUEZONE class is used.
<code>_coding[0..n]</code>	integer	VSCP measurement coding to use for events. Not used if class is Index use if class is CLASS2.MEASUREMENT_STR
<code>_multipli[0..n]</code>	double	This is the factor that will be multiplied with the value read from the lm-sensors file. If absent the value is multiplied by one. Note that this is a floating point value and can not be entered as a hex value.
<code>_divide[0..n]</code>	double	This is the factor that will be divided with the value read from the lm-sensors file. If absent the value is divided by one. Note that this is a floating point value and can not be entered as a hex value.
<code>_readoffset[0..n]</code>	int	Normally for a lm-sensors file the readoffset should be 0 (default) as the value starts at zero offset. But for other files, such as the output from 1-wire tool such as DigiTemp http://www.digitemp.com/ , the offset to the data may be different.
<code>_index</code>	int	Index use if class is CLASS2.MEASUREMENT_STR, CLASS1.MEASUREZONE or CLASS1.SETVALUEZONE
<code>_zon</code>	int	Zone use if class is CLASS2.MEASUREMENT_STR, CLASS1.MEASUREZONE or CLASS1.SETVALUEZONE
<code>_subzon</code>	int	Subzone use if class is CLASS2.MEASUREMENT_STR, CLASS1.MEASUREZONE or CLASS1.SETVALUEZONE
<code>_unit</code>	int	Unit use if class is CLASS2.MEASUREMENT_STR, CLASS1.MEASUREZONE or CLASS1.SETVALUEZONE

Example of vscpd.conf entry for the lmsensors driver.

```
<driver enable="true" >
  <name>lmsensors1</name>
  <path>/usr/local/lib/vscp12_lmsensorsdrv.so</path>
  <config>2</config>
  <guid>00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00</guid>
</driver>
```

Example for variable configuration. In this sample the core temperature for two CPU's is reported as CLASS1.MEASUREMENT, Type=6 temperature.

```
<!-- **** VSCP lmsensor Level II driver variables -->
<!-- Number of sensors to read -->
<variable type="int" >
  <name>lmsensors_numberofsensors</name>
  <value>1</value>
</variable>

<!-- Variabed for temperature CPU core 0 -->
<!-- Interval for CPU core0 -->
<variable type="int" >
  <name>lmsensors_interval0</name>
  <value>1</value>
</variable>

<!-- VSCP GUID --> <variable type="guid" >
<variable>
  <name>lmsensors_guid0</name>
  <value>00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00</value>
</variable>

<!-- Path to sensor data file for CPU core 0 -->
<variable type="string" >
<name>lmsensors_path0</name>
<value>/sys/class/hwmon/hwmon1/device/temp2_input</value>
</variable>

<!-- VSCP class value -->
```

```

<variable type="int" >
<name>lmsensors_vscpclass0</name>
<value>10</value> <!-- Measurement class -->
</variable>

<!-- VSCP Type value -->
<variable type="int" >
    <name>lmsensors_vscptype0</name>
    <value>6</value> <!-- Temperature -->
</variable>

<!-- Multiply value -->
<variable type="int" >
    <name>lmsensors_multiply0</name>
    <value>1</value>
</variable>

<!-- VSCP datacoding value -->
<variable type="int" >
    <name>lmsensors_datacoding0</name>
    <value>8</value>
</variable>

<!-- Divide value -->
<variable type="int" >
    <name>lmsensors_divide0</name>
    <value>1</value>
</variable>

<!-- **** Variabled for temperature CPU core 0 -->
<!-- **** -->
<!-- **** -->

<!-- Interval for CPU core1 -->
<variable type="int" >
    <name>lmsensors_interval1</name>
    <value>1</value>
</variable>

<!-- VSCP GUID -->
<variable type="guid" >
    <name>lmsensors_guid1</name>
    <value>00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00</value>
</variable>

```

```

<!-- Path to sensor data file for CPU core 1 -->
<variable type="string" >
    <name>lmsensors_path1</name>
    <value>/sys/class/hwmon/hwmon1/device/temp3_input</value>
</variable>

<!-- VSCP class value -->
<variable type="int" >
    <name>lmsensors_vscpclass1</name>
    <value>10</value> // Measurement class
</variable>

<!-- VSCP Type value -->
<variable type="int" >
    <name>lmsensors_vscptype1</name>
    <value>6</value> // Temperature
</variable>

<!-- VSCP datacoding value -->
<variable type="int" >
    <name>lmsensors_datacoding1</name>
    <value>8</value>
</variable>

<!-- Multiply value -->
<variable type="int" >
    <name>lmsensors_multiply1</name>
    <value>1</value>
</variable>

<!-- Divide value -->
<variable type="int" >
    <name>lmsensors_divide1</name>
    <value>1</value>
</variable>

```

30.4 Socketcan driver

SocketCAN, the official CAN API of the Linux kernel, has been included in the kernel more than 3 years ago. Meanwhile, the official Linux repository has device drivers for all major CAN chipsets used in various architectures and bus types. SocketCAN offers the user a multiuser capable as well as hardware independent socket-based API for CAN based communication and configuration. Socketcan nowadays give access to the major CAN adapters that is available on the market. Note that as CAN only can handle Level I events only events up to class < 1024

can be sent to this device. Other events will be filtered out. Also received events

Driver: vscp2_rawethernetdrv.so The configuration string have the following format

Interface

The first parameter interface is the socketcan interface to use. Typically this is “can0, can0, can1...” Defaults is vcan0 the first virtual interface. If the variable *prefix_interface* is available it will be used instead of the configuration value.

Variable name	Type	Description
<i>_interface</i>	string	The socketcan interface to use. Typically this is “can0, can0, can1...” Defaults is vcan0 the first virtual interface.
<i>_filter</i>	string	Standard VSCP filter in string form. <i>1,0x0000,0x0006,ff:ff:ff:ff:ff:ff:01:00:00:00:00:00:00:00:00:00</i> as priority,class,type,GUID Used to filter what events that is received from the socketcan interface. If not give all events are received.
<i>_mask</i>	string	Standard VSCP mask in string form. <i>1,0x0000,0x0006,ff:ff:ff:ff:ff:ff:01:00:00:00:00:00:00:00:00:00:00</i> as priority,class,type,GUID Used to filter what events that is received from the socketcan interface. If not give all events are received.

vscpd.conf example

```
<driver enable="true" >
    <name>SocketCAN1</name>
    <path>/usr/local/lib/vscpl2_socketcandrv.so</path>
    <config>can0</config>
    <guid>00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00</guid>
</driver>
```

30.5 TCP driver

The tcp driver can be used to connect to other daemons or connect to limited TCP/IP devices that export a subset of the commands available in the VSCP daemons. A required subset must be available described here 14.6.3 and here 14.6.4.

The driver will try to hold a connection open even if the remote node disconnects. This makes it possible to replace a node or take it down for maintenance and still have the link online again as soon as the node is powered up.

Driver: vscpl2_tcpdrv.so The configuration string have the following format

host;port;user;password;filter;mask

Host and port tells the address to the other server. User and password is the credentials to log in to the machine. Filter and mask can be used to filter incoming traffic from the remote host. All of the parameters in the configuration string can also be set with variables.

Variable name	Type	Description
<i>_host_remote</i>	string	IP address or a DNS resolvable address to the remote host. Mandatory and must be declared either in the configuration string or in this variable.
<i>_port_remote</i>	integer	The port to use on the remote host. Default is 9598.
<i>_user_remote</i>	string	Username used to log in on the remote sever.
<i>_password_remote</i>	string	Password used to login on the remote server.
<i>_filter</i>	string	Standard VSCP filter in string form. <i>1,0x0000,0x0006,ff:ff:ff:ff:ff:ff:01:00:00:00:00:00:00:00:00</i> as priority,class,type,GUID Used to filter what events that is received from the socketcan interface. If not give all events are received.
<i>_mask</i>	string	Standard VSCP mask in string form. <i>1,0x0000,0x0006,ff:ff:ff:ff:ff:ff:01:00:00:00:00:00:00:00:00</i> as priority,class,type,GUID Used to filter what events that is received from the socketcan interface. If not give all events are received.

vscpd.conf example

```
<driver enable="true" >
  <name>tcpiplink1</name>
  <path>/usr/local/lib/vscp2drv_tcpiplink.so</path>
  <config>192.168.1.20;9598;admin;secret</config>
  <guid>00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00</guid>
</driver>
```

30.6 MQTT driver

MQTT provides a lightweight method of carrying out messaging using a publish/subscribe model. This makes it suitable for "machine to machine" messaging such as with low power sensors or mobile devices such as phones, embedded computers or microcontrollers like the Arduino. A good example of this is all of the work that Andy Stanford-Clark (one of the originators of MQTT) has done in home monitoring and automation with his twitting house and twitting ferry. The MQTT protocol is maintained by IBM.

This driver allow publishing of VSCP events as well as subscribing to VSCP events.

It is also described in the wiki http://www.vscp.org/wiki/doku.php/howto/driver_level2_mqtt

Driver: *vscpl2_mqtt.so* The configuration string have the following format
“subscribe”/“publish”;channel;host;port;user;password;keepalive;filter;mask

The first configuration parameter sets if the intention is to subscribe (“subscribe”) to an existing MQTT channel or to publish (“publish”) events on a channel. The second parameter is the topic. This is a text string identifying the topic. It is recommended that this string starts with “vscp/”. Host is the host where the MQTT broker is located (defaults to *localhost*). Port is the host-port at the host and defaults to 1883. user/password is credentials for the channel if they are needed.

Variable name	Type	Description
<code>_type</code>	string	“subscribe” to subscribe to a MQTT topic. “publish” to publish events to a MQTT topic. Defaults to “subscribe”.
<code>_topic</code>	string	This is a text string identifying the topic. It is recommended that this string starts with “vscp/”. Defaults to “vscp”
<code>_host</code>	string	IP address or a DNS resolvable address to the remote host. Mandatory and must be declared either in the configuration string or in this variable. Defaults to “localhost”
<code>_port</code>	integer	The port to use on the remote host. Default is 1883.
<code>_user</code>	string	Username used to log in on the remote MQTT sever. Defaults to empty. Currently not used.
<code>_password</code>	string	Password used to login on the remote MQTT server. Defaults to empty. Currently not used.
<code>_keepalive</code>	int	Keepalive value for channel. Defaults to 60.
<code>_filter</code>	string	Standard VSCP filter in string form. <i>1,0x0000,0x0006,ff:ff:ff:ff:ff:ff:01:00:00:00:00:00:00:00:00</i> as priority,class,type,UUID Used to filter what events that is received from the socketcan interface. If not give all events are received/sent.
<code>_mask</code>	string	Standard VSCP mask in string form. <i>1,0x0000,0x0006,ff:ff:ff:ff:ff:ff:01:00:00:00:00:00:00:00:00</i> as priority,class,type,UUID Used to filter what events that is received from the socketcan interface. If not give all events are received/sent.
<code>_simplify</code>	string	This is not yet implemented in the current driver. If this variable is defined it will simplify the effort to publish or subscribe to information. It can only be used for one of three classes but may be extended in the future. VSCP_CLASS1_MEASUREMENT, VSCP_CLASS1_MEASUREMENT64, VSCP_CLASS1_MEASUREZONE and VSCP_CLASS2_MEASUREMENT_STR class=VSCP_CLASS1_MEASUREMENT Variable content: “ <i>10,vscptype;coding</i> ” A floating point string value is sent (publish) and received (subscribe) with the selected coding. class=VSCP_CLASS1_MEASUREMENT64 Variable content: “ <i>60,vscptype</i> ” A floating point string value is sent (publish) and received (subscribe). Coding is always zero i.e. the default coding. class=VSCP_CLASS1_MEASUREZONE ⁴⁸⁷ Variable content: “ <i>65,vscptype,0,zone,zubzone</i> ” A floating point string value is sent (publish) and received (subscribe). Coding is always zero i.e. default coding. A received value is sent using zone/subzone. class=VSCP_CLASS2_MEASUREMENT_STR Variable content: “ <i>1040,vscptype,coding,zone,zubzone</i> ”

```

vscpd.conf example

<driver enable="true" >
  <name>VSCP MQTT Publisher driver 1</name>
  <path>/usr/local/lib/vscp2drv_mqtt.so</path>
  <config>publish;vscp;localhost;1883;;60</config>
  <guid>00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00</guid>
</driver>
```

The format for data on the MQTT wire is the same as the format for sends and receives in the TCP/IP interface. This means they look like this

head , class , type , obid , time-stamp , GUID , data0 , data1 , data2 ,

and as usual the GUID can be replaced by “-” to use the interface GUID.

Using the methods

- writeVscpEventToString
- writeVscpEventExToString
- getVscpEventFromString
- getVscpEventExFromString

available in the vscphelper library (C/C++/C#/Python) it is easy to convert between VSCP Event and text form. It is also very easy to get other properties from the event such as real number measurement with the help of the same library.

30.7 Bluetooth proximity driver

The Bluetooth proximity driver can be used to detect Bluetooth devices. It can send out the ID of the device using *CLASS1:INFORMATION=20, Type=37* when it enters/leave the detection zone and also a detect event *CLASS1:INFOMATION=20, Type=49* if configured to do so.

Driver: vscp2_btproximitydrv.so The configuration string is not used.

The *flags* bit field is not used.

Variables fetched from the VSCP demon configuration file.

Variable name	Type	Description
_paustime	integer	Pause in seconds between detection attempts.
_zone	integer	Zone to use for token events.
_subzone	integer	Sub-zone to use for token events.
_detectindex	integer	Index to use for detect events.
_detectzone	integer	Zone to use for detect events.
_detextsubzone	integer	Sub-zone to use for detect events.
_send_token_activity	boolean	Set to true to enable token events.
_send_detect	boolean	Set to true to enable detect events.
_disable_radio_detect	boolean	Normally also the radio unit is detected. Set this variable to true to disable this.

Example of vscpd.conf entry for the Bluetooth driver.

Part X

Cookbook & HOWTO's

31 VSCP

31.1 How to make a VSCP system with operation confirm

Here we want to design a system with one button that could turn on an arbitrary number of other nodes on to there ON-state on a segment. The nodes get activated (lamps lit) with the CLASS1.CONTROL, Type=5, Turn-ON-event and the nodes should respond with CLASS1.INFORMATION, Type=3 (ON) to indicate they have been lit.

On a CAN bus checking the replies may not be critical as it will most often work. But there is a chance that it will not as CAN only guarantees delivery to at least one node. On other low level networks it may be another matter and a validation may be required.

In a static design it is quite easy to make certain the event reach its destinations. One just program the responses expected and then resend if the node does not confirm the control event.

In a dynamic environment a self learning mechanism may be needed. The controlling node can check the device type for nodes in its landscape and fill in a binary map with info about nodes that should react on an event. This map can then be used as a means to confirm responses. It is a maximum of 16 bytes on a bus without hard set nicknames 32 bytes otherwise.

Something like this

1. Heartbeat. Node 8 is a lamp. My button A can control that lamp. Set bit 7 (offset 0) in bit-array.
2. Heartbeat. Node 2 is a lamp. My button A can control that lamp. Set bit 1 in bit array. And so on.
3. Button A is pressed. Send CLASS1.CONTROL, Type=5, TurnOn event to zone/sub-zone. The controlling node makes a copy of the bit-array.
4. Node 2 respond with CLASS1.INFORMATION, Type=3, ON and the controlling node set bit 1 (offset 0) in its copy of the bit-array. And it check if the array looks the same as the original. It does not so continue to wait.
5. Node 8 respond with CLASS1.INFORMATION, Type=3, ON and the controlling node set bit 7 in its copy of the array. It again looks if the arrays are equal and they are so the operation is completed.
6. Another scenario can be that Node 8 did not respond. In this case the press button procedure will timeout and the missed positions (not all) can be retransmitted.

So what happen when a node is removed? A reset/learn button on the controlling unit that starts to learn is easiest but can be self learning here also so that a node that never is responding is reported to be in error or it can even be checked.²

31.2 How to receive VSCP events using C

```

1  ****
2  * Copyright (C) 2006 by Ake Hedman
3  * a k h e @ g r o d a n s p a r a d i s . c o m
4  *
5  * Part of VSCP project, http://www.vscp.org
6  *
7  *
8  *
9  * This program is free software; you can redistribute it and/or modify
10 * it under the terms of the GNU General Public License as published by
11 * the Free Software Foundation; either version 2 of the License, or
12 * (at your option) any later version.

```

²I struggled a lot with the above for decision matrix elements. Here it must be dynamic and configurable. A quite elegant solution to this problem is described here 7.3 on page 80. Send event conditional. This can be used in lower end nodes also where a bit-array can be used instead.

```

13 *
14 * This program is distributed in the hope that it will be useful,
15 * but WITHOUT ANY WARRANTY; without even the implied warranty of
16 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
17 * GNU General Public License for more details.
18 *
19 * You should have received a copy of the GNU General Public License
20 * along with this program; if not, write to the
21 * Free Software Foundation, Inc.,
22 * 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.
23 ****
24 #include <stdio.h>
25 #include <sys/socket.h>
26 #include <fcntl.h>
27 #include <arpa/inet.h>
28 #include <stdlib.h>
29 #include <string.h>
30 #include <unistd.h>
31 #define USERNAME "admin" // logon username
32 #define PASSWORD "secret" // logon password
33 #define RCVBUFSIZE 512 // TCP/IP received buffer size
34 #define GUID "0:1:2:3:4:5:6:7:8:9:10:11:12:13:14:15" // GUID to use for i/f
35
36 // States for data parsing
37 #define STATE_HEAD 0
38 #define STATE_CLASS 1
39 #define STATE_TYPE 2
40 #define STATE_OBID 3
41 #define STATE_TIMESTAMP 4
42 #define STATE_GUID 5
43 #define STATE_DATA 6
44
45 // Prototypes
46 void DieWithError( char *errorMessage );
47 int getResponse( int sock, char *buf );
48
49 ///////////////////////////////////////////////////
50 // main
51 //
52 // address [port]
53 //
54
55 int main( int argc, char *argv[] )
56 {
57     int i,j;
58     char *p;
59     int sock; // Socket descriptor
60     struct sockaddr_in ServerAddr; // Server address
61     unsigned short ServerPort; // Server port
62     char *szServerIP; // Server IP address (dotted quad)
63     char buf[RCVBUFSIZE]; // Receivebuffer
64     char wrkbuf[512]; // Buffer for general work
65     int bytesRcvd, totalBytesRcvd; // Bytes read in single recv() and total bytes read
66     int nEvents; // # events waiting to be received
67     unsigned char guid[16]; // Storage for GUID
68     int state;
69     unsigned int vscp_head; // vscp_head
70     unsigned int vscp_class; // vscp_class
71     unsigned int vscp_type; // vscp_type
72     unsigned long vscp_obid; // vscp_obid
73     unsigned long vscp_timestamp; // vscp_timestamp
74     char strguid[512]; // strguid
75     unsigned char data[512-25]; // data
76     int cntData;
77
78     printf("VSCP_client_TCP/IP_Event_receive_example.\n");
79     printf("-----\n");
80

```

```

81     printf("Use ctrl-C to end.\n");
82
83 // Test for correct number of arguments
84 if ( ( argc < 2 ) || ( argc > 3 ) ) {
85     fprintf( stderr, "Usage: %s<Server IP>[port]\n", argv[0] );
86     exit(1);
87 }
88
89 szServerIP = argv[ 1 ]; // server IP address (dotted quad)
90 if ( 3 == argc ) {
91     ServerPort = atoi( argv[ 2 ] ); // Use given port, if any
92 }
93 else {
94     ServerPort = 9598; // VSCP server standard port
95 } // Create a a socket
96
97 if ( ( sock = socket( PF_INET, SOCK_STREAM, IPPROTO_TCP ) ) < 0 ) {
98     DieWithError("socket() failed");
99 }
100
101 // Non blocking
102 //if ( fcntl( sock, F_SETFL, O_NDELAY ) < 0 ) {
103 //    DieWithError("fcntl F_SETFL, O_NDELAY");
104 //} // Construct the server address structure
105 memset( &ServerAddr, 0, sizeof( ServerAddr ) );
106 ServerAddr.sin_family = AF_INET;
107 ServerAddr.sin_addr.s_addr = inet_addr( szServerIP );
108 ServerAddr.sin_port = htons( ServerPort );
109
110 // Establish the connection to the server
111 if ( connect( sock,
112             ( struct sockaddr * ) &ServerAddr,
113             sizeof( ServerAddr ) ) < 0 ) {
114     DieWithError("connect() failed");
115 }
116
117 // Check to see that we are connected
118 memset( buf, 0, sizeof( buf ) );
119 if ( ( bytesRcvd = recv( sock, buf, RCVBUFSIZE - 1, 0 ) ) <= 0 ) {
120     DieWithError("recv() failed or connection closed prematurely");
121 }
122
123 if ( NULL == strstr( buf, "OK" ) ) {
124     DieWithError("Failed to connect to server");
125 }
126
127 // * * * Set username * * *
128 // Send the username to the server
129 sprintf( buf, "user%ss\r\n", USERNAME );
130 if ( send( sock, buf, strlen( buf ), 0 ) != strlen( buf ) ) {
131     DieWithError("Failed to send username.");
132 }
133
134 // Get response
135 memset( buf, 0, sizeof( buf ) );
136 if ( ( bytesRcvd = recv( sock, buf, RCVBUFSIZE - 1, 0 ) ) <= 0 ) {
137     DieWithError("Failed to set username. No response.");
138 }
139
140 // Check to see if username was accepted
141 if ( NULL == strstr( buf, "OK" ) ) {
142     DieWithError("Username not accepted.");
143 }
144
145 // * * * Set password * * *
146 // Send password to the server
147 sprintf( buf, "pass%ss\r\n", PASSWORD );
148 if ( send( sock, buf, strlen( buf ), 0 ) != strlen( buf ) ) {

```

```

149     DieWithError( "Failed_to_send_password." );
150 }
151
152 // Get response
153 memset( buf, 0, sizeof( buf ) );
154 if ( ( bytesRcvd = recv( sock, buf, RCVBUFSIZE - 1, 0 ) ) <= 0 ) {
155     DieWithError("Failed_to_set_password.No_response.");
156 }
157
158 // Check to see if username was accepted
159 if ( NULL == strstr( buf, "OK" ) ) {
160     DieWithError("Password.wa_not_accepted");
161 }
162
163 // * * * Set interface GUID * * *
164 // This is not needed but added here to make the example complete.
165 // Set interface GUID
166 sprintf( buf, "SGID=%s\r\n", GUID );
167 if ( send( sock, buf, strlen( buf ), 0 ) != strlen( buf ) ) {
168     DieWithError("Failed_to_send_SGUID_command.");
169 }
170
171 // Get response
172 memset( buf, 0, sizeof( buf ) );
173 if ( ( bytesRcvd = recv( sock, buf, RCVBUFSIZE - 1, 0 ) ) <= 0 ) {
174     DieWithError("Failed_to_set_interface_GUID.No_response.");
175 }
176
177 // Check to see if the set GUID command was accepted
178 if ( NULL == strstr( buf, "OK" ) ) {
179     printf("Interface_GUID_not_accepted.Use_default_GUID_instead.");
180 }
181
182 while ( 1 ) {
183
184     // Check if data is available
185     memset( buf, 0, RCVBUFSIZE );
186     sprintf( buf, "CDTA\r\n" );
187     if ( send( sock, buf, strlen( buf ), 0 ) != strlen( buf ) ) {
188         DieWithError("Failed_to_send_CDTA_command.");
189     }
190
191     // Get response
192     memset( buf, 0, RCVBUFSIZE );
193     if ( ( bytesRcvd = recv( sock, buf, RCVBUFSIZE - 1, 0 ) ) <= 0 ) {
194         DieWithError("Failed_to_check_for_data.No_response.");
195     }
196
197     // Check # events waiting
198     nEvents = atoi( buf );
199     if ( nEvents ) {
200
201         printf("Event(s) available\n");
202
203         // Fetch event(s)
204         for ( i=0; i<nEvents; i++ ) {
205
206             // fetch data
207             memset( buf, 0, RCVBUFSIZE );
208             sprintf( buf, "RETR\r\n" );
209             if ( send( sock, buf, strlen( buf ), 0 ) != strlen( buf ) ) {
210                 DieWithError("Failed_to_send_RETR_command.");
211             }
212
213             getResponse( sock, buf );
214             // Check to see if the event was received OK
215             if ( NULL == strstr( buf, "OK" ) ) {
216                 printf("Event was not received OK. Response=%s", buf );

```

```

217     }
218
219     // Data received is packed as:
220     // head, class, type, obid, timestamp,
221     // GUID, data0, data1, data2, .....
222     // we have to parse the data.
223     //
224
225     state = STATE_HEAD;
226     cntData = 0;
227     p = strtok( buf, ",," );
228     while( p != NULL ) {
229
230         switch ( state++ ) {
231
232             case STATE_HEAD:
233                 // Get head
234                 vscp_head = atoi( p );
235                 break;
236
237             case STATE_CLASS:
238                 // Get class
239                 vscp_class = atoi( p );
240                 break;
241
242             case STATE_TYPE:
243                 // Get type
244                 vscp_type = atoi( p );
245                 break;
246
247             case STATE_OBID:
248                 // Get obid
249                 vscp_obid = atol( p );
250                 break;
251
252             case STATE_TIMESTAMP:
253                 // Get timestamp
254                 vscp_timestamp = atol( p );
255                 break;
256
257             case STATE_GUID:
258                 // Get GUID string
259                 strcpy( strguid, p );
260                 break;
261
262             default:
263                 // data
264                 data[ cntData ] = atoi( p );
265                 if ( cntData > (512-25) ) p = NULL;;
266                 // end it all
267                 cntData++;
268                 break;
269         } // switch
270
271         p = strtok( NULL, ",," );
272
273     } // while
274
275
276
277     // Extract the GUID
278     memset( guid, 0, sizeof(guid) );
279     p = strtok( strguid, ":" );
280     for ( j=0; j<16; j++ ) {
281         guid[ j ] = atoi( p );
282         p = strtok( NULL, ":" );
283         if ( NULL == p ) break;
284     }

```

```

285         // Report received event
286         printf("Head=%d,obid=%lu,timestamp=%lu\n",
287                vscp_head,
288                vscp_obid,
289                vscp_timestamp );
290
291         printf("CLASS=%d,TYPE=%d\n",
292                vscp_class,
293                vscp_type );
294
295         printf("GUID=");
296         for ( j=0; j<16; j++ ) {
297             printf( "%d", guid[ j ] );
298         }
299
300         printf("MSB->LSB\n");
301         printf("DATA=");
302         for ( j=0; j<cntData; j++ ) {
303             printf("%d", data[ j ] );
304         }
305
306         printf("\n");
307
308     } // for
309 }
310 else {
311     usleep( 100 );
312 } // events waiting
313
314 }
315
316 close( sock );
317 return EXIT_SUCCESS;
318 }
319
320 ///////////////////////////////////////////////////////////////////
321 // getResponse
322 //
323 //
324
325 int getResponse( int sock , char *buf )
326 {
327     char respbuf[ RCVBUFSIZE ];
328     int totcnt = 0;
329     int cnt;
330
331     do {
332         if ( ( cnt = recv( sock , respbuf , RCVBUFSIZE - 1 , MSG_PEEK ) ) <= 0 ) {
333             return 0;
334         }
335         else {
336             if ( ( cnt = recv( sock , respbuf , RCVBUFSIZE - 1 , 0 ) ) <= 0 ) {
337                 return 0;
338             }
339             memcpy( buf+totcnt , respbuf , cnt );
340             totcnt += cnt;
341             if ( NULL != strstr( buf , "+OK" ) ) break;
342             if ( NULL != strstr( buf , "-OK" ) ) break;
343         }
344     }
345     while ( cnt );
346
347     return totcnt;
348 }
349
350 ///////////////////////////////////////////////////////////////////
351 // DieWithError
352

```

```

353 // 
354 // 
355 void DieWithError(char *errorMessage)
356 {
357     perror( errorMessage );
358     exit( 1 );
359 }

```

31.3 How to send an event from C

```

1 ///////////////////////////////////////////////////////////////////
2 // Simple VSCP TCP/IP sample
3 // 
4 // Code skeleton fetched from
5 // http://cs.baylor.edu/~donahoo/practical/CSockets/textcode.html
6 // 
7 // akhe@grodansparadis.com - http://www.vscp.org
8 // 
9
10 #include <stdio.h>
11 #include <sys/socket.h>
12 #include <arpa/inet.h>
13 #include <stdlib.h>
14 #include <string.h>
15 #include <unistd.h>
16 #define USERNAME "admin"          // logon username
17 #define PASSWORD "secret"        // logon password
18 #define GUID "0:1:2:3:4:5:6:7:8:9:10:11:12:13:14:15" // G UID to use
19 #define RCVBUFSIZE 2048 // TCP/IP received buffer size
20
21 // Prototypes
22 void DieWithError(char *errorMessage); /* Error handling function */
23
24 ///////////////////////////////////////////////////////////////////
25 // main
26 // address class type data [port]
27 ///////////////////////////////////////////////////////////////////
28
29
30 int main( int argc, char *argv[] )
31 {
32     int sock;                                // Socket descriptor
33     struct sockaddr_in ServerAddr; // Server address
34     unsigned short ServerPort;    // Server port
35     char *szServerIP;                // Server IP address (dotted quad)
36     char *szClass;                  // VSCP class
37     char *szType;                   // VSCP type
38     char *szData;                   // VSCP data
39     char buf[ RCVBUFSIZE ];
40     int bytesRcvd, totalBytesRcvd; // Bytes read in single recv()
41                                     // and total bytes read
42
43     // Test for correct number of arguments
44     if ( ( argc < 5 ) || ( argc > 6 ) ) {
45         fprintf( stderr,
46             "Usage: %s<Server_IP><class><type><data1,data2,data3,...>.[port]\n",
47             argv[0] );
48         exit(1);
49     }
50
51     szServerIP = argv[ 1 ];           // server IP address (dotted quad)
52     szClass = argv[ 2 ];            // VSCP class
53     szType = argv[ 3 ];
54     szData = argv[ 4 ];
55     if ( 6 == argc ) {
56         ServerPort = atoi( argv[ 5 ] ); // Use given port, if any
57     }

```

```

58     else {
59         ServerPort = 9598; // VSCP server standard port
60     } // Create a a socket
61
62     if ( ( sock = socket( PF_INET, SOCK_STREAM, IPPROTO_TCP ) ) < 0 ) {
63         DieWithError("socket() failed");
64     }
65
66     // Construct the server address structure
67     memset( &ServerAddr, 0, sizeof( ServerAddr ) );
68     ServerAddr.sin_family = AF_INET;
69     ServerAddr.sin_addr.s_addr = inet_addr( szServerIP );
70     // Establish the connection to the echo server
71     ServerAddr.sin_port = htons( ServerPort );
72     if ( connect( sock,
73                     ( struct sockaddr * ) &ServerAddr,
74                     sizeof( ServerAddr ) ) < 0 ) {
75         DieWithError("connect() failed");
76     }
77
78     // Check to see that we are connected
79     if ( ( bytesRcvd = recv( sock, buf, RCVBUFSIZE - 1, 0 ) ) <= 0 ) {
80         DieWithError("recv() failed or connection closed prematurely");
81     }
82     if ( NULL == strstr( buf, "OK" ) ) {
83         DieWithError("Failed to connect to server");
84     }
85
86     // * * * Set username * *
87     // Send the username to the server
88     sprintf( buf, "user% s\r\n", USERNAME );
89     if ( send( sock, buf, strlen( buf ), 0 ) != strlen( buf ) ) {
90         DieWithError("Failed to send username.");
91     }
92     // Get response
93     if ( ( bytesRcvd = recv( sock, buf, RCVBUFSIZE - 1, 0 ) ) <= 0 ) {
94         DieWithError("Failed to set username. No response.");
95     }
96     // Check to see if username was accepted
97     if ( NULL == strstr( buf, "OK" ) ) {
98         DieWithError("Username not accepted.");
99     }
100    // * * * Set password * *
101    // Send password to the server
102    sprintf( buf, "pass% s\r\n", PASSWORD );
103    if ( send( sock, buf, strlen( buf ), 0 ) != strlen( buf ) ) {
104        DieWithError("Failed to send password.");
105    }
106    // Get response
107    if ( ( bytesRcvd = recv( sock, buf, RCVBUFSIZE - 1, 0 ) ) <= 0 ) {
108        DieWithError("Failed to set password. No response.");
109    }
110    // Check to see if password was accepted
111    if ( NULL == strstr( buf, "OK" ) ) {
112        DieWithError("Password was not accepted.");
113    }
114    // * * * Set interface GUID * *
115    // Set interface GUID
116    sprintf( buf, "SGID% s\r\n", GUID );
117    if ( send( sock, buf, strlen( buf ), 0 ) != strlen( buf ) ) {
118        DieWithError("Failed to send SGID command.");
119    }
120    // Get response
121    if ( ( bytesRcvd = recv( sock, buf, RCVBUFSIZE - 1, 0 ) ) <= 0 ) {
122        DieWithError("Failed to set interface GUID. No response.");
123    }
124    // Check to see if the set GUID command was accepted
125    if ( NULL == strstr( buf, "OK" ) ) {

```

```

126             printf("Interface_GUID not accepted. Use default GUID instead.");
127         }
128         // * * * Send Event * * *
129         // Send event
130         sprintf( buf,
131                 "SEND_0,%s,%s,0,0,-,%s\r\n",
132                 szClass,
133                 szType,
134                 szData );
135         if ( send( sock, buf, strlen( buf ), 0 ) != strlen( buf ) ) {
136             DieWithError("Failed to send event");
137         }
138         // Get response
139         if ( ( bytesRcvd = recv( sock, buf, RCVBUFSIZE - 1, 0 ) ) <= 0 ) {
140             DieWithError("Failed to send event. No response.");
141         }
142         // Check to see if the send event was accepted
143         if ( NULL == strstr( buf, "OK" ) ) {
144             printf("Send event command failed..");
145         }
146         close(sock);
147         exit(0);
148     }
149 }
150 ///////////////////////////////////////////////////////////////////
151 // DieWithError
152 ///////////////////////////////////////////////////////////////////
153 //
154 //
155 void DieWithError(char *errorMessage)
156 {
157     perror(errorMessage);
158     exit(1);
159 }

```

31.4 How do I send an event from Perl

31.4.1 VSCP TCP Connection

The following code use the TCP interface of the daemon

```

1 use IO::Socket;
2 printf("________________________________\n");
3 printf("_____Very Simple Perl Test\n");
4 printf("________________________________\n");
5 my $sock = new IO::Socket::INET(
6                                         PeerAddr =>
7                                         'localhost',
8                                         PeerPort => '9598',
9                                         Proto => 'tcp'
10                                        );
11 die "Could not create socket : $!\n" unless $sock;
12
13 recv( $sock, $buffer, 200, 0 );
14 print $buffer;
15
16 print $sock "user admin\n";
17 recv( $sock, $buffer, 200, 0 );
18 print $buffer;
19
20 print $sock "pass secret\n";
21 recv( $sock, $buffer, 200, 0 );
22 print $buffer;
23
24 #
25

```

```

26  # Send a full GUID event
27  #
28  # Format is
29  #      "send head, class, type, o bid, timestamp, GUID, data1, data2, data3 . . . "
30  #
31  # Class=20  CLASS1.INFORMATION
32  # TYPE = 3  ON
33  # o bid = 0
34  # timestamp = 0
35  # GUID = 0:1:2:3:4:5:6:7:8:9:10:11:12:13:14:15    on the form MSB->LSB
36  # Zone=1
37  # SubZone=80
38  #
39  print $sock "SEND_0,20,3,0,0,0:1:2:3:4:5:6:7:8:9:10:11:12:13:14:15,0,1,80\n";
40  recv( $sock, $buffer, 200, 0);
41  print $buffer;
42  #
43  #
44  # The same thing can be sent using
45  #
46  # SEND 0,20,3,0,0,-,0,1,80
47  #
48  # where the interface GUID is used.
49
50  print $sock "SEND_0,20,3,0,0,-,0,1,80\n";
51  recv( $sock, $buffer, 200, 0);
52  print $buffer;
53  close($sock);

```

No command verification is done in this sample. Obviously this should be added. Just look for a '+' in the response for an OK return value.

31.5 How do I send an event from PHP

There are two ways to do this. Either you connect to the TCP port or send a VSCP broadcast or connect to the VSCP daemon and send your event over the TCP/IP interface.

31.5.1 VSCP Broadcast

This code snippet send a CLASS1.INFORMATION TYPE=3 ON Event using an UDP datagram.

```

1  <?php
2      $socket = socket_create(AF_INET, SOCK_DGRAM, SOL_UDP);
3      socket_set_option($socket,
4                          SOL_SOCKET, // socket level
5                          SO_BROADCAST,
6                          1 );
7
8      // create the request packet
9      $head = 0;
10     $class = 20;
11     // LEVEL I Information
12     $type = 4;
13     // ON Event
14     // We use GUID for LAB usage
15     $GUID[ 0 ] = 0;
16     $GUID[ 1 ] = 0;
17     $GUID[ 2 ] = 0;
18     $GUID[ 3 ] = 0;

```

```

19     $GUID[ 4 ] = 0;
20     $GUID[ 5 ] = 0;
21     $GUID[ 6 ] = 0;
22     $GUID[ 7 ] = 0;
23     $GUID[ 8 ] = 0;
24     $GUID[ 9 ] = 0;
25     $GUID[ 10 ] = 0;
26     $GUID[ 11 ] = 0;
27     $GUID[ 12 ] = 0;
28     $GUID[ 13 ] = 0;
29     $GUID[ 14 ] = 0;
30     $GUID[ 15 ] = 0;
31     $size = 3;
32     $data[ 0 ] = 0;
33     $data[ 1 ] = 1;
34     // Zone 1
35     $data[ 2 ] = 80;
36     // Sub-zone 80
37     $crc = 0;
38     $packet = chr( $head ) .
39             chr( $class / 256 ) .
40             chr( $class & 255 ) .
41             chr( $type / 256 ) .
42             chr( $type & 255 );
43     for ( $i=0; $i<16; $i++ ) {
44         $packet = $packet . chr( $GUID[ $i ] );
45     }
46     $packet = $packet . chr( $size / 256 ) . chr( $size & 255 );
47     for ( $i=0; $i<$size; $i++ ) {
48         $packet = $packet . chr( $data[ $i ] );
49     }
50     $crc = calculate_common_crc16c( $packet );
51     $packet = $packet . chr( $crc / 256 ) . chr( $crc & 255 );
52     // UDP is connectionless, so we just send on it.
53     socket_sendto($socket, $packet, ( 25 + $size ), 0x100, "255.255.255.255", 9598 );
54     // this function is used to calculate the CCITT crc16c
55     // for an entire buffer function
56     calculate_common_crc16c( $buffer ) {
57         $crc16c = 0xFFFF; // the crc initial value
58         $buffer_length = strlen( $buffer );
59         for ( $i = 0; $i < $buffer_length; $i++ ) {
60             $ch = ord( $buffer[$i] );
61             $crc16c = update_common_crc16c( $ch, $crc16c );
62         }
63     }
64 ?>

```

31.5.2 VSCP TCP Connection

The following code does the same thing as above but use the TCP interface of the daemon

```

1  <?php
2      printf("<h1><h2>TCP/IP Connection</h2></h1>");
3      $service_port = getservbyname( 'www', 'tcp' );
4      $service_port = 9598;
5      $address = gethostbyname( '192.168.1.6' );
6      $socket = socket_create( AF_INET, SOCK_STREAM, 0 );
7      if ( $socket < 0 ) {
8          echo "socket_create() failed : reason : " .
9              socket_strerror( $socket ) . "\n" . "<br>";
10     }
11     else {
12         echo "OK.\n" . "<br>";
13     }
14     echo "Attempting to connect to '$address' on port '$service_port' ...";

```

```

15     $result = socket_connect ($socket , $address , $service_port);
16     if ($result < 0) {
17         echo "socket_connect() failed.\nReason: ($result)\n" .
18             socket_strerror($result) . "\n" . "<br>";
19     } else {
20         echo "OK.\n" . "<br>";
21     }
22     // Read response
23     $out = socket_read ($socket , 2048);
24     echo $out . "<br>";
25     $in = "USER_admin\n";
26     echo "Sending username..." ;
27     socket_write ($socket , $in , strlen ($in));
28     echo "OK.\n" . "<br>";
29
30     // Read response
31     $out = socket_read ($socket , 2048);
32     echo $out . "<br>";
33     $in = "PASS_secret\n";
34     echo "Sending password..." ;
35     socket_write ($socket , $in , strlen ($in));
36     echo "OK.\n" . "<br>";
37     // Read response
38     $out = socket_read ($socket , 2048);
39     echo $out . "<br>";
40     // send head, class, type, obid, timestamp, GUID, data1, data2, data3 . . .
41     // class=20 - Information
42     // Type=4 - OFF
43     // Zone=1
44     // SubZone=80
45     $in = "SEND_0,20,4,0,0,0:0:0:0:0:0:0:0:0:0:0:0:0,0,1,80\n";
46     echo "Sending CLASS1_INFORMATION_TYPE=ON . . .";
47     socket_write ($socket , $in , strlen ($in));
48     echo "OK.\n" . "<br>";
49     echo "Closing socket . . .";
50     socket_close ($socket);
51     echo "OK.\n\n";
52 ?>

```

31.6 How do I send an event from Python

31.6.1 VSCP TCP Connection

The following code use the TCP interface of the daemon

```

1 #!/usr/bin/env python
2 import socket
3 import sys
4 import urllib
5
6 url = "http://tinyurl.com/temperaturLos"
7 #Url till yr.no:s rss-feed
8
9 opener = urllib.FancyURLopener({})
10 page = opener.open(url)
11
12 for line in page.readlines():
13     if "m/s" in line:
14         line = line.split(".")
15         temperatur = line[1][-5].strip()
16         vindstyrka_list = line[3].split(",")
17         vindstyrka_list = vindstyrka_list[1].split("fra")
18         vindstyrka = vindstyrka_list[0][-4].strip()
19         break
20
21 # debug output

```

```

22 print "Temperature=" + temperatur + " grader celsius"
23 print "Vindstyrka=" + vindstyrka + " m/s"
24
25 # create socket
26 s = socket.socket( socket.AF_INET, socket.SOCK_STREAM )
27
28 # connect to server
29 host = "dosilos.se" port = 9598
30 s.connect( ( host, port ) )
31 data = s.recv( 10000 )
32 print data
33
34 s.send( "user=admin\r\n" )
35 data = s.recv( 10000 )
36 print data
37
38 s.send( "pass=secret\r\n" )
39 data = s.recv( 10000 )
40 print data
41
42 #
43 # Send a full GUID event
44 # =====
45 # Format is
46 # "send head, class, type, o bid, timestamp, GUID, data1, data2, data3 . . . "
47 #
48 # Class=20 CLASS1.INFORMATION
49 # TYPE = 3 ON
50 # o bid = 0
51 # timestamp = 0
52 # GUID = 0:1:2:3:4:5:6:7:8:9:10:11:12:13:14:15 on the form MSB->LSB
53 # Zone=1
54 # SubZone=80
55 #
56
57 s.send( "SEND_0,20,3,0,0,0:1:2:3:4:5:6:7:8:9:10:11:12:13:14:15,0,1,80\r\n" )
58 data = s.recv( 10000 )
59 print data
60
61 #
62 # The same thing can be sent using
63 #
64 # SEND 0,20,3,0,0,-,0,1,80
65 #
66 # where the interface GUID is used.
67
68 s.send( "SEND_0,20,3,0,0,-,0,1,80\r\n" )
69 data = s.recv( 10000 )
70 print data
71
72 # close connection
73 s.close();

```

Part XI

FAQ

32 VSCP FAQ

32.1 General

32.1.1 Q: Where does CANAL end and VSCP start?

CANAL is CAN abstraction layer. It is a generic interface to interface any CAN hardware adapter.

The VSCP/canal daemon uses CANAL to interface hardware drivers. This way the daemon (or any other application software that wants to use the device) has one interface for drivers with very different properties.

VSCP Level I events fits perfectly in a CAN frame by design. Therefore it is possible to write drivers that interface diverse things such as WINLIRC (IR receivers/transmitters), xAP, SSH remote links, RS-232, X10 and more.

The driver is a “dll” or a “so” under Unix/Linux. The interface is well known and common to all and you can therefore build software that can work the same way on different hardware devices.

The VSCP daemon exports a CANAL interface to all drivers connected to the daemon. A wrapper class is available that lets you write software that looks the same regardless if it is using the daemon or talks directly to a driver.

32.1.2 Q: Does it cost money to use VSCP?

VSCP is both open and free and there is no charge for its usage. You are also allowed to use the code and change it in any way you like. Some code are GPL/LGPL which means you have to share your changes. Some code are free for all uses.

32.1.3 Q: Can I make commercial applications from the code?

There are three licenses used in this project. GPL, LGPL and freeware. If you use and change GPLed code you have to share your own changes and code as well. For LGPL code you are allowed to use it in your proprietary code without charge and need to reveal your code. The freeware code can be used in any way you like. Freeware is used for most general VSCP client routines and for all firmware code.

If you don't like this there is a commercial license available from eurosouce <akhe@eurosouce.se>

32.1.4 Q: I need a vendor ID. How do I get it?

Send a mail to guid@vscp.org with your contact information (mail, phone, web) and we will return a GUID to you as soon as possible. There is no cost for this

service.

32.1.5 Q: I can't find anything about routing between segments in the specifications.

As of inter segment communication it can be complicated and easy.

One approach is to have a level II bus that holds segments together. When events pass the segment barrier from the Level I segment nicknames are translated into full GUID's. When GUID's is received that is on the segment that a specific "segment boundary device" (segment controller) is on they are transferred to the local segment with the GUID replaced by the nickname. The segment controller thus has to know the GUID's of the devices on its segment. Not a hard thing even dynamically.

Another and simpler approach is to let the "segment controller" take the event from the segment. Replace the nickname with its own nickname-ID and send it out on the other segment(s). Much like a proxy.

32.1.6 Q: Ouch!!! I try to understand all this but its just to much. How do I start?

Its no need to understand everything at least not when one start to experiment with VSCP.

There is only one concept that is important to understand really. That is:

Each node can inform others with the help of events (which are well specified). It normally does not matter if anyone listens or just talks. Sometimes this talk is pretty certain to reach its destination, (CAN, TCP/IP), sometimes it don't know (UDP,RF etc) and from VSCP point of view it does not care.

After that

1. All nodes are identified by a 128-bit Globally unique ID, GUID.
2. 128 8-bit registers are defined by each node that have predefined contents (not all are used). See 6 on page 75
3. Up to 128 8-bit registers can be defined for a nodes functionality/configuration. See 6 on page 75
4. Nodes can have a decision matrix built in. This matrix can be programmed to make the node react on external events on the form Event - Decision - Action. 7.3 on page 80
5. Every node got a Module Description File (MDF) that describes its registers, events, decision matrix capabilities etc in a higher level language (XML). The location for this file can be fetched from the registers. See 9 on page 88 There is a real life example here http://www.eurosource.se/sht_001.xml.

That's it really. It all boils down to

1. Sending events of predefined class + Type (we together adding events/- classes as needs arise (or an interest group can own its own classes and do this themselves) (Currently defined stuff here. See 12 on page 107
2. Reading/writing registers.

By thinking this way you can have a node such as the temperature node that triggers other stuff without knowing it and to which things can be added in a plug an play fashion where only the decision matrix needs to be manipulated at the new nodes and no master is needed but can be incorporated without affecting the rest of the system.

32.1.7 Q: Zone/sub-zone. What are they used for?

As everything in VSCP (as for CAN) is broad casted zone/sub-zone can be used to “address” a group of nodes. This way it is possible to send for example a mute command to them. Several other are here 12.7 on page 165. Some events should also respond with events that contain the modules zone/subzone for example 12.6 on page 152

32.1.8 Q: I don't understand the event naming convention. Please explain.

When we write CLASS1.ALARM we refer to a Level I event in the ALARM (CLASS=1) group (12.2 on page 130).

In the same way CLASS1.PROTOCOL can be found here 12.1 on page 107

The naming convention used is like this CLASS1 refers to a Level I event. CLASS2 refers to a Level II event. .xxxxx Refers to events in the class (listed at the top of each page).

So

CLASS1.INFORMATION is here 12.6 on page 152

and

CLASS2.INFORMATION is here 13.3 on page 220

So to find CLASS1.PROTOCOL Type 37 look at the page 12.1 on page 107

32.1.9 Q: Must my device understand all of them events?

No. Your code just need to understand the events in CLASS1.PROTOCOL and this is handled by the ready made firmware code. Then you might want to handle some other events of your choice or perhaps implement a decision matrix so you give the user of your device the decision to decide which event that should trigger a specific functionality (action) of your device.

32.2 VSCP over CAN

32.2.1 Q: What transfer speed does VSCP use?

125 kbps is the standard bit-rate over CAN. Other bit-rates can be used but all nodes must support 125 kbps but it is not recommended.

32.2.2 Q: How long cable can be used?

This is dependent on the speed but if using the standard speed of 125 kbps the cable can be max 500 meters. The same timing as for CANopen is used and more info can be found here <http://www.canopen.org/canopen/protocol/bittiming.html>.

32.2.3 Q: Can VSCP and CANopen be used on the same bus?

Yes! VSCP uses extended 29 bit addressing and CANopen used standard 11-bit addressing.

32.3 VSCP and GUID

32.3.1 Q: What is the GUID?

GUID stands for *Globally Unique ID* and is for VSCP a number consisting of 16 bytes (128 bits) and this number should be unique throughout the world for every node in a VSCP system. No two nodes should have the same GUID. This is much the same as two nodes on an Ethernet don't have the same MAC address.

32.3.2 Q: Do i have to pay for a vendor ID?

No it's free of charge. When you ask for a vendor ID you get a 12 byte number that represents the first 12 bytes of the complete 16-bit GUID. The remaining four bytes can be used in any way you like which gives you 4294967295 possibilities. Typical this is used as a serial number for your device.

32.3.3 Q: From where can I get my own VSCP GUID?

Write to guid@vscp.org and we will register your GUID. Please add your mail and postal address to the mail.

32.3.4 Q: Do I have to be a big company to get my own VSCP GUID?

No everyone can get there own ID.

32.3.5 Q: Our big big big company need more then just 4294967295 unique GUID's. Can we get more?

Yes write to guid@vscp.org and tell us why you need more GUID's (4294967295 is a lot!!!) and you can get an additional vendor ID or an extra byte (that is 1099511627775 nodes).

32.3.6 Q: I already have a MAC address or some other vendor ID. Can I use that as the basis for a GUID?

Yes! There are several well known IDs that are reserved and can be used without asking us for a GUID. See 5 on page 72for more information.

32.3.7 Q: How is the GUID from a device connected to the daemon organized?

Normally the first found Ethernet MAC address of the computer where the daemon runs used as the basis for this GUID. The least significant byte is the nickname-ID for the node sending the event and the next least significant byte is the ID for the driver interface this node is at.

The actual GUID used can be set in the configuration file. Also here the least significant byte will be the nickname-ID and the next least significant byte the interface ID.

Example

```
FF FF FF FF FF FF FF FE 00 05 5D 8C 02 00 02 01
```

For this GUID the Ethernet MAC address has been used and a node with nickname-ID = 1 connected to device interface with ID 2 sent the event.

32.4 VSCP Level I

32.4.1 Q: I want to use the same nickname for a faulty module that is replaced. How do I do that?

There are two methods to achieve this.

Method 1 Configure the new node with the old nickname before adding it to the segment. It will start up using the same nickname as the old module.

Method 2 Remove the faulty module and add the new one. It will find a new nickname-address when it starts up and goes through the nickname discovery process. As the nickname is in the >128 register space we can just write a new nickname there using the write register events.

```
class=CLASS1_PROTOCOL type=Write Register  
data byte 0 = old nickname data byte 1 = 0x91 data byte 2 =  
new nickname
```

The response should come from the “old node”.

```
class=CLASS1_PROTOCOL type=Read/Write response  
data byte 0 = 0x91 data byte 1 = new nickname
```

After this the node will use the new nickname on the segment.

32.4.2 Q: How is modules designed that contains combined functionality?

VSCP uses GUID for addressing. Depending on the environment it is running in other identification tokens - as the nickname on Level I segments - can be used. Only one rule applies and that is that the GUID should be possible to deduce from the id-token.

Nodes can be grouped by zone/sub-zone. Both are values from 0-255 where 255 have the meaning “all”. It’s up to the user/installer to decide how they are used. Typically zone can be a house, a room or something and sub-zone a sub part of that. Note that this is not addresses. Any number of nodes can have the same zone/sub-zone.

If the zone/sub-zone concept is used it is preferably possible to configure them through some registers.

Some nodes have functionality that holds a combination of different functionality. Typical examples are a light detector and a relay, eight input module etc. Also we have another group measurement devices that can have several sensors.

For the combined devices the zone is typically used for the device and the sub-zone select the individual functionality. By doing this we can create decision matrix elements that do something like this

- Do this action if this event is for zone/sub-zone.

Optionally also the originating nickname can be used in the selection.

If a node identifies similar subgroups with the sub-zone member something like this can be used

- Do this action if this event is for this zone and sub-zone as argument.

Again the nickname can be used also.

The user can thus program the device and tell it on what event and for which zones/sub-zones it should react.

For measurement devices things can be a bit different. This is because this type of devices got a sensor index already built into the measurement event. This is coded into the data coding byte that is present for all measurement events (8 on page 85). Three bits are available so up to eight sensors can be handled by one device.

The Paris module (http://www.vscp.org/wiki/doku.php?id=smart_relay_control_intelligent_relay_control_module) is a reference installation where the combined concept is shown and the Kelvin Smart II module (http://www.vscp.org/wiki/doku.php?id=kelvin_smart_temperature_times_2_module) shows a multi sensor module.

32.4.3 Q: My application needs more then 128 registers. How do I solve that?

Look at registers 0x92/0x93 here 6 on page 75

they select which page of registers you map into register address space 0x00 - 0x7F. For nodes that just need one page they have no meaning for a node that have more then 128 registers they have to be used.

Usage is First set the page - read/write register 0x92/x93.

Two events are available to take care of this in an atomic way. Look at CLASS1.PROTOCOL, Type=36 etc

32.5 VSCP Level II

32.5.1 Q: How is data laid out in Level II over UDP?

See 2.6 on page 62

32.5.2 Q: How are filters/masks used?

The following table illustrates how the filters works

Mask bit n	Mask 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.

32.6 The VSCP daemon

32.6.1 Q: Windows: The VSCP daemon is started but nothing works.

If you have have started the daemon and try to communicate using two instances in CanalWorks for example or the testif application and you get a an error telling you that no server is found this may be due to the canald service started without a privileged user. This is fully described here canalservice_documentation.

32.6.2 Q: How are events routed by the daemon?

This is somewhat work in progress therefore there are two answers to this question. How it works today and how it will work.

An event from a low end device hooked to the driver interface will go out to the client interface as a Level I event and out on the UDP interface as a Level II event with the GUID of the low level interface OR the real GUID of the device if it's known.

A Level I event sent on the client interface will be sent on all low level interfaces and also on the UDP interface as a Level II event with GUID of the client interface if not CLASS.PROTOCOL. If it is CLASS.PROTOCOL and has a GUID the system knows of it will be sent only to that device. If not know old behavior is used.

A Level II event sent on the client interface will be sent out on the UDP interface as it is. The user decide if the interface GUID or some other GUID should be used. Same as before.

A Level II event received from the UDP interface will be sent to all clients.

A Level I event received from the UDP interface will be sent to all clients and all devices if not CLASS.PROTOCOL. If it is CLASS.PROTOCOL and has a GUID the system knows of it will be sent only to that device. If not know old behavior is used.

32.6.3 Q: I get error about a missing shared library when I try to start vscpd under Linux/Unix.

You have to add the shared library to your systems search path. A good explanation is available here

<http://www.eyrie.org/~eagle/notes/rpath.html>

32.6.4 Q: I use the logger driver under Unix and get two events in the log for every sent.

This can be due to a miss configured hostname. Check your hosts file. The 127.0.0.1 entry should be something like

127.0.0.1 localhost localhost.localdomain

The reason you get two events logged is that the daemon can't determine that the UDP event it sends out is coming from your machine and therefore receives its own events also. You can see this as the low eight bits of the ID always is set to zero for the extra event.

33 CANAL FAQ

33.1 General Drivers

33.1.1 How to get going with driver x on windows.

There are a couple of ways to do this.

Install the latest VSCP & Friends package from <http://sourceforge.net/projects/m2m/files/>. Remember to select at least the driver you want to install and the logger driver. You can of course choose all of them if you want as it is perfectly fine to mix them all. Available drivers are described here VIII on page 442

Now you have to select what component you prefer to use of VSCP & Friends.

One way is to only use VSCP Works which is available on both Linux and WIN32 and log data in a window. In that case you connect *directly to the driver*.

The Other way is to use the vscpd (the daemon which also is available both on Linux and WIN32). In this case you can have as many drivers from different manufacturers active at the same time as you like. You also can access data over TCP/IP and UDP.

VSCP Works

1. Open VSCP Works.
2. Select “New session” under the file menu or the blank page icon to the left.
3. You can select the CANAL daemon interface here before any other interface is configured but this connect to the daemon see the B steps. You have to set up a driver. Do this by clicking add...
4. Set a “Driver Name” that works for you.
5. Set the driver path. During the VSCP & Friends installation the driver will be installed in the */windows/system32* folder. Select the file wanted driver from there.
6. The device Configuration string is a string that configures a specific driver. It is documented for each driver here VIII on page 442 Also fill in flags ad other information.
7. Click OK to save the adapter.
8. No select the requested adapter and click “OK” The window is open but you are not connected.
9. Click “Activate i/f” to open a connection.

You can now define other channels or other drivers and open as many windows as you like.

Using the VSCP Daemon The VSCP daemon can have any number of drivers connected at the same time. This also mean that you can talk to and receive from all of them using VSCP Works, the TCP/IP or other tool. How to add a driver to the daemon is described here 14.4 on page 240

Part XII

Appendix

Appendix A - Assigned GUID's

01 00 00 00 00 00 00 00 00 00 00 00 00 00 XX XX XX XX XX eurosource
Brattbergavägen 17 820 50 LOS Sweden Phone: +46 (0)8 40011835

Email: info@eurosource.se Web: <http://www.eurosource.se>

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

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

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

05 00 00 00 00 00 00 00 00 00 00 00 00 00 XX XX XX XX XX RTist BV Asterlaan
18 3871 CB Hoevelaken The Netherlands Phone: +31-6-5331-0072
Email: robtu@rtist.nl Web: <http://www.rtist.nl>

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

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

08 00 00 00 00 00 00 00 00 00 00 00 00 00 XX XX XX XX XX Total Telematics
Tigh-Na-Bruach 67 Evan Street Stonehaven AB39 2HR
Great Britain Phone: +44 771 266 7629
Email: info@total-telematics.com Web: <http://total-telematics.com>

09 00 00 00 00 00 00 00 00 00 00 00 XX XX XX XX Gediminas Simanskis Phone: +37069912663 Vilnius , Lithuania
Email: info@edevices.lt Web: <http://www.edevices.lt>

0A 00 00 00 00 00 00 00 00 00 00 00 XX XX XX XX Hideo "genie" Kobayashi Email: genie@netsynth.org
<http://netsynth.org/>, Japan

0B 00 00 00 00 00 00 00 00 00 00 00 XX XX XX XX CAN-West Controls Henk Hofstra Houtweg 9 1815 DP Alkmaar The Netherlands
Phone: +31725150781 Email: Henk-Hofstra@orange.nl
Web: <http://can-west.nl>

0C 00 00 00 00 00 00 00 00 00 00 00 XX XX XX XX PROEL S.p.A Tiziano Tosi Via alla Ruenia,37/43 64027 Sant'Omoro (Teramo) Italy
Phone: +39086181241 Web: www.proelgroup.com

0D 00 00 00 00 00 00 00 00 00 00 00 XX XX XX XX Deltatec – Tecnologia de Sistemas Lda José Semedo Av. Ivens, nº 16 4º Dto Dafundo 1495-725 Cruz Quebrada Portugal
Email: jsemedo (at) deltatec.pt Web: <http://www.deltatec.pt>

0E 00 00 00 00 00 00 00 00 00 00 00 XX XX XX XX AZYL PPHU mr Paweł Gatner Litewska 46/13 51-354 WROCLAW POLAND,
email: azyl (at) azylnet.com

0F 00 00 00 00 00 00 00 00 00 00 00 XX XX XX XX John Voigt, Reston VA USA

10 00 00 00 00 00 00 00 00 00 00 00 XX XX XX XX Marcelo Poma Av. José Sorrento 439 S2013SOC – Rosario ARGENTINA

11 00 00 00 00 00 00 00 00 00 00 00 XX XX XX XX Jiri Kubias, Addat s.r.o. U krematoria 24 Liberec 1 46001 Czech republic
email: addat (at) addat.cz

12 00 00 00 00 00 00 00 00 00 00 00 XX XX XX XX Werner & Associates, LLC, P. O. Box 620, Finksburg, Maryland,
21048-00620 USA, email: wwerner (at) qis.net,
Web: <http://www.werner-associates.com>

13 00 00 00 00 00 00 00 00 00 00 00 XX XX XX XX Neil Wrightson. N.W.Electronics Skype: Neil_Wrightson Web: www.nwe.net.au

14 00 00 00 00 00 00 00 00 00 00 00 XX XX XX XX UnixMedia -
Media Convergence Solutions, Milano - Italy -
Email: info (at) unixmedia.it , Web: <http://www.unixmedia.it>

1C 00 00 00 00 00 00 00 00 00 00 00 XX XX XX XX Kiril Petrov,
Sofia, Bulgaria, ice (at) geomi.org as my email address, and <http://geomi.org>

1E 00 00 00 00 00 00 00 00 00 00 00 XX XX XX XX IP COMMUNICATIONS JSC, Ho Chi Minh City, Vietnam,
thacl (at) ipcoms.com, and www.ipcoms.com

1F 00 00 00 00 00 00 00 00 00 00 00 XX XX XX XX sglux SolGel
Technologies GmbH, Max-Planck-Str. 3, Berlin, Germany,
langen@sglux.de, <http://www.sglux.de>

20 00 00 00 00 00 00 00 00 00 00 00 XX XX XX XX Ignite Solutions
D-46, Plot-27, Sec-10, Dwarka, New Delhi - 110075, India,
Phone: +91-8826779814 <http://www.ignite-solutions.in/>

21 00 00 00 00 00 00 00 00 00 00 00 XX XX XX XX Davide Bortolini
Treviso, Italy, davide@bortolini.it

22 00 00 00 00 00 00 00 00 00 00 00 XX XX XX XX SKA Polska Sp.
z o.o. Al. Jerozolimskie 125/127, p. 406 02-017 Warszawa Poland
tel./fax: +48 22 632 17 75 www: www.ska-polska.pl email: info (at) ska-polska.pl

23 00 00 00 00 00 00 00 00 00 00 00 XX XX XX XX Steven J. Ackerman,
Consultant // ACS, Sarasota, Florida, USA
tel./fax: +1 (941)377-5775 www: <http://www.acscontrol.com> email: steve
(at) acscontrol.com

24 00 00 00 00 00 00 00 00 00 00 00 XX XX XX XX Josef Melvald,
Sydney, Australia email: [xmexjox \(at\) yahoo.com.au](mailto:xmexjox(at)yahoo.com.au)

25 00 00 00 00 00 00 00 00 00 00 00 XX XX XX XX

26 00 00 00 00 00 00 00 00 00 00 00 XX XX XX XX

27 00 00 00 00 00 00 00 00 00 00 00 XX XX XX XX

28 00 00 00 00 00 00 00 00 00 00 00 XX XX XX XX

29 00 00 00 00 00 00 00 00 00 00 00 XX XX XX XX

30 00 00 00 00 00 00 00 00 00 00 00 XX XX XX XX

31 00 00 00 00 00 00 00 00 00 00 00 XX XX XX XX

32 00 00 00 00 00 00 00 00 00 00 00 XX XX XX XX Grodans Paradis
AB, Brattbergavägen 17 820 50 LOS Sweden Phone: +46 (0)8 40011835
Email: info@grodansparadis.com Web: Grodans Paradis AB

33 00 00 00 00 00 00 00 00 00 00 00 XX XX XX XX Arpad Toth,
Hungary.

DE 00 00 00 00 00 00 00 00 00 00 00 00 XX XX XX XX Stefan Langer -
datenheim.de, Germany, st.langer (at) gmx.net