Notes for changes in Microchips 18F CAN bootloader (AN247) and the vscpboot application

Ake Hedman, akhe@eurosource.se
rev 0.3: 2004-09-22

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

- Message format has been changed to suit VSCP protocol.
- Messages from the node uses the node id in the origin field.
- The node sends an initial message at startup.
- 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 multiplatforms (Windows, Unix and in the future Macintosh)

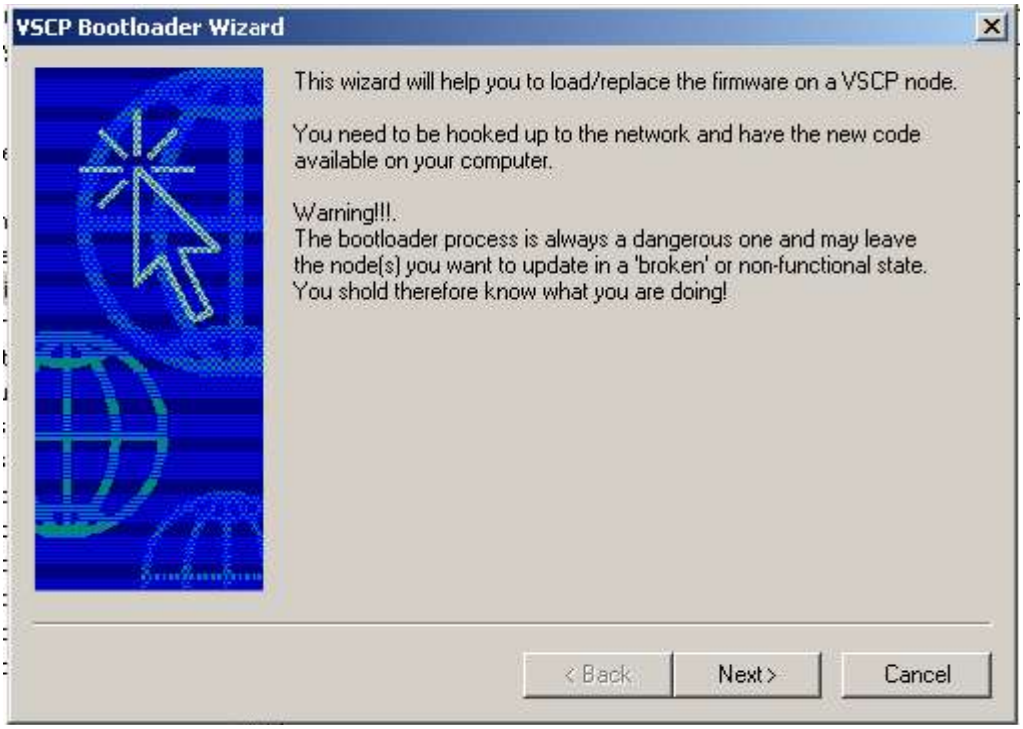
vscpboot application

The VSCP application is a simple wizard to bootload a node in a VSCP environment. It should be no problem to adopt it for your own needs. It is written in C++ and use the wxWidgets library to be portable to other platforms. The source is in the VSCP source distribution or in the VSCP CVS.

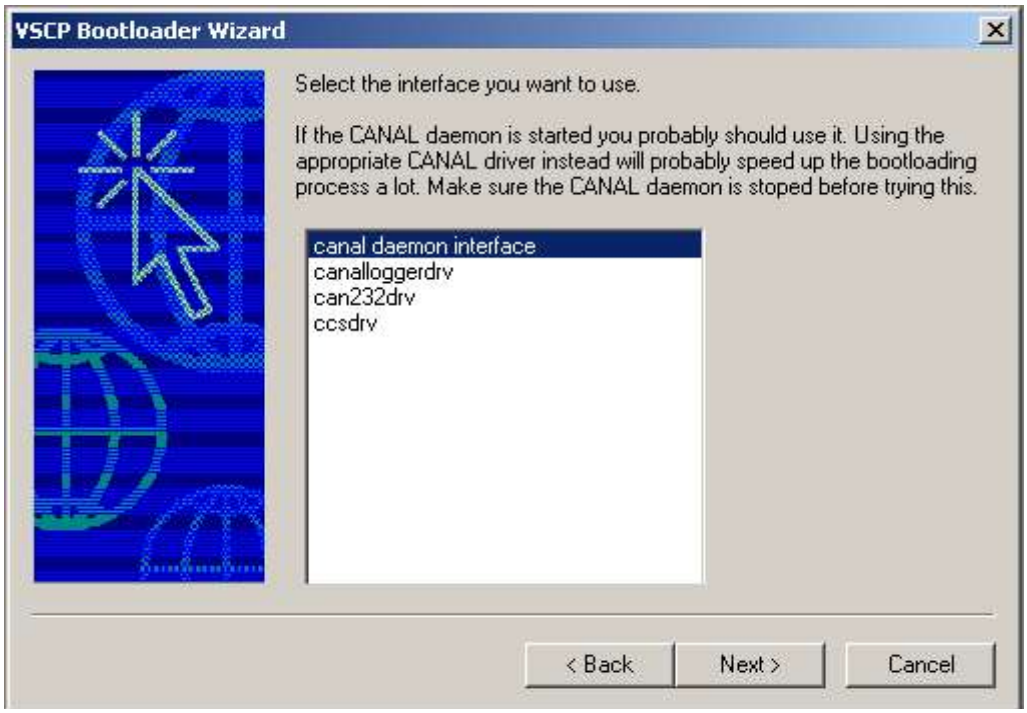
You need to have the CANAL service started or at least have one CANAL driver available to be able to use the application. If you need to design your own CANAL driver please check the CANAL documentation on how to do so. This is a very easy task.

How to use it

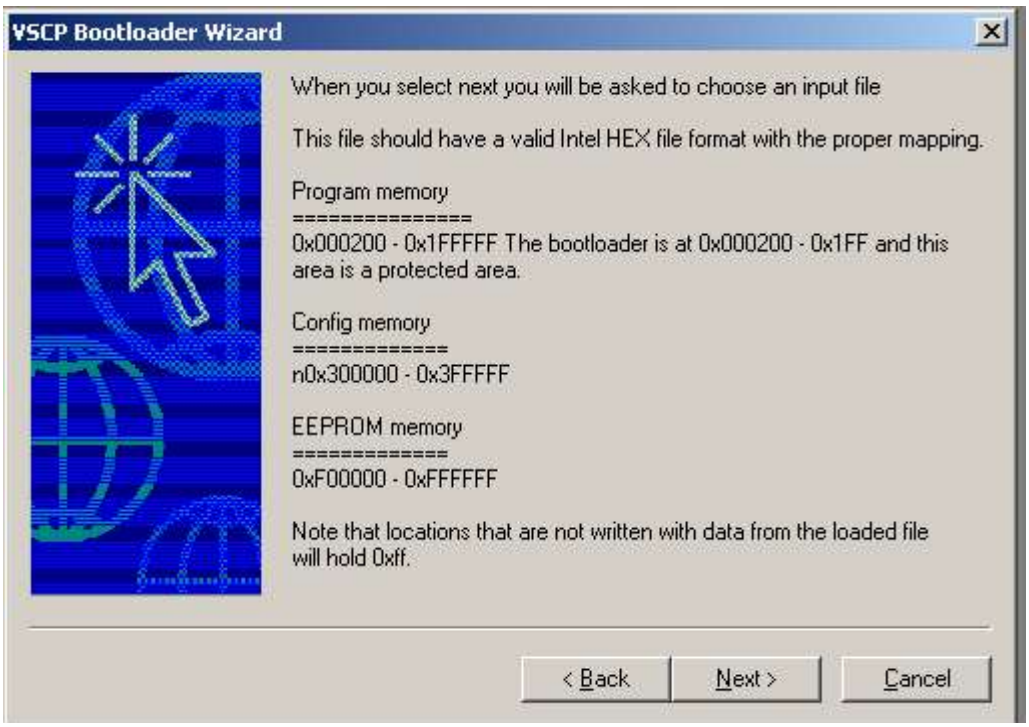
Start the vscpboot.exe application wizard. The welcome screen will be displayed.



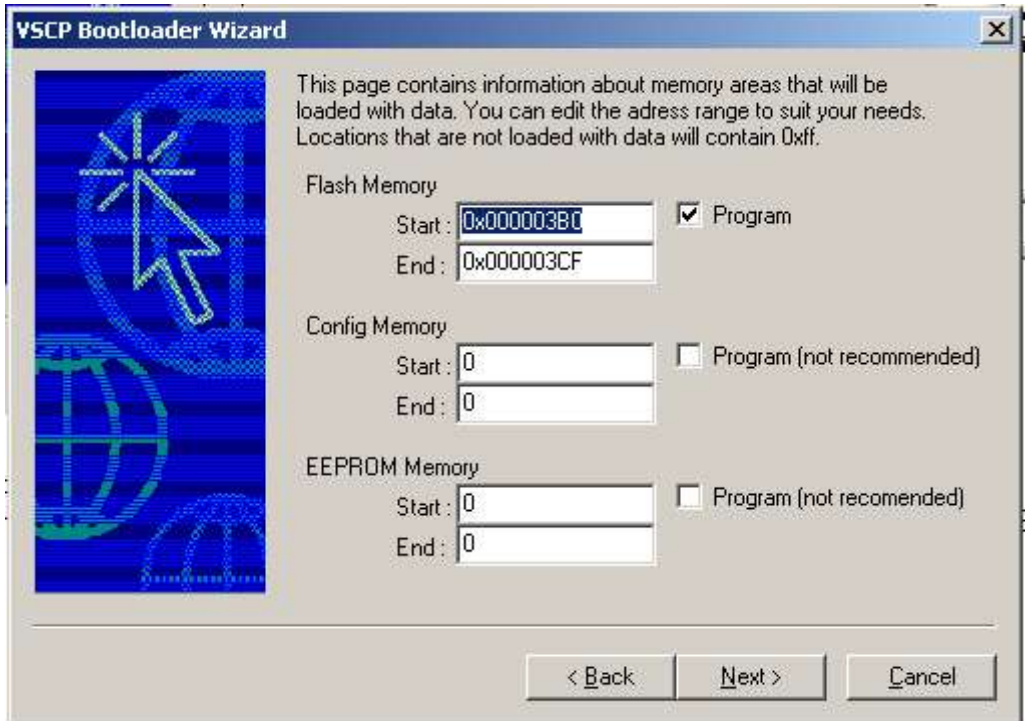
Press next to move forward



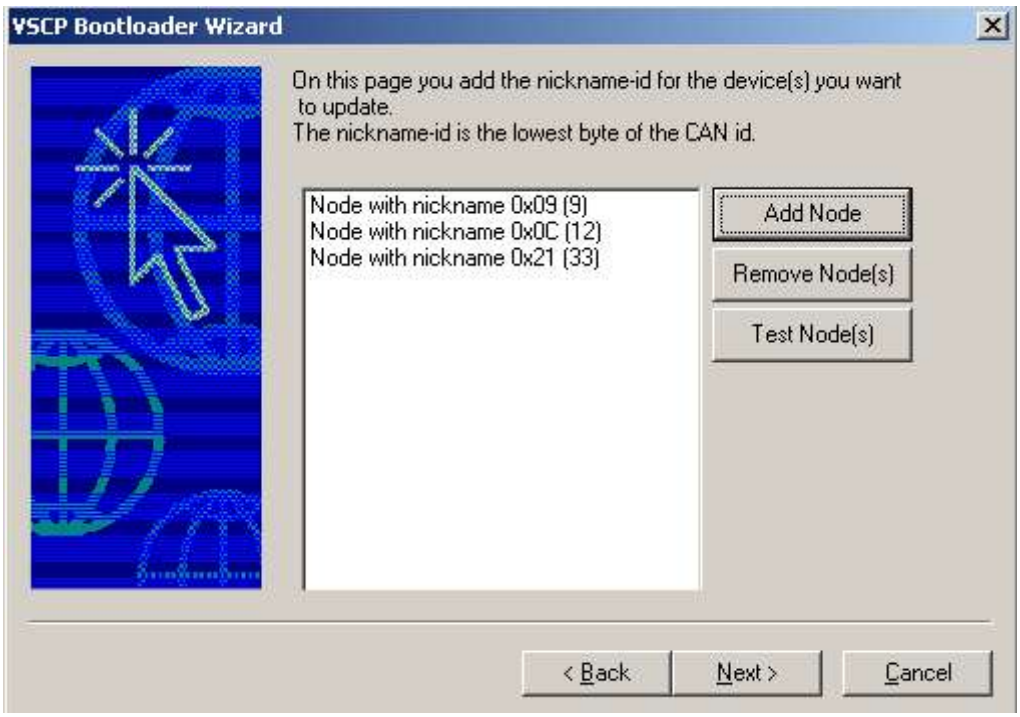
Select the interface you want to communicate through. It i sfaster to go directly through a driver then to use the CANAL daemon. If the listbox is blank you need to install CANAL and/or one or more CANAL drivers. See the documentation for CANAL on how to do this.



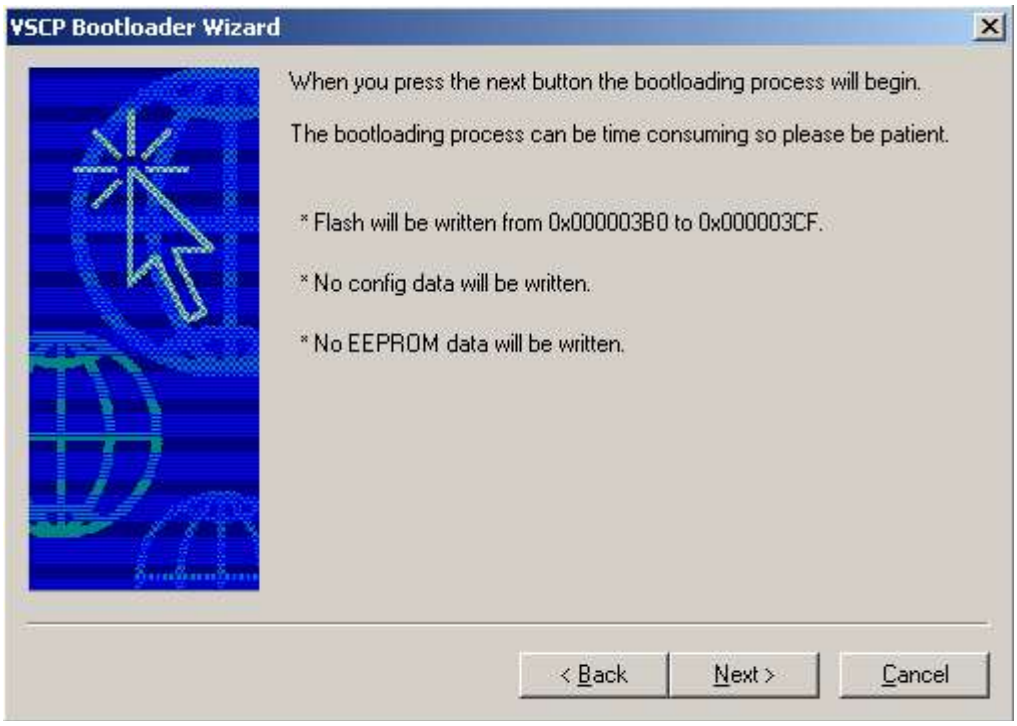
This screen tells how the input file is interpreted and loaded. The input file should be a standard Intel Hex file. When you press next you are asked to select the file and load it. If everything is OK with the file the next screen will be shown.



Here you have the ability to choose which areas should be loaded. You also can change the address range. If you try to load data into the boot sector a warning will be issued. Also to write configuration bits are very dangerous. Likewise for VSCP nodes it is not good to write to EEPROM memory. In boot cases you should know what you are doing and understand that you cn end up with a node that does not work if you write the wrong information to the wrong place.



On this screen you choose the nodes you want to load data to.



This is the last screen before the actual bootloading process starts. Just press next and wait until the code has been loaded. If everything goes well the node will reboot into the new code after the bootloading and start the new application firmware.

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, this bootloader type is supplied 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..

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 rested. This is usually how the first command to the node is formated.

2.) Write data to the device by sending multiple **PUT_DATA** commands to it.Add each byte sent to the node. This is 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 bootinfo to start-application state. If the checksum fails it will stay in bootloade mode.

The VSCP bootloader wizard handles this task.

Filter and Mask settings

If the filter allows messages for all nicknames to come in to the node it is possible to bootload many nodes at the same time. The node will answere with its node id but only react on packages for node 0xff and its own id. In that way it is possible to have part of the programming common for all nodes and parts specific.

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 hardcoded or not.

Filter

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

Responses are of type 20 and 21.

PUT_BASE_INFO

Priority = 0
Hardcode = 0
Class = 0
Type = 16

Command id is **0x000010nn** where **nn** is the nickname for the noe that initiated the bootload.

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 write and erase to memory.
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.
0c03: 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 autoincrement 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 not write set MODE_ERASE_ONLY

if MODE_AUTO_INC the memory pointer will increment automatically.

PUT_DATA

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

Command id is **0x000011nn** where **nn** is the nickname for the noe that initiated the bootload.

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.

GET_BASE_INFO

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

Command id is **0x000012nn** where **nn** is the nickname for the noe that initiated the bootload.

This command get 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.

GET_DATA

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

Command id is **0x000013nn** where **nn** is the nickname for the noe that initiated the bootload.

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

The respons 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 don't want to share memory content can report all data as **0xff**.

References

The Microchip application note AN 247 is the basis work for this application work. It and its accompaning source can be found at http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=1999&ty=&dty=&section=&NextRow=&ssUserText=AN247&x=7&y=8&DesignDocSelect=

The latest version of this document, CANAL, VSCP, drivers etc can be found at http://www.vscp.org

The wxWidgets library can be found at http://wxwidgets.org/