VLCB Versatile Layout Control Bus

PIC Bootloader Service Specification
Service# 10 PIC BOOT

Version 3.2, June 2025, for Service version 3

Compatible with CBUS ® 4.0 Rev 8j

0.1 Table of Contents

0.2 Document History

Date	Changed by	Summary of changes	Service version
23rd December 2022	Ian Hogg M.5144	Initial document	1
14 April 2023	Ian Hogg M.5144	Changed name to VLCB	1
31 July 2024	Ian Hogg M.5144	Improvements to description of the bootloader protocol	1

7 November 2024	Ian Hogg M.5144	Addition of bootloader versioning	2
23 March 2025	Ian Hogg M.5144	Added clarification of push button handling recommendations as Section 13.	2
14 April 2025	Ian Hogg M.5144	Removed the MODE command leaving BOOTM opcode.	3
11 June 2025	Ian Hogg M.5144	Added additional explanation and recommendations on programming of unspecified addresses within Flash and EEPROM space	3

1 VLCB Services

This document describes the VLCB Bootloader service. This service is an optional addition to the VLCB Minimum Node Specification.

This service is

- BOOT for the CBUS compatible bootloader to allow VLCB firmware to be downloaded to existing CBUS modules.

It is anticipated that another Service will be available

- BOOT2 for new, transport independent protocol.

Application firmware must be able to be loaded using either protocol/service.

Modules wishing to use the existing CBUS PIC bootloader shall conform to this specification.

2 Bootloading

FCU (and other bootloading programs) and the bootloader firmware communicate using the bootloader protocol.

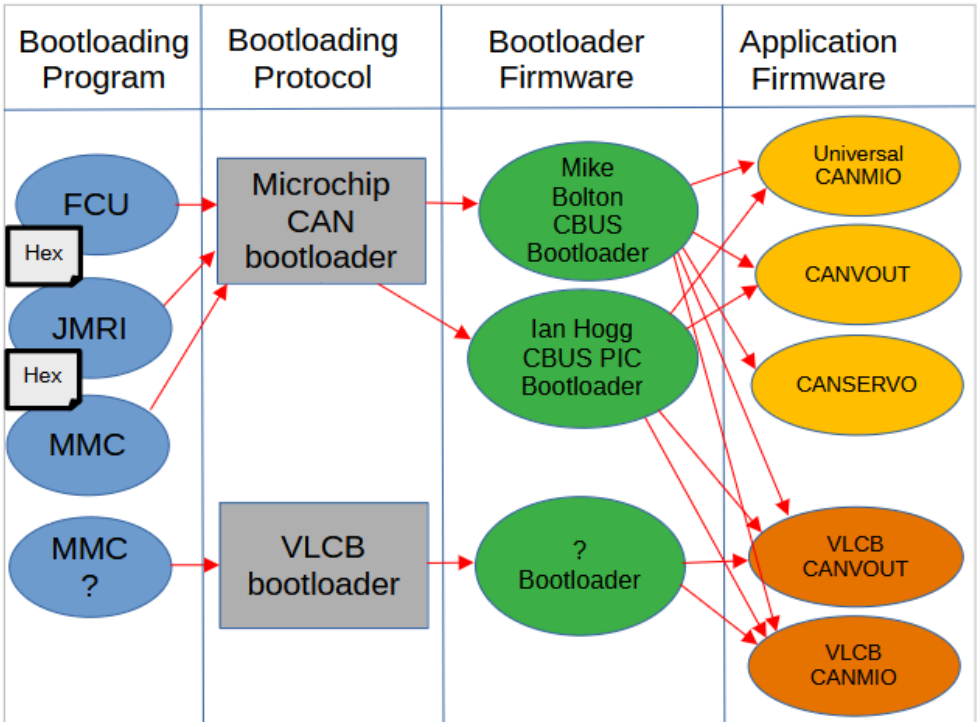
The end users and module applications have little control over the bootloader firmware – it is unchanged when loading module firmware using the bootloader. The current bootloader firmware will already be present on CBUS modules.

The PIC bootloader firmware uses the upper byte of the address to distinguish between Config, EEPROM and Flash but this is not enforced nor required by the bootloader protocol. The use of the top byte of the address is processor dependent.

The Bootloader code resides in the first 2K of flash program memory (0x0000 ~ 0x7FF). At power on the bootloader runs first and checks the BOOT flag byte at the top byte of EEPROM. If this byte is clear then control is transferred to the application at its loadAddress otherwise control continues within the module's Bootloader. The bootloader awaits and processes bootloader messages. It is able to write to application flash memory (0x800 upwards), EEPROM memory and CONFIG settings.

The bootloader itself is never overwritten with this process.

There are a number of different components at work with bootloading and each needs to work with other components. The diagram below shows the components and their interworking.



The CBUS PIC bootloader protocol is described in Appendix A.

2.1 Dependencies on other services

The BOOT service depends upon the mandatory Minimum Module Service.

3 Additional requirements for compatibility with FCU

FCU is the most widely used program used to download firmware images to modules. FCU places additional requirements upon the module. Other downloader programs such as JMRI and LayoutMesh exist and these may have their own additional requirements.

In order to be compatible with the FCU Bootloading program the application must comply with the following requirements:

- Be presented as a single file in Intel hex format. See https://en.wikipedia.org/wiki/Intel_HEX,
- Hex file to contain the parameter block between 0x820~0x83F.

FCU attempts to check compatibility between the module and the hex file. It tries to check:

- The correct processor family.
- The module to have or exceed the required FLASH memory size.
- Have the bootable flag set in bit 3 of parameter 8 returned by RQNPN.
- Support the BOOTM opcode which sets the bootflag and then resets the processor to enter the bootloader.

In order to allow FCU check for compatibility it is also recommended to have

- The CPU manufacturer set in parameter 19 and at address 0x832 of the hex file.
- Have the processor set in parameter 9 and at address 0x828 of the hex file.
- Have the LoadAddress at 0x82A of the hex file.
- Return the actual CPU type using parameters 15~18.

4 Loading the bootloader firmware

The bootloader firmware must not allow itself to be overwritten by the bootloading process therefore to bootstrap the bootloader firmware it must be loaded another way such as using the ICSP connector or downloading a separate application which is able to overwrite the Bootloader.

5 Summary of Opcodes

Refer to the VLCB Opcode Specification document for details of the opcodes.

Request to Module	Module's Response	Use/meaning
BOOTM		Enter bootloader

6 Service Specific Modes

None.

7 Service Specific Status Codes

No additional GRSP status codes are specified by the Bootloader service.

8 Service Specific Diagnostic Data

No additional RDGN diagnostic data numbers are specified by the Bootloader service.

9 Service Specific Automatic Power-up Tests

No service specific power-up tests are specified by the Bootloader service.

10 Bootloader Versioning

Since the module's application can be updated independently of the module's bootloader and that different bootloaders exist with different capabilities it is useful for the loading program to be able to determine the capabilities of the module's bootloader before starting any bootloading process.

Version 2 of the BOOT service adds the capability to return the module's bootloader type and version within the ESD bytes for the service. The service must interrogate the

bootloader to determine its type and version and must not rely upon the version of the bootloader with which it was originally distributed.

Bootloaders supporting versioning should embed the fixed string “BL_VERSION=” within its program memory followed by the two bytes for the bootloader type and bootloader version. In C this can be achieved as in this example:

```
#define BL_VERSION  1

const char bl_version[] = {
'B','L','_','V','E','R','S','I','O','N','=', BL_TYPE_IanHogg,
BL_VERSION};
```

To determine the bootloader’s type and version to return in the ESD bytes the BOOT service can then perform a search of the bootloader’s memory 0x000~0x7FF for the fixed string and return the following two bytes for the type and version of the bootloader.

11 Documentation

The module’s documentation must state if it can be downloaded by the existing CBUS bootloader firmware.

The module’s documentation must state if it is compiled with a BOOT Service compatible bootloader firmware.

The module’s documentation must state if each version is compatible with previous versions’ EEPROM data.

12 Service data

12.1 Parameters

To be compatible with the CBUS PIC bootloader within the FCU it is necessary to locate the parameter block starting at address 0x0820 within the module's hex memory image file.

The following parameters are associated with the PIC Bootloader service and to be compatible with FCU bootloading program:

Address	Param#	Name	Usage	VLCB should set these values
0x820	1			
0x821	2			
0x822	3	ModuleId	Module type identifier. See cbusdefs.h	maybe a single ID for all VLCB e.g. 0xFC.
0x823	4			
0x824	5			
0x825	6			
0x826	7			
0x827	8.3	Bootable	Indicates if FCU can use the CBUS PIC bootloading protocol	Set to 1 if the requirements of the FCU bootloading process are met.
0x828	9	Processor Id		Type of CPU, 15=PIC18F2K80. See cbusdefs
0x828	10			
0x82A~ 0x82D	11~14	Load Address	The start address of the application code.	0x0800[1]
0x82E~ 0x831	15~18	Processor code	PICs support the ability to determine the processor type (DEVID)	Hex file contains 0x0. RQNPN to read processor code dynamically from hardware.

[1] Load Address: According to the bootloader protocol the application firmware can specify any start address however the current CBUS PIC based bootloader firmware assumes an address of 0x800.

0x832	19	Manufacturer code		1 for microchip, 2 for Atmel, 3 for Arm
0x833	20			
0x834	21			
0x835	22			
0x836	23			
0x837	24			
0x838~ 0x839		Number of parameters		0x0014
0x83A~ 0x83D		Address of module name	The module name is stored as a left justified, padded with spaces 7 character ASCII string within the module address space. This is the address of the name and can be used within the hex file.	Little endian memory address of the module name
0x83E~ 0x83F		checksum		The 16 bit addition of bytes in parameter block 0x820 ~0x84D

12.2 ESD data bytes

The BOOT service version 1 does not provide any data within the ESD message, all data bytes are returned set to 0.

Version 2 of the BOOT service supports the following ESD bytes:

ESD byte	**Usage**
1	The type of the bootloader within the module
2	The version of the bootloader within the module
3	Unused set to 0

13 Push Button handling recommendations

The PIC bootloader should provide facilities to allow the bootloader process to be started even if the top byte of EEPROM is not 0xFF.

It should be possible to start the application with the push button held down or not held down.

The following process is recommended to determine whether the bootloader or the application is to be run:

If the top byte of EEPROM has a value of 0xFF then run the bootloader process.

If the PB is not pushed then run the module application.

While the PB is pushed {

If the PB has been held down for 2 seconds or more then run the application.

}

Run the bootloader.

Appendix A - PIC Bootloader protocol

Also see the MERG wiki https://merg.org.uk/merg_wiki/doku.php?id=cbus:bootloader.

A.1 Licence

The FCU bootloader for PIC processors is based upon that of Microchip AN247. Implementations are distributed under the Microchip license which requires that it is only to be used on Microchip hardware.

See page 15 of http://ww1.microchip.com/downloads/en/AppNotes/00247a.pdf for details of the Microchip licensing. This bootloader is distributed with the CBUS software as a "system library" as defined in section 1 of the GNU Public license and is not licensed under GNU or Creative Commons. You must conform to Microchip license terms in respect of the bootloader.

A.2 MMC/FCU BootLoader Process

The VLCB/CBUS implementation for the MMC and FCU configuration tools is as follows:

1. Upon selecting a node to be reloaded MMC/FCU will prompt for a hex file to be loaded.
2. The target processor of the hex file (Address 0x828) is compared with the module's processor type (RQNPN param 15) . A warning is generated if they are incompatible.
3. The FCU will send that node a BOOTM message.
4. The module will write 0xFF into the top most byte of EEPROM (the boot flag) and then cause a processor reset.
5. The bootloader code resides at address 0x0000 and will be executed at reset.
6. The bootloader checks the value of the boot flag and if it is 0x00 it will pass control to the module application code.
7. Otherwise the bootloader awaits receipt of extended CAN messages containing commands or data to be written to memory. The HEX files will contain any EEPROM and CONFIG data by default but the user can opt to not program these in the MMC/FCU settings. See section A.3 Bootloader CAN Message Format for details of the message formats.
8. Upon receiving the final bootload message MMC/FCU will send a RESET command and the module will verify the data, clear the boot flag and perform a reset itself.

When programming the Flash section that MMC/FCU will automatically fill any unused space between the lowest up to the highest specified Flash address with 0xFF. Therefore uninitialised data must occupy space **above** the program space.

The module and MMC/FCU will ignore any addresses within Flash below 0x800 as this is reserved for the bootloader, which shouldn't be overwritten by itself.

Any gaps in the Flash data will be filled with 0xFF.

When programming EEPROM FCU will always program the entire EEPROM space, filling any unspecified addresses with 0xFF.

When programming EEPROM MMC will program 16 bytes ‘around’ any bytes specified in the hex file. That is it will take the address of any byte specified in the hex file and program the 16 bytes between (address AND 0xFF0) to (address OR 0x00F) any unspecified bytes are set to 0xFF.

When using a PICkit the PIC is erased, setting all Flash and EEPROM to 0xFF, before writing the data as specified in the hex file.

A.3 Bootloader CAN Message Format

A.3.1 CAN Identifier

All Bootloader CAN messages use extended frames i.e. a 29 bit Extended ID is used instead of the Standard ID. Note that CAN hardware often continues to use the term SID for the lower 11 bits of the ID and use EID to refer to the upper 18 bits of the ID.

The upper 27 bits of the ID should be ignored and only the lower 2 bits of the ID are used thus:

ID	Usage
xxxxxxxxxxxxxxxxxxxxxxxxxxx00	Control frame request to module
xxxxxxxxxxxxxxxxxxxxxxxxxxx00	Response to control frame request from module
xxxxxxxxxxxxxxxxxxxxxxxxxxx01	Data to be written to module
xxxxxxxxxxxxxxxxxxxxxxxxxxx11	Data read from module

The DLC of the frame should be set to indicate the number of data bytes in the message (0 to 8).

A.3.2 Control Frame Request format

A Control frame request consists of 8 data bytes (DLC=8):

D0	D1	D2	D3	D4	D5	D6	D7
ADDRL	ADDRH	ADDRU	RESVD	CTLBT	SPCMD	CHKL	CHKH

Where:

- ADDRL - Bits 0 to 7 of the memory pointer.
- ADDRH - Bits 8 - 15 of the memory pointer.
- ADDRU - Bits 16 - 23 of the memory pointer.
- RESVD - Reserved for future use.
- CTLBT - Control bits. See table below for definitions
- SPCMD - Special command. See table below for definitions
- CPDTL - Bits 0 - 7 of 2s complement checksum
- CPDTH - Bits 8 - 15 of 2s complement checksum

The SPCMD byte determines the operation being requested.

All the Control Frame Requests always contain all 8 bytes however not all are used for all commands.

The module’s bootloader must save the values of the memory pointer (ADDRU, ADDRH, ADDRL) and the control bits (CTLBT) for subsequent data transfer.

Special Commands:

Command	Value	Usage	Bytes used	Response
CMD_NOP	0x00	Save the address provided.	ADDRL, ADDRH, ADDRU, CTLBT	None
CMD_RESET	0x01	Issue a soft reset after setting last EEPROM data to 0x00	ADDRL, ADDRH, ADDRU, CTLBT	None but module should return to normal VLCB operation
CMD_RST_CHKSM	0x02	Reset the checksum and status flags	ADDRL, ADDRH, ADDRU, CTLBT	None
CMD_VERIFY	0x03	Add the checksum to special data, verify that total is zero, respond with nok/ok	ADDRL, ADDRH, ADDRU, CTLBT, CHKH, CHKL	Response with 1 data byte. OK if the checksum is correct, NOK is the checksum failed or any other error was detected during data transfer.
CMD_BOOT_TEST	0x04	Just responds with a message frame back to verify boot mode	ADDRL, ADDRH, ADDRU, CTLBT	Response with 1 data byte with data BOOT.

The Control bits provide some additional control over the read/write data process:

Bit name	Bit position	Meaning
MODE_WRT_UNLCK	0x01	Set this to allow write and erase operations to memory.
MODE_ERASE_ONLY	0x02	Set this to only erase Program Memory on a put command. Must be on a Flash block boundary.
MODE_AUTO_ERASE	0x04	Set this to automatically erase Program Memory while writing data.
MODE_AUTO_INC	0x08	Set this to automatically increment the pointer after writing.
MODE_ACK	0x10	Set this to generate an acknowledge after a 'put' (PG Mode only)

A.3.3 Control Frame Response format

Responses have a single data byte (DLC=1):

NOK	0x00
OK	0x01
BOOT	0x02

A.3.4 Data frame format

A Data frame consists of 8 data bytes containing the data that is to be written or that which has been read:

D0	D1	D2	D3	D4	D5	D6	D7
DATA0	DATA1	DATA2	DATA3	DATA4	DATA5	DATA6	DATA7

A.5 Typical download message sequence

Dir	Message	Comments
		The user of the management system selects a module and a hex file to be loaded.
→	VLCB RQNPN	Management system requests parameters to obtain the module's CPU type so that the management system can validate the hex file for compatibility.
		The management system validates the CPU from the parameters with the CPU type within the hex file parameter block.
→	VLCB BOOTM	Management system sends a BOOTM request to put the module into bootloader mode.
		The module writes 1 into the top EEPROM location and performs a reset.
		The module restarts, enters the bootloader code and checks the top EEPROM location. Since the memory is non zero the module continues within the bootloader and awaits further bootloader commands.

→	Control, 0x000000 Boot test (AUTO_INC, AUTO_ERASE, UNLOCK) ID=bxxxxxxxxxxxxxxxxxxxxxxxxxxxx00 Data=0x00 00 00 00 0D 04 00 00	The Management system performs a boot test to check that the module has successfully entered bootloader mode.
←	Boot mode ID=bxxxxxxxxxxxxxxxxxxxxxxxxxxxx00 Data=0x02	The module responds to indicate that it is in bootloader mode.
→	Control, 0x000800 Reset checksum (AUTO_INC, AUTO_ERASE, UNLOCK) ID=bxxxxxxxxxxxxxxxxxxxxxxxxxxxx00 Data=0x00 08 00 00 0D 02 00 00	Reset the checksum counters and error status ready for a new download.
→	Put Data, data bytes for Flash data ID=bxxxxxxxxxxxxxxxxxxxxxxxxxxxx01 Data=0xXX XX XX XX XX XX XX XX	First 8 bytes of program (flash) data to be written.
→	… ID=bxxxxxxxxxxxxxxxxxxxxxxxxxxxx01 Data=0xXX XX XX XX XX XX XX XX	Intermediate data bytes. Note if there are gaps within the hex file the management system should fill these with 0xFF so that the program space is contiguous.
→	Put Data, data bytes for Flash data ID=bxxxxxxxxxxxxxxxxxxxxxxxxxxxx01 Data=0xXX XX XX XX XX XX XX XX	Last 8 bytes of program (flash) data to be written.
→	Control, 0x300000 (AUTO_INC, AUTO_ERASE, UNLOCK) - address of EEPROM varies by processor type. ID=bxxxxxxxxxxxxxxxxxxxxxxxxxxxx00 Data=0x30 00 00 00 0D 00 00 00	Set the address to the EEPROM area.
→	Put Data, data bytes for EEPROM data ID=bxxxxxxxxxxxxxxxxxxxxxxxxxxxx01 Data=0xXX XX XX XX XX XX XX XX	First 8 bytes of EEPROM data to be written.
→	… ID=bxxxxxxxxxxxxxxxxxxxxxxxxxxxx01 Data=0xXX XX XX XX XX XX XX XX	Intermediate data bytes.
→	Put Data, data bytes for EEPROM data ID=bxxxxxxxxxxxxxxxxxxxxxxxxxxxx01 Data=0xXX XX XX XX XX XX XX XX	Last 8 bytes of EEPROM data to be written.
→	Control, 0xF00000 (AUTO_INC, AUTO_ERASE, UNLOCK) - address of CONFIG varies by processor type. ID=bxxxxxxxxxxxxxxxxxxxxxxxxxxxx00 Data=0xF0 00 00 00 0D 00 00 00	Set the address to the CONFIG area.
→	Put Data, data bytes for CONFIG data ID=bxxxxxxxxxxxxxxxxxxxxxxxxxxxx01 Data=0xXX XX XX XX XX XX XX XX	First 8 bytes of CONFIG data to be written.

→	… ID=bxxxxxxxxxxxxxxxxxxxxxxxxxxxx01 Data=0xXX XX XX XX XX XX XX XX	Intermediate data bytes.
→	Put Data, data bytes for CONFIG data ID=bxxxxxxxxxxxxxxxxxxxxxxxxxxxx01 Data=0xXX XX XX XX XX XX XX XX	Last 8 bytes of CONFIG data to be written.
→	Control, 0x000000 verify (AUTO_INC, AUTO_ERASE, UNLOCK) checksum ID=bxxxxxxxxxxxxxxxxxxxxxxxxxxxx00 Data=0x00 00 00 00 0D 03 XX XX	The management system calculates the checksum for the data sent or maintains a running total. The checksum is sent within a VERIFY message.
		The module should validate the checksum.
←	Control, OK ID=bxxxxxxxxxxxxxxxxxxxxxxxxxxxx00 Data=0x01	If the checksum is correct and no errors were encountered during loading then return an OK response otherwise return a NOK response.
→	Control, 0x000000 reset(AUTO_INC, AUTO_ERASE, UNLOCK) checksum ID=bxxxxxxxxxxxxxxxxxxxxxxxxxxxx00 Data=0x00 00 00 00 0D 01 00 00	The management system sends a RESET command to exit the bootloader and start the newly downloaded application.
		The module writes 0 into the top EEPROM location and performs a reset.
		The module restarts, enters the bootloader code and checks the top EEPROM location. Since the memory is zero the module enters the application.

A.6 Bootloader Module requirements

Since the bootloader itself is not overwritten during a load from FCU there must be compatibility between the bootloader execution and the module application. The PIC18F2XKXX series and PIC18F2XQXX series of PICs support a clock phase locked loop as a 4 x clock multiplier. The PLL may be enabled either within configuration bits or under software control. Unfortunately when set using the CONFIG bits the application software cannot disable the PLL.

To permit compatibility and interoperability when operating with a 16MHz oscillator the PLL should be DISABLED in configuration bits so that **the bootloader executes at a clock speed of 16MHz**. The application can enable the PLL if required so that the application executes at 64MHz or left with the PLL disabled to operate at 16MHz.

The user program must have the following vectors.

- User code reset vector 0x0800
- User code HPINT vector 0x0808
- User code LPINT vector 0x0818

This would normally be achieved using XC8 option --codeoffset=0x800

Appendix B - Implementation Notes

- This specification stipulates that all but the lower 2 bits of the EID should be ignored. FCU obeys this requirement however it has been found that LayoutMesh requires the EID to be set to a specific value to be recognised. It is therefore recommended that the ID for all frames from the module should be set to binary
  000 000000 0010 0000 0100 0000 00xx

- When a module supports 2 LEDs both of the LEDs should be illuminated whilst running the bootloader.

- FCU has support for an undocumented addition to the Boot test response in which the processor ID may be returned in the second data byte.

- Program memory (Flash) addresses between 0x0000~0x07FF are reserved for the bootloader.

- The top byte of EEPROM is reserved for the boot flag.

- Application code starts at 0x800. Code assembled for 0x800 offset so that reset vector is at 0x800, High priority interrupt vector at 0x0008 and Low priority interrupt vector at 0x0018.

- Having the parameter block at the fixed address 0x820~0x838 means that it is not compatible with some processor types e.g. PIC24.

- The application reset vector (0x800) must be stored in the parameter block at the loadAddress at 0x082A.

- Care must be taken in selecting EEPROM addresses to be used by an application as FCU and MMC operate differently when programming EEPROM. It is recommended that the application is configured to specify all required values in the EEPROM section of the hex file instead of relying upon them being defaulted to 0xFF.

- The bootloading process uses CAN extended frames outside of the CBUS spec.

- The protocol is explicitly tied to CAN.

- The protocol is limited to 24 bit addresses.

- Getting the correct CONFIG bits (fuses) can be difficult if the application and require different CONFIG settings.

- The module's bootloader must correctly interpret the upper address byte to recognise Flash, EEPROM and CONFIG regions.

- The bootloader and application will have the PLL disabled in CONFIG but may be subsequently enabled or disabled by the application.

- The clock frequency expected by the application must be compatible with the bootloader/hardware.

- The application's CONFIG settings must be compatible with those of the bootloader.

- The bootloader firmware is based on the Microchip protocol. There are a few implementations are available such as:
    - C18 assembler by Mike Bolton - Bootloader_type=1,
    - XC8 C by Konrad Orlowski - Bootloader_type=2,
    - XC8 C by Ian Hogg - Bootloader_type=3.

  Some of the existing known bootloaders do not fully meet the Bootloader specification:

	Bootloader.asm by Mike Bolton Type=1	CbusLibXC8 by Konrad Orlowski Type=2	CBUS_PIC_Bootloader by Ian Hogg Type=3
Writing of Flash	✓	✓	✓
Writing of EEPROM	✓	✓	✓
Writing of CONFIG	✓	✓	✓
Reading of Flash	✘	✘	✓
Reading of EEPROM	✘	✘	✓
Reading of CONFIG	✘	✘	✓
MODE_AUTO_INC	✓	✓	✓
MODE_WRT_UNLCK	✓	✓	Always
MODE_ERASE_ONLY	✓	✓	✓
MODE_AUTO_ERASE	✓	✓	✓
MODE_ACK	✘	✘	✓
Checksum verify	✓	✓	✓
Bootloader version	✘	✘	✓