Flash Kernel Programming on a HC12 micro controller

Version 0.01
09/29/03

Application Note AN-IMC-1-002

Author(s) Sven Deckardt
Restrictions Draft Document
Abstract The purpose of this application note is to explain how to implement a CCP flash kernel for the HC12 and to configure CANape Graph for flash programming.

Table of Contents

1.0 Overview

This documentation explains how to write a flash kernel for a HC12D60 micro controller and how to configure the settings in CANape Graph to enable flash programming. The purpose of a flash kernel is to download a hex file from CANape Graph into the flash memory of the electronic control unit (ECU).

The flash kernel is loaded automatically by CANape into the micro controller's RAM via CCP whenever the flash memory must be reprogrammed. The flash kernel contains beside a minimal CCP and CAN driver and all the needed flash routines for performing the flash programming.

This document should be used as a frame work, which enables you to write your own flash kernel for your ECU.

Note: The source code of this application note is available only on customer request at Vector.

Contact Information: www.vector-cantech.com or 1-248-449-9290

General usage of a flash kernel

The purpose of an ECU is to perform calculations with sensor data or other signals in the RAM. The main application is stored in the ROM or flash memory of the ECU. In running operation the user is able to change the behavior of the ECU via changing some parameters with help of a measurement and calibration tool, like CANape Graph (see Figure 1). The general disadvantage is that only RAM data can be changed. Data stored in the flash memory can only be programmed via special flash routines.

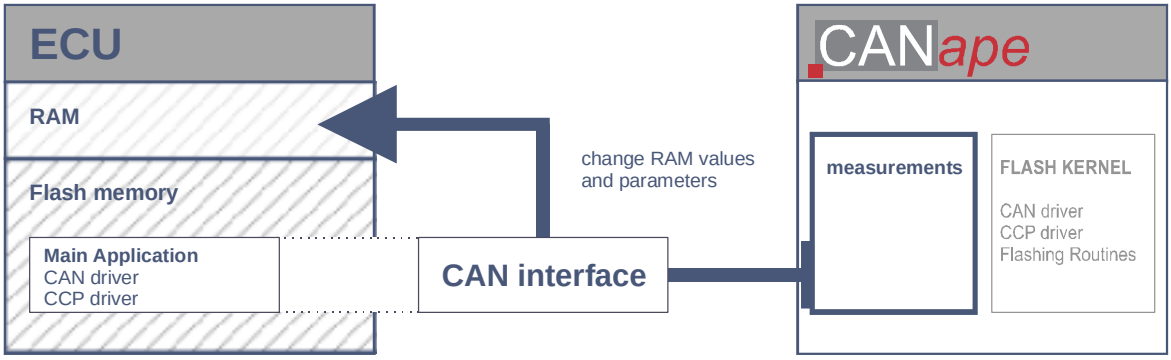


Figure 1: normal operation of ECU, measurements are done with CANape

To solve this problem it is possible to integrate flash routines into the code of the main ECU application. The disadvantage of this solution is, that flash memory is wasted unnecessary because these flash routines are not used very often and for security reasons they should not be available in the released product. Another solution is the usage of a flash kernel. The flash kernel is loaded by CANape Graph into the micro controller's RAM via CCP whenever the flash memory must be reprogrammed. The flash kernel contains the needed flash routines, its own CAN and CCP driver to communicate via the CAN interface with CANape Graph.

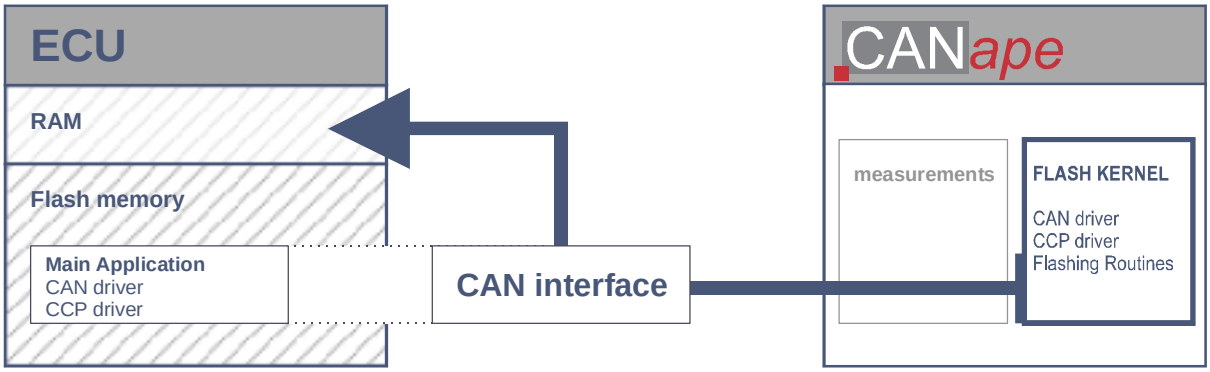


figure 2: user loads down flash kernel with integrated CCP and CAN driver into the RAM of the ECU with help of CANape

After the flash kernel has been downloaded, it is executed in the RAM of the ECU. The mini CCP driver communicates with CANape Graph and waits for the data, which has to be programmed by the flash routine.

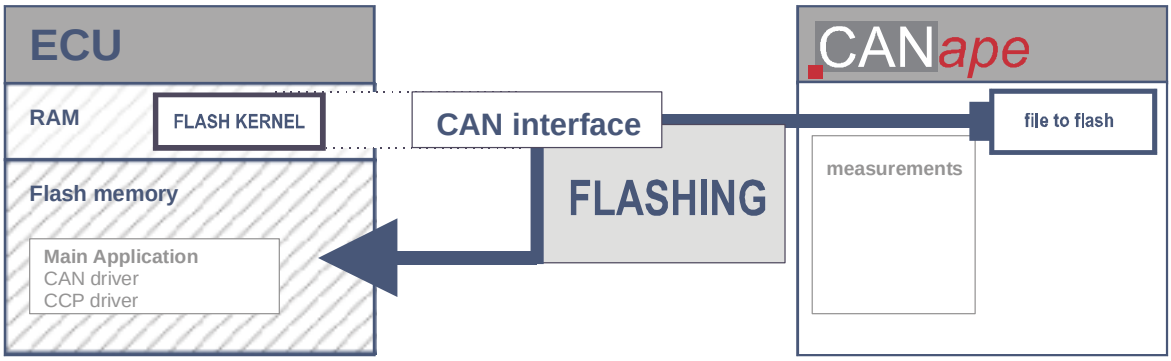


figure 3: CANape flashes a hex file (e.g. main application) into the flash of the ECU by using the CAN interface of the kernel, old main application is stored in the flash memory will be erased and then re-programmed

Note:

If there are any problems during the flash programming, please check the following points:

- Is the RAM area empty (is it used by the flash kernel)?
- Does an interrupt function from the main application overwrite the flash kernel, e.g. “timer interrupt”

2.0 Flash Kernel

2.1 Structure

A flash kernel contains three parts of code:

- a CAN driver, for the CAN communication
- a CCP driver for the communication between the ECU and CANape Graph
- the flash programming routines for programming the flash memory of the ECU

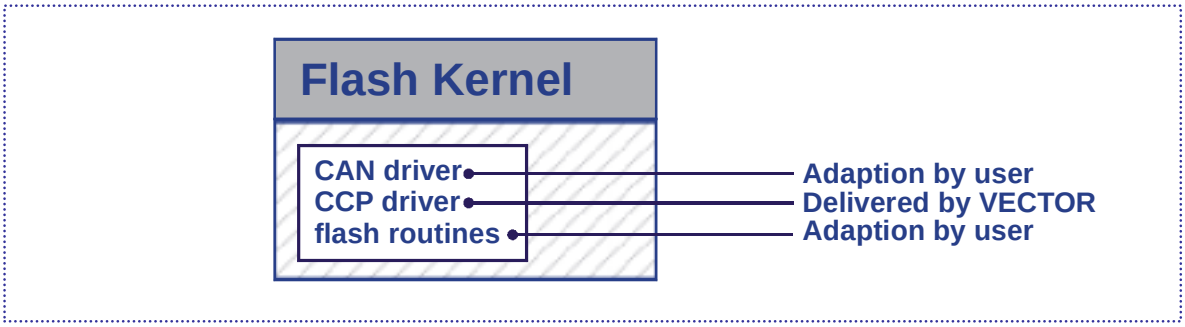


figure 4: contents of a flash kernel

Because different micro controllers are used in ECU's, the user has to adapt the CAN driver and the flash programming routines to the ECU. The CCP driver which is delivered by VECTOR, do not need any changes.

In chapter 4 (CCP_TEST example with a Flash Kernel) an example how to adapt these files will be discussed.

The flash kernel file (e.g. D60ccp.fkl) consists of two parts: an ASCII-header part and an Intel®-Hex file part. The kernel header file must be adapted by the user. The flash kernel has to be compiled and must be converted into the Intel®-Hex format.

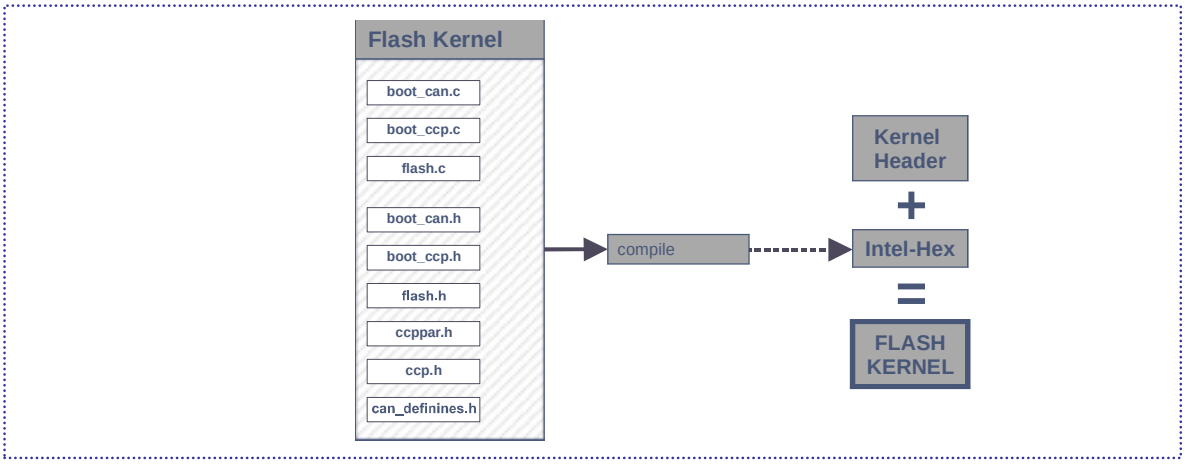


figure 5: details of a flash kernel

2.1.1 The flash kernel header file

The flash kernel header file contains some information about the kernel file name, the addresses of the RAM location and the start address of the main function in the flash kernel.

Note:
The main application of the flash kernel must start with the function: **ccpBootLoaderStartup()**, ensure FLASH_KERNEL_RAM_START has got the right function address. Sometimes the flash kernel location is at the same address like the vector interrupt table. To check this, the developer must add the size of the kernel to the FLASH_KERNEL_RAM_START address. For Example here FLASH_KERNEL_RAM_START + FLASH_KERNEL_SIZE = 1533. This means the RAM location from 0x1000 – 0x1533 must be empty.

```
[FLASH_KERNEL_CONFIG]

FLASH_KERNEL_NAME="xxxxx.fkl"
FLASH_KERNEL_COMMENT="Flash Kernel for xxxxxx"
FLASH_KERNEL_FILE_ADDR=0x1000
FLASH_KERNEL_SIZE=0x0533
FLASH_KERNEL_RAM_ADDR=0x1000
FLASH_KERNEL_RAM_START=0x1000


[FLASH KERNEL DATA]
```

figure 6: example header file of a flash kernel

2.1.2 The Intel®-Hex file

Generally the user gets a Motorola-S file when using a HC12 micro controller. However, the flash kernel needs an Intel®-Hex file. The conversion can be created with the program HEXTOOL.EXE, which converts a Motorola-S file into an Intel®-Hex file. The HEXTOOL program is delivered by VECTOR.

To generate an Intel®-Hex file you need for example following commands:

- HEXTOOL.EXE D60xcp.s D60ccp.hex
- copy kernelhead.txt + D60ccp.hex D:\CAN\CANape32\Exec\D60ccp.fkl

```
:20040000ED85EC463BED84EC44160BBE1B82EC821605403D1606783D3B3BE6892629160721
:2004200071CC07E61605CEC6056BAA6389CCEA60C3FFFF6C80ED82026D82EC8026F2E689E8
:2004400026E91605C1201FE6F30002873BEC881606F96CB12712ED82026D82ED86026D862E
…
…
…
…
:2007A0004301CC000A07183DFD07F90C4301CC00160721FD07F90D4301CC000A07163D3BB3
:2007C0003B87CD0477136C80ED801A5F6E800476F71B843D3B3B87596C80ED801A5F6E80D9
:0607E0000476F71B843DC6
:0207F90000F40A
:00000001FF
```

figure 7: example of an Intel®-Hex file

2.2 Flash Routines

The user has to write four flash routines, which are called by the CCP driver:

- **int flashEraseBlock (unsigned char *ptr)**
  erases the flash memory sections
- **int flashByteWrite (unsigned char *dest, unsigned char data)**
  write a single byte at a given destination (flash memory address)
- **void flashInit()**
  prepare the ECU for programming
- **void flashExit()**
  calls user specific follow-up routines

The code of these functions is different for each micro controller. This application note is explained how to write these functions for a Motorola HC12 micro controller. For further information see the source code **flash.c** of our implementation in at section 4.

2.3 General Preparation

To use the flash programming ability via flash kernel, it is necessary to activate "*flash programming kernel download*" define in the **ccppar.h** file of your main application of the ECU.

change:

```
…
…
/* Activate the flash programming kernel download */
// #define CCP_BOOTLOADER_DOWNLOAD
…
…
```

to:

```
…
…
/* Activate the flash programming kernel download */
#define CCP_BOOTLOADER_DOWNLOAD
…
…
```

Note:
This option enables the flash kernel downloaded via CANape Graph, but the resident CCP driver inside the ECU does not contain any flash routines!

3.0 D60 Flash Kernel Sample

3.1 CAN / CCP Driver

The CAN driver provided from VECTOR in this example contains only three basic functions. The CCP driver for the flash kernel delivered by VECTOR is used without any changes.

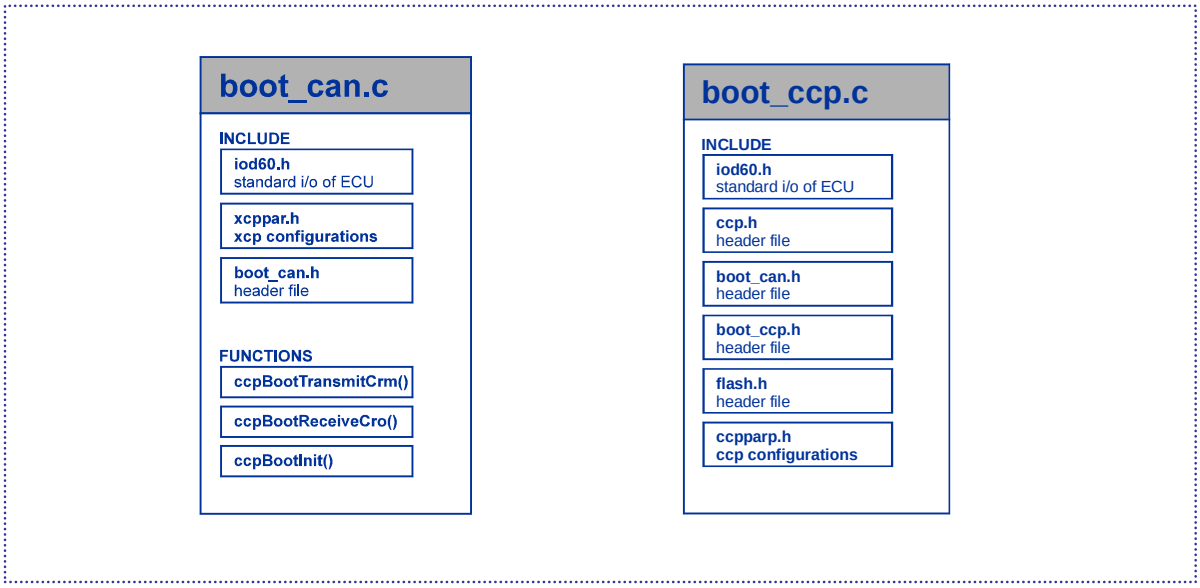


figure 8: boot_can.c and boot_ccp.c file

CAN driver

- **ccpBootInit()**
  initializes the CAN driver and configure the bus timing
- **ccpBootTransmitCrm()**
  function to send a single CAN message of max. eight bytes
  and the message is stored in transmit buffer 0 and then transmitted
- **ccpBootReceiveCro()**
  checks if a new message is available in the receive buffer and then
  the value 1will be return, if a message is correctly received

3.2 Flash Routines

As described before, the user has to write four main functions in the flash.c file.

called by the CCP driver

- **int flashEraseBlock (unsigned char *ptr)**

  This function contains a routine which analyses the address **ptr* and selects the correct flash array and byte length of the D60 HC12:

```
// check address ------------------------------------------------------------

address = (unsigned short) ptr;             // assign pointer addr to variable
if (address<0x8000) {
  len = 0x7000;
  OFFSET = 0xF8;                            // select Flash Array 1
}
else {
  len = 0x4000;
  OFFSET = 0xF4;                             // select Flash Array 2
}
```

figure 9: flash array selection, code example of D60 HC12 flash kernel

  The selected flash array will be erased completely by applying the erase voltage. If the flash array was successfully erased, the value 1 will be returned. See **flash.c** for further details.

  Note:
  The D60 HC12 contains two flash arrays $1000…$7FFF and $8000…&FFFF. The used demo board in our example contains an internal monitor program which reserves $C000… $FFFF, so the second flash array cannot completely be erased (byte length = 4000h).

- **int flashByteWrite (unsigned char *dest, unsigned char data)**

  This function contains a routine which analyses the address **dest* and selects the correct flash array and programs the single byte stored in *data* at that address.

- **void flashInit()**

  Disable all interrupts by setting the I-bit

- **voidflashExit()**

  is empty

called by the flash routines

- **flash_empty()**

  this function checks, if the flash array at address *startflash* with the length *length* contains only values of 0xFFFF (=empty) and returns in this case the value one.

- **fpApplyEraseVoltage()**

  applies erase voltage with a special timing

- **fpApplyProgrammingVoltag()**

  applies programming voltage with a special timing

- **fpDelayms ()**

  delay in ms

- **fpDelayus()**

  delay in µs

Important Notes

To avoid faults in the programming procedure ensure, that:

- the flash kernel size is small as possible, don't waste RAM! The primary CAN/CCP driver needs also RAM space until the flash kernel runs.
- do not overwrite your stack pointer or its contents
- the flash arrays must be correctly selected
- the flash kernel header must contain the correct RAM address and file size

3.3 XCPPAR.H

The **ccppar.h** file contains important configurations and options for the XCP driver inside the flash kernel. To make a proper flash kernel it is very important to set following options:

```
#define CCP_PROGRAM
#define CCP_BOOTLOADER_DOWNLOAD
#define CCP_BOOTLOADER_SIMPLE
```

CCP_PROGRAM
tells, that flash routines are included in this application (flash kernel)

CCP_BOOTLOADER_DOWNLOAD
enables flash kernels to be downloaded

CCP_BOOTLOADER_SIMPLE
when defined, only a few main functions are implemented to hold the file size small

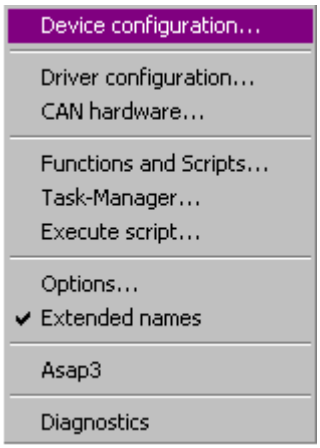
```
// ------------------------------------------------------------------------------
// DEFINITION
// ------------------------------------------------------------------------------
#ifndef __CCPPAR_H__
#define __CCPPAR_H__
#define RAM
#define ROM
extern void disable_interrupt(void);
extern void enable_interrupt(void);
#define CCP_DISABLE_INTERRUPT disable_interrupt();
#define CCP_ENABLE_INTERRUPT  enable_interrupt();
#define BYTE       unsigned char
#define WORD       unsigned short
#define DWORD      unsigned long
#define BYTEPTR    unsigned char*
#define MTABYTEPTR unsigned char*
#define CCP_STATION_ID "BootKernelHC12"
#define CCP_STATION_ADDR 0x000
#define CCP_PROGRAM
#define CCP_BOOTLOADER_DOWNLOAD
#define CCP_BOOTLOADER_SIMPLE
#endif
// ------------------------------------------------------------------------------
```

figure 10: ccppar.h, code example of D60 HC12 flash kernel

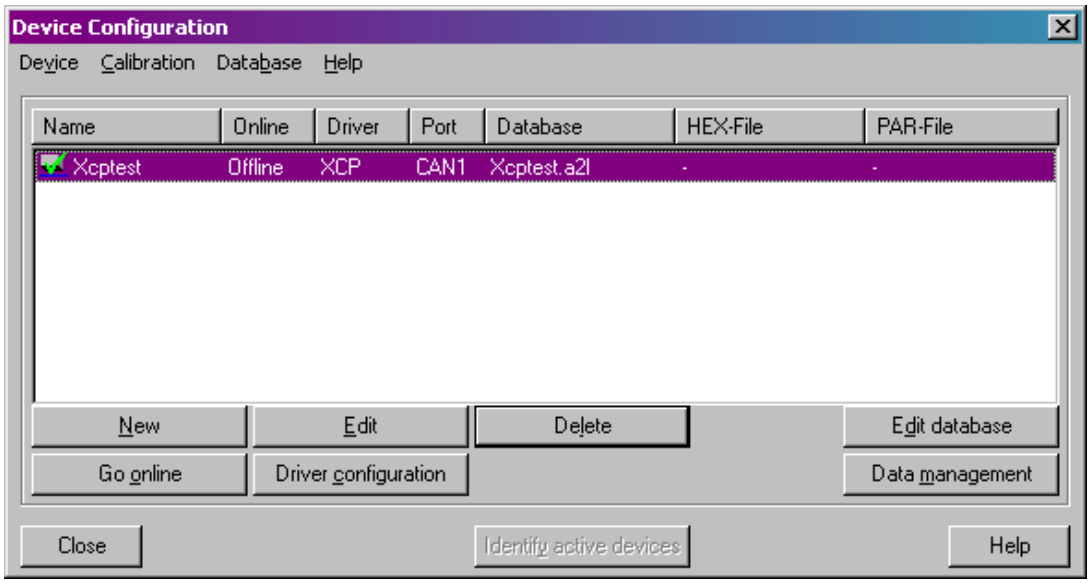
3.4 Configuration of CANape Graph

To enable the flash programming in CANape Graph via flash kernel, following points must be realized:

- copy the needed flash kernel file with extension *.fkl into the CANape\Exec - or in the working folder
- open the CANape project, click “Options / Device configuration” in the menu



- click on the button “Driver configuration”



- change to the tab “FLASH” and insert the start address and the byte length of each flash array of the ECU

Note
In our example the second flash array ($8000…$FFFF) has a byte length of 4000h only. An internal monitor program reserves the memory location from ($C000-$FFFF).

- enable the following check-boxes:

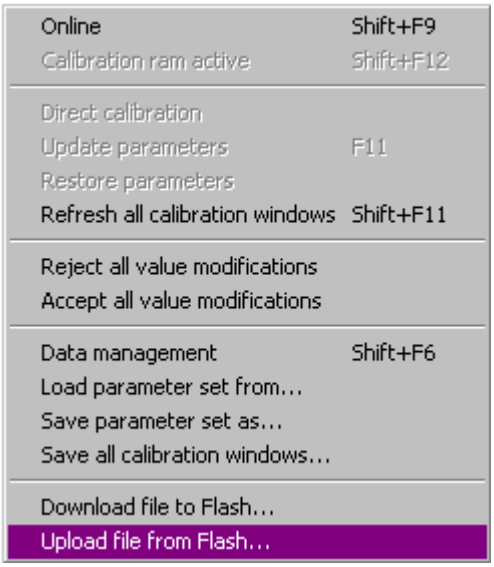
  - 0xFF Optimization
  - Reconnect

  and select the previously copied flash kernel (**D60ccp.fkl**) in the combo box.

- click “Ok”, followed by “Close”

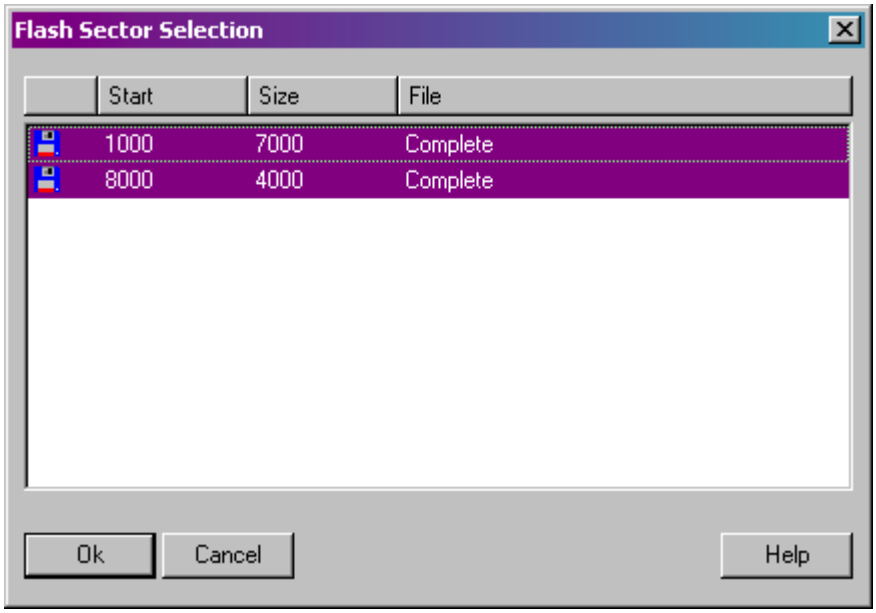
To check, if the flash kernel runs properly, make an upload from your ECU ROM content into CANape:

- choose “Calibration / Upload file from Flash…”



- select a file name to save

- choose now the menu entry “Download file to flash” and select the previously saved upload file, and click the “Ok” button



If everything was configured correctly, the hex file will be flashed. In the status bar a progress bar displays the flash programming process.

4.0 Contacts

Vector Informatik GmbH
Ingersheimer Straße 24
70499 Stuttgart
Germany
Tel.: +49 711-80670-0
Fax: +49 711-80670-111
Email: info@vector-informatik.de

Vector CANtech, Inc.
39500 Orchard Hill Pl., Ste 550
Novi, MI 48375
Tel: (248) 449-9290
Fax: (248) 449-9704
Email: info@vector-cantech.com

VecScan AB
Fabriksgatan 7
412 50 Göteborg
Sweden
Tel: +46 (0)31 83 40 80
Fax: +46 (0)31 83 40 99
Email: info@vecscan.com

Vector France SAS
168 Boulevard Camélinat
92240 Malakoff
France
Tel: +33 (0)1 42 31 40 00
Fax: +33 (0)1 42 31 40 09
Email: information@vector-france.fr

Vector Japan Co. Ltd.
Nishikawa Bld. 2F, 3-3-9 Nihonbashi
Chuo-ku Tokyo 103-0027
Japan
Tel: +81(0)3-3516-7850
Fax: +81-(0)3-3516-7855
Email: info@vector-japan.co.jp