C2Prog User Manual

Version 1.8

June 18, 2019

Disclaimer

THE SOFTWARE IS PROVIDED ”AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Contents

1 Introduction

C2Prog is a flash programming tool for TI C2000™ MCUs. In addition to reflashing over JTAG, C2Prog also supports reflashing over RS-232, RS-485, TCP/IP and Controller Area Network (CAN). The programmer is, therefore, well suited for deployment in the field where the JTAG port is typically not accessible.

Some salient features of the programmer are:

- Ease of use and reliable operation.
- Support for multiple communication interfaces and protocols.
- Smart flash erase, or manual sector section selection.
- Automatic 32-bit CRC generation for flash integrity verification at MCU bootup.
- “Extended Hex” file format for firmware distribution (encapsulates all settings for programming, including the secondary bootloader).
- Firmware password protection.
- Firmware encryption (licensed separately).
- Fast serial communication protocol that works reliably with USB-to-RS-232 converters.
- Communication protocol compatible with multidrop networks (RS-485).
- Support for Texas Instruments XDS JTAG emulators.
- CAN communications based on ISO-14229/15765.
- GNU debug (Gdb) server stub.
- Command-line and DLL interface for batch programming and integration of C2Prog functionality into other applications (requires “professional” or “integration” license for C2Prog –- see our `CodeShop`).
- Flexible and modular design allowing for customer specific solutions.
- Small footprint application.

2 Quick Start

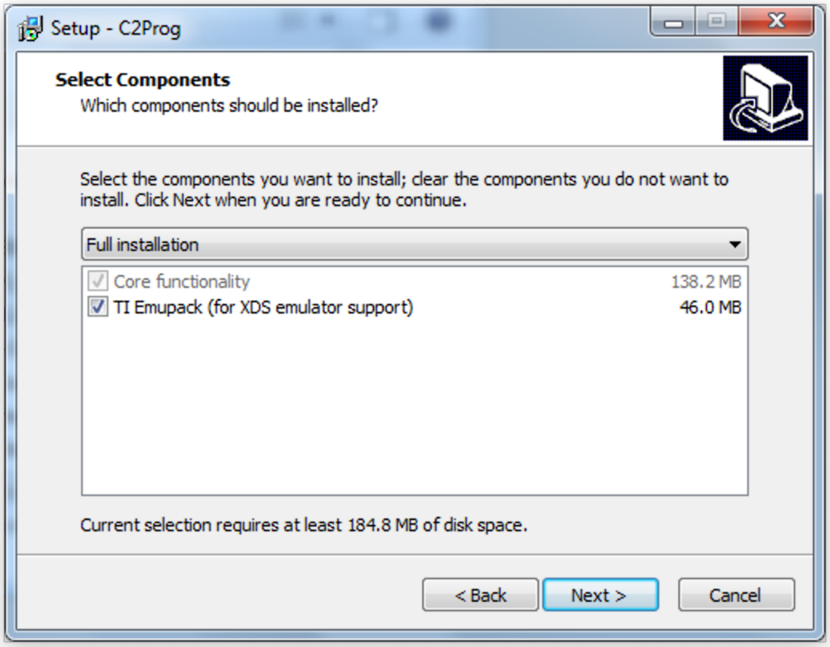
2.1 Installation

The most recent version of the programmer can be downloaded from CodeSkin's website:

`http://www.codeskin.com/C2Prog.html`

The application is installed using the Windows installer **setup.exe**. It can also be uninstalled from the control panel similar to other Windows software.

During installation, you have the option to install the TI emulator drivers. This selection is required if you intend on using C2Prog for programming over JTAG (XDS emulators).



Setup - C2Prog

Select Components

Which components should be installed?

Select the components you want to install; clear the components you do not want to install. Click Next when you are ready to continue.

Full installation

☑ Core functionality 138.2 MB

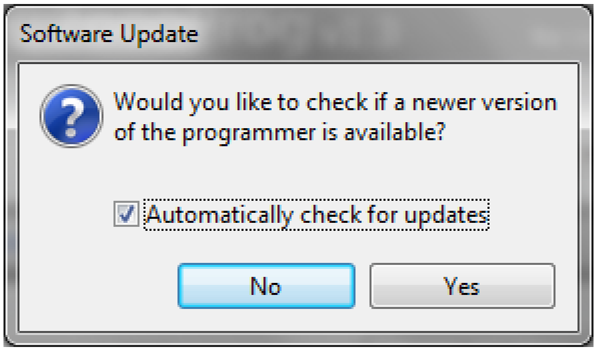
☑ TI Emupack (for XDS emulator support) 46.0 MB

Current selection requires at least 184.8 MB of disk space.

☞ The installation of the TI emulator drivers can take several minutes, especially if a virus protection program is running in the background during the installation.

☞ If C2Prog is used in conjunction with Code Composer Studio (CCS), it is possible to configure C2Prog to use the emulator drivers already included with CCS. See Section 3.3.2 on page 11 for more information on this advanced feature.

When the programmer is launched for the first time, an option is presented to check if a newer version is available. It is highly recommended that the most recent version be installed.



Software Update

Would you like to check if a newer version of the programmer is available?

☑ Automatically check for updates

If the **Automatically check for updates** option is enabled, the programmer will periodically query the update-server to check if a newer version of the application is available.

☞ **Privacy:** When communicating with the CodeSkin update-server, C2Prog will transmit the C2Prog version number and installation ID. The web server will further have access to the public IP address of the network from which the request is made. CodesSkin may use this data for compiling anonymous statistics. However, no proprietary or personally identifiable data is ever transmitted or collected.

2.2 Supported Binary Files

C2Prog supports all binary files generated by the TI codegen tools, including COFF/ELF (*.out) files and Intel hex (.hex) files.

If you want to use hex files with C2Prog, then you must generate them as follows:

C2400 Tools:

```
dsphex -romwidth 16 -memwidth 16 -i -o .\Debug\code.hex .\Debug\code.out
```

C2000 Tools:

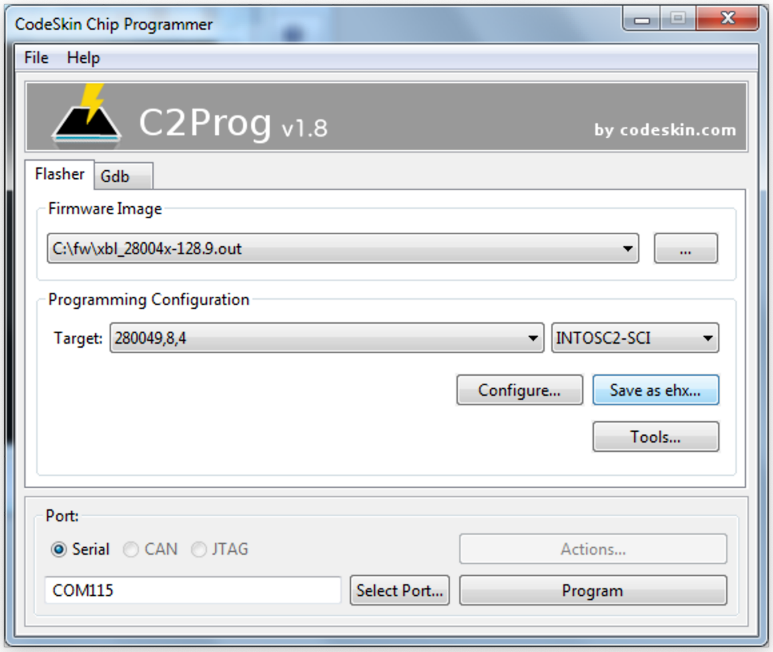
```
hex2000 -romwidth 16 -memwidth 16 -i -o .\Debug\test.hex .\Debug\test.out
```

ARM Tools:

```
armhex -romwidth 8 -memwidth 8 -i -o .\Debug\test.hex .\Debug\test.out
```

2.3 Programming (over RS-232)

This section demonstrates the use of C2Prog in conjunction with TI’s SCI bootloader. Please refer to the TI Boot ROM Reference Guides for information on how to configure your target for SCI programming. In order to operate with the C2Prog default settings, the serial link must be capable of full duplex communication at 115200 baud. Slower links are supported, but will require custom settings.



CodeSkin Chip Programmer

Firmware Image: C:\fw\xbl_28004x-128.9.out

Programming Configuration

Target: 280049,8,4 | INTOSC2-SCI

Port: ● Serial ○ CAN ○ JTAG

COM115

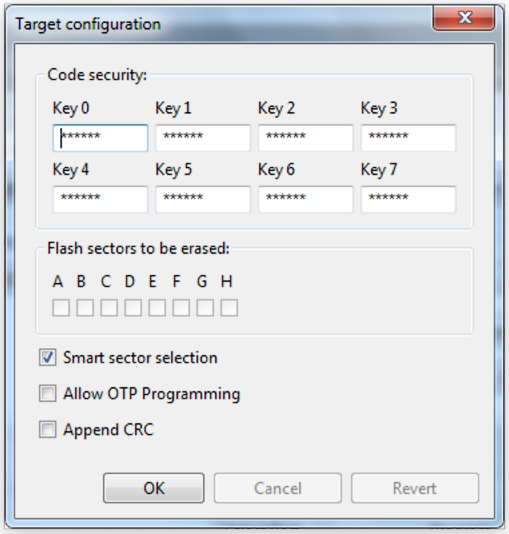
Activate the **Flasher** tab and select the **Firmware Image** (binary file) by means of the **...** popup menu.

Standard binary files do not define any target specific programming information such as the type of MCU to be flashed, the oscillator frequency, the communication protocol to be used, etc. You must therefore configure this information in the **Programming Configuration** section by choosing the appropriate MCU part-name and option (e.g. clock frequency and communication interface - see Section 3.1 on page 9). Contact CodeSkin at info@codeskin.com if no match is found for your hardware.

Next click on **Configure...** to open the configuration dialog.

For programming a locked C2000™ MCU, valid CSM keys (passwords) must be provided as 4 digit hex numbers (with ’0x’ or ’$’ prefix). In the case of 32-bit keys, each key is split into two 16-bit keys in big endian (BE) order. Note that contrary to the other fields, keys are not remembered as defaults. The keys are also automatically reset to 0xFFFF whenever the target selection is changed.

Now you must select which flash sectors should be erased prior to programming. This is either done manually by checking the individual boxes, or by choosing **Smart Sector Selection**. The smart sector feature automatically detects which sectors require erasing by parsing the contents of the binary file.



Target configuration

Code security:

Key 0	Key 1	Key 2	Key 3
******	******	******	******
Key 4	Key 5	Key 6	Key 7
******	******	******	******

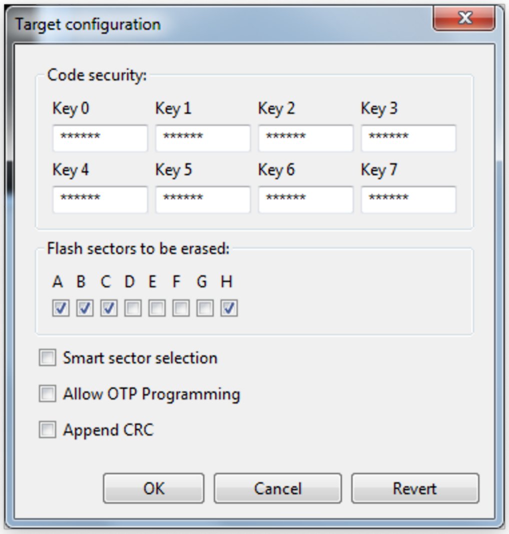
Flash sectors to be erased:

A	B	C	D	E	F	G	H
□	□	□	□	□	□	□	□

☑ Smart sector selection

□ Allow OTP Programming

□ Append CRC



Target configuration

Code security:

Key 0	Key 1	Key 2	Key 3
******	******	******	******
Key 4	Key 5	Key 6	Key 7
******	******	******	******

Flash sectors to be erased:

A	B	C	D	E	F	G	H
☑	☑	☑	□	□	□	□	☑

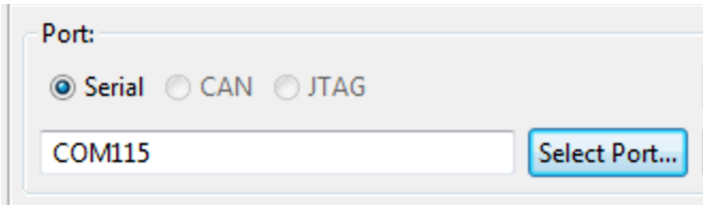
□ Smart sector selection

□ Allow OTP Programming

□ Append CRC

As a further option, **Append Checksum** can be selected, which instructs the programmer to append a 32-bit CRC checksum to the hex data. This checksum can be used by the MCU to verify the integrity of the flash data, as described later in this document.

Once all the programming configurations are made, configure the COM port, either by typing its name directly into the text-field, or clicking on the **Select Port...** button. Valid entries for the serial port on the windows platform are COM1, COM2, etc.



Port:

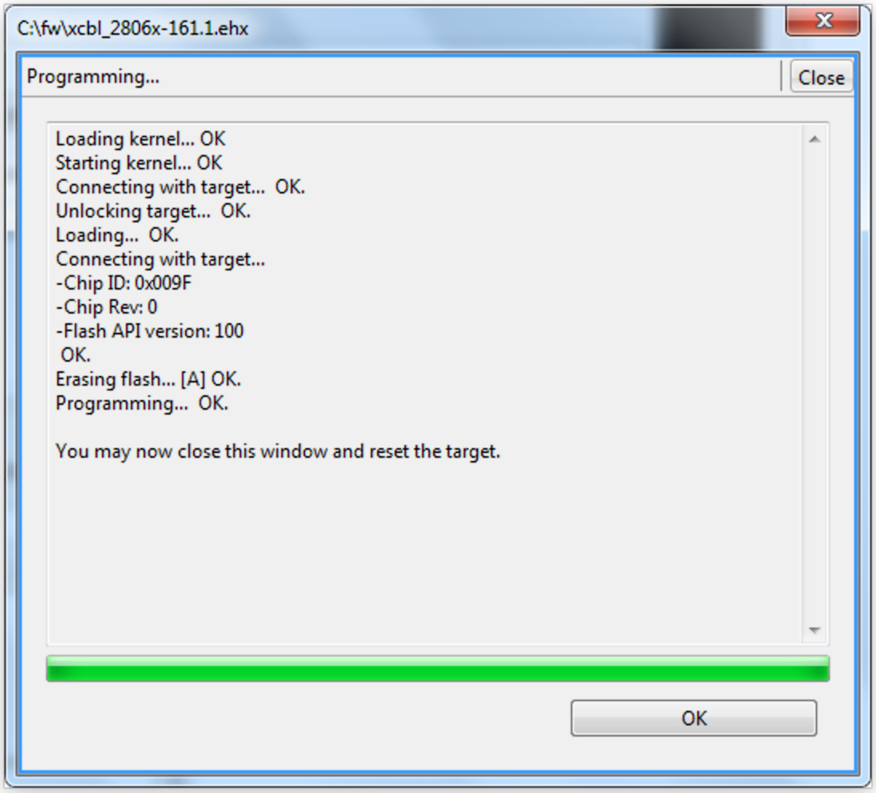
● Serial ○ CAN ○ JTAG

COM115

The **Scan Ports** button in the port selection dialog automatically scans for all available serial ports. Please note that on some computers this feature can take a very long time to execute, especially if Bluetooth COM ports are present.

Now the reflashing process can be started by clicking the **Program** button. This will open a new window which displays status information while the programming progresses, as shown below.

⚠ Do not interrupt the programming as this can cause the MCU to be permanently locked. Also, do not power-cycle or reset the target during programming.



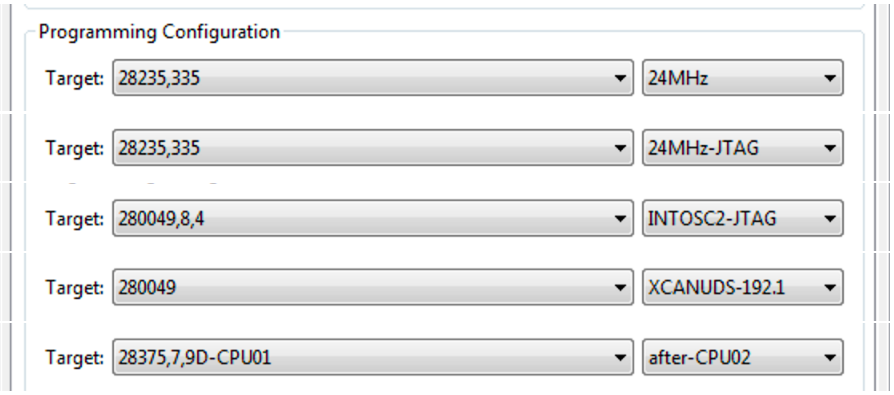
```
Loading kernel... OK
Starting kernel... OK
Connecting with target...  OK.
Unlocking target...  OK.
Loading...  OK.
Connecting with target...
-Chip ID: 0x009F
-Chip Rev: 0
-Flash API version: 100
 OK.
Erasing flash... [A] OK.
Programming...  OK.

You may now close this window and reset the target.
```

Finally, you may save the programming configuration combined with the firmware image to an Extended Hex file by clicking on the **Save as ehx...** button. When this file is subsequently selected in C2Prog, all programming settings are automatically configured. This format is thus well suited for distributing firmware images.

3 Detailed Description

3.1 Communication Protocols

C2Prog supports a large number of communication protocols and interfaces. Originally designed with a focus on serial communication (RS-232/485), C2Prog now also supports TCP/IP, Controller Area Network (CAN) and JTAG. The communication protocol is selected by means of drop-down fields in the **Programming Configuration**. Shown below are some examples:



Programming Configuration

Target: 28235,335 | 24MHz

Target: 28235,335 | 24MHz-JTAG

Target: 280049,8,4 | INTOSC2-JTAG

Target: 280049 | XCANUDS-192.1

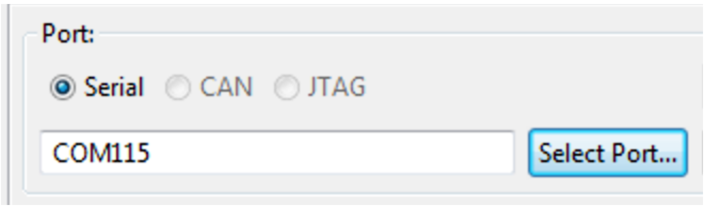
Target: 28375,7,9D-CPU01 | after-CPU02

A few comments about target options:

- Frequency values, e.g. **24MHz**, specify the external clock/crystal frequency.
- Options without explicit frequency value use the internal oscillator of the MCU.
- **INTOSC** also refers to the internal oscillator.
- The qualifier after the frequency parameter specifies the communication interface, e.g. **SCI**, **CAN**. The default interface is SCI, i.e. **24MHz** stands for SCI communication with an external clock of 24 MHz.
- Options can also refer to customer specific configurations, such as in the **XCANUDS-192.1** example.
- When programming a dual-core processor over SCI, it is possible to speed-up the process by first programming CPU2, and subsequently programming CPU1 with the **after-CPU02** or **after-C28** option.

3.2 Programming over Serial Link

The serial interface can be used with TI's SCI bootloader as well as customer specific bootloaders developed by CodeSkin. Please refer to the TI Boot ROM Reference Guides for information on how to configure your target for SCI programming. In order to operate with the C2Prog default settings, the serial link must be capable of communicating at 115200 baud and support full duplex-transmission. Slower links and half-duplex operation are supported, but will require custom settings.


Port:

◉ Serial ○ CAN ○ JTAG

COM115

Valid entries for the serial port are **COM1**, **COM2**, etc.

C2Prog works most reliably with converters that utilize the FTDI chipset. A good choice, for example, is the Parallax USB to Serial Adapter. The serial port of TI's controlCARDs, LaunchPads and "Experimenter's Kit USB Docking Station" is also proven to work well with C2Prog.

The serial protocol also allows communication over TCP/IP (for example, in conjunction with targets that use Lantronix Serial-to-Ethernet or Serial-to-WiFi converters).

The port configuration string for TCP/IP is "socket:<ip>:<port>", where "ip" stands for the server IP address, and "port" is the port of the server, e.g. **socket:192.168.0.100:1001**.

3.3 Programming over JTAG

C2Prog currently supports XDS emulators.

⚠ It is extremely important that the correct external clock/crystal frequency is selected, as C2Prog has no means of verifying the frequency when using JTAG. Selecting the wrong frequency may damage the MCU.

The port configuration must be formatted as follows:

- xds100v<v>:<sn>:<pid>
- xds110:<sn>
- xds110-2w:<sn>
- xds2xx

where <v> must be equal to ”1”, ”2”, etc, <sn> is the serial number of the emulator (if several are connected), and <pid> is the USB PID of the emulator hardware in hex notation. The "2w" postfix specifies use of the 2-pin cJTAG mode. Both the <sn> and <pid> parameters are optional. The default value for <pid> is 0xa6d0. Some examples for valid configuration strings include:

- “xds100v1”, “ xds100v2”, “ xds100v3”, “ xds110”, “ xds110-2w”,
- “xds100v2:TIUDCI83”
- “xds100v2:TIUDCI83:6010”
- “xds100v2::6010”

The serial number of a particular unit can be determined by means of the command-line utility stored in the `{C2Prog}\tiemu\ccs_base\common\uscif\ftdi\utility` directory.

3.3.1 XDS110 Considerations

TI’s emupack will automatically update the firmware of XDS110 emulators. This process is not always 100% reliable. Should your XDS110 emulator become unresponsive, open a command-line prompt, navigate to the `{C2Prog}\tiemu\ccs_base\common\uscif\xds110` folder and execute the following sequence of commands to restore the firmware that is compatible with C2Prog.

```
xdsdfu -m
xdsdfu -f firmware.bin
```

3.3.2 Advanced Configuration

It may be desirable to have C2Prog use the same set of TI emulator drivers as the version of CCS that is installed on the same computer for code development. This can be achieved by means of a manual configuration.

At startup, C2Prog checks for a file named `config.xml` in a folder `c2pemuconf` located in your user directory. i.e.: `C:\Users\joe\c2pemuconf\config.xml`

The contents of this file should be formatted as shown below. In this example, C2Prog is configured to use the drivers of CCSv8 installed at `C:\ti`.

```
<?xml version="1.0" encoding="UTF-8"?>
<root>
  <ti>
    <defaultparams>
      <rootdir>C:\ti\ccsv8\ccs_base</rootdir>
    </defaultparams>
  </ti>
</root>
```

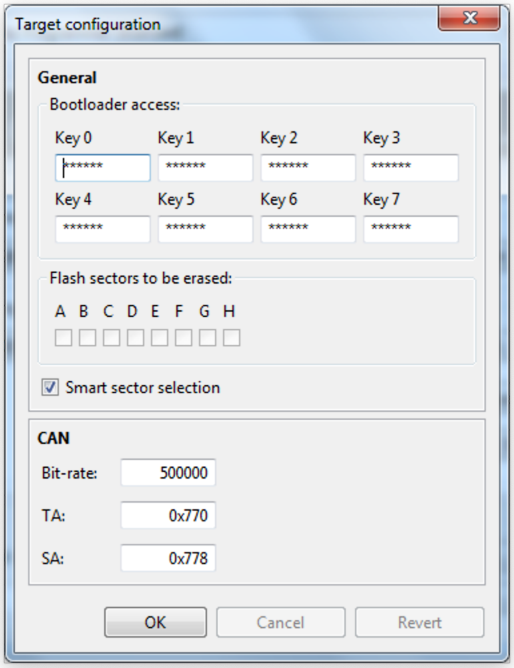
Referring to an existing installation of drivers also can save disk space.

3.4 Programming over CAN

C2Prog allows programming over CAN using the “Unified Diagnostics Protocol” as defined in ISO-14229/15765. CAN hardware from Vector, Kvaser, Lawicel, NI, Peak and EMS-Wuensche is supported. The syntax for the port configuration is as follows:

- Lawicel: “canlw:0” – only one port supported
- Kvaser: “cankv:<n>” – where <n> is the port number (0 or 1)
- Vector: “canvec:<n>” – where <n> is the port number (0 or 1)
- NI-CAN: “canni:<n>” – where <n> is the port number as configured in NI-MAX
- NI-XNET: “canxnet:<n>” – where <n> is the port assigned to the adapter
- Peak CAN: “canpk:<n>” – where <n> is the device number as configured in PCAN-View, e.g. 255; other channels on the same device are identified as “255B”, “255C”, etc
- EMS-Wuensche: “canew:<n>” – where <n> corresponds to the channel number ([CHANn]).

A special convention is used to allow the configuration of custom bit-timings. In lieu of a normal baudrate, e.g. “250000”, it is possible to specify the BTR0 and BTR1 values directly as “0x8000<BTR1><BTR0>”, where BTRx stands for the bit-timing registers of the SJA1000 (or compatible) CAN controller, assuming a clock frequency of 16 MHz. For example, a value of 0x80001C03 specifies 87.5% sampling at 125 kbit/s.



Target configuration

General

Bootloader access:

Key 0: ******
Key 1: ******
Key 2: ******
Key 3: ******
Key 4: ******
Key 5: ******
Key 6: ******
Key 7: ******

Flash sectors to be erased:

A ☐ B ☐ C ☐ D ☐ E ☐ F ☐ G ☐ H ☐

☑ Smart sector selection

CAN

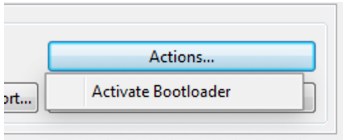
Bit-rate: 500000

TA: 0x770

SA: 0x778

When configuring CAN identifiers in the **Configure...** dialog, they can be entered as hex numbers (with ’0x’ or ’$’ prefix) or decimal numbers. An ’X’ postfix specifies an extended CAN identifier.

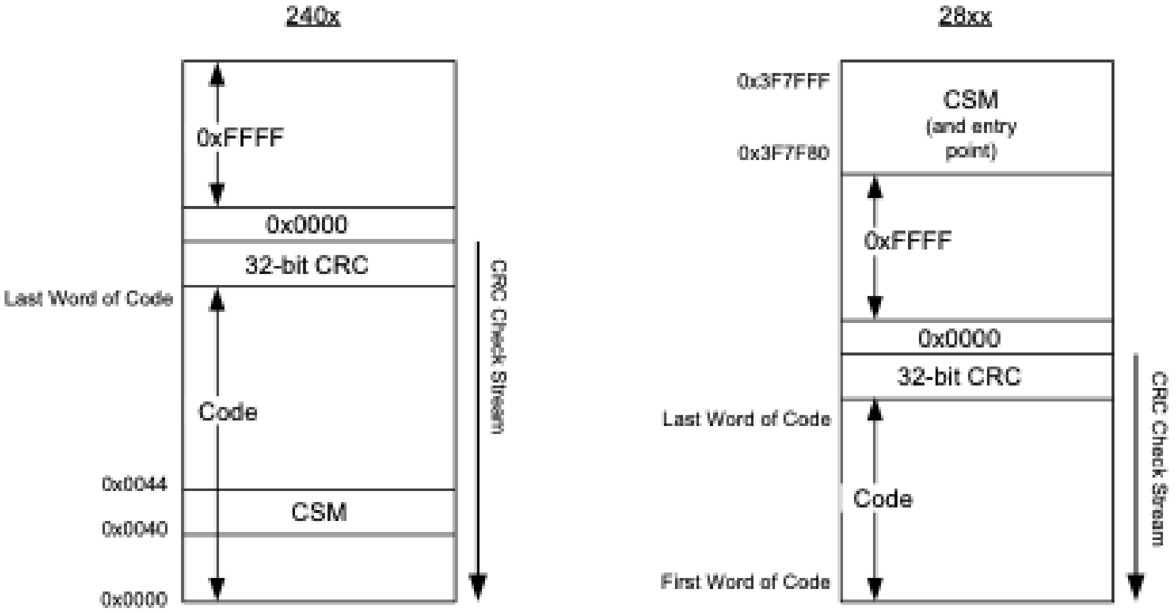
CodeSkin’s custom CAN bootloaders offer a mechanism to recover from unresponsive application code. This **Activate Bootloader** feature can be accessed from the **Actions...** menu.



See notes on bootloaders in the Appendix B on page 24 for more information regarding CAN bootloaders. We also encourage our users to contact CodeSkin for custom solutions.

3.5 CRC Checksum

C2Prog can be configured to automatically append a 32-bit CRC to the code being programmed into flash. This allows for the embedded application to verify the flash integrity at each powerup or even periodically during operation.



If the **Append CRC** box is checked, or the “-crc” command line option is used, C2Prog will first parse the binary-file and determine the lowest and highest address to be programmed. For a 240x MCU, this includes the CSM zone (4 keys), for a 28xx MCU, the CSM zone is ignored (including keys, reserved words, and program entry point). Next, the 32-bit CRC is calculated and appended at the top of the memory, i.e. at the two addresses above the highest address of the hex-file determined before. In addition, a CRC delimiter, one zero word (0x0000), is placed above the two CRC words.

The 32-bit CRC algorithm used has the following parameters:

- Polynomial: 0x04c11db7
- Endianess: big-endian
- Initial value: 0xFFFFFFFF
- Reflected: false
- XOR out with: 0x00000000

Test stream: 0x0123, 0x4567, 0x89AB, 0xCDEF results in CRC = 0x612793C3. Please refer to Appendix C on page 25 of this manual for a CRC32 implementation example.

In contrast to a typical data-stream with the CRC transmitted at the end, the C2Prog CRC must be verified by processing the flash data starting with the CRC, i.e. one memory address below the CRC delimiter (0x0000). A successful data-verify results in a CRC register value of zero (0x00000000).

A typical flash verification algorithm running at MCU powerup appears as follows:

1. Set a memory pointer to the highest possible program address.
2. Decrement the pointer until it points to the CRC delimiter (0x0000), skipping all 0xFFFF values.
3. Decrement the counter by one more address (at which time it points to the first CRC word).
4. Initialize the CRC register to 0xFFFFFFFF.
5. Update the register with the value addressed by the memory pointer (CRC polynomial: 0x04C11DB7).
6. Decrement memory pointer.
7. Repeat 5-6 until the memory pointer reaches the lowest program address.
8. If, at this point, the register holds 0x00000000, then the data integrity has been successfully verified.

Sample Code for 28xx with code in flash sector A:

```
#define FLASH_TOP (const uint16_t*)(0x3F7F7FL)
#define FLASH_BOT (const uint16_t*)(0x3F6000L)

const uint16_t* FlashPtr;
uint32_t CRCRegister;

FlashPtr = FLASH_TOP;
// search for CRC delimiter
while((*FlashPtr != 0x0000) && (FlashPtr > FLASH_BOT)){
FlashPtr--;
}
// process stream, CRC first
CRCRegister = 0xFFFFFFFFL;
while(FlashPtr > FLASH_BOT){
FlashPtr--;
// each CRC32Step() shifts one byte into the CRC register
CRCRegister = CRC32Step1(((*FlashPtr >> 8) & 0xFF), CRCRegister);
CRCRegister = CRC32Step(((*FlashPtr >> 0) & 0xFF), CRCRegister);
}
// at this point CRCRegister should be reading zero
```

3.6 Code Security

C2Prog will attempt to unlock the code security module (CSM) using the keys provided with the ”-keys” command-line option or **Configure...** dialog. In the case of 32-bit keys, each key is split into two 16-bit keys in big endian (BE) order.

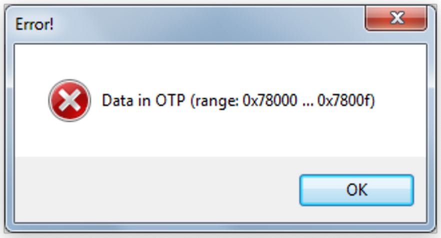
For older C2000 MCUs with the keys located at a fixed locations in erasable flash memory, C2Prog will attempt to extract the keys from the firmware image if default keys of all 0xFFFF are provided.

If the MCUs has the PSWDLOCK mechanism, specifying all 0x0000 in C2Prog (-keys, **Configure...**) will instruct C2Prog to attempt to unlock the CSM by reading the password locations in OTP. This will of course only work if the password lock has not been set and is therefore useful to verify production CSM settings.

☞ In case of MCUs with dual code security modules (DCSM), C2Prog only provides support for zone 1.

3.7 OTP

C2Prog helps prevent unintentional writing to one-time programmable (OTP) memory. Unless the user generates the Extended Hex file with the "-otp" option or checks the **Allow OTP Programming** box, C2Prog will not allow for the programming of the OTP locations.



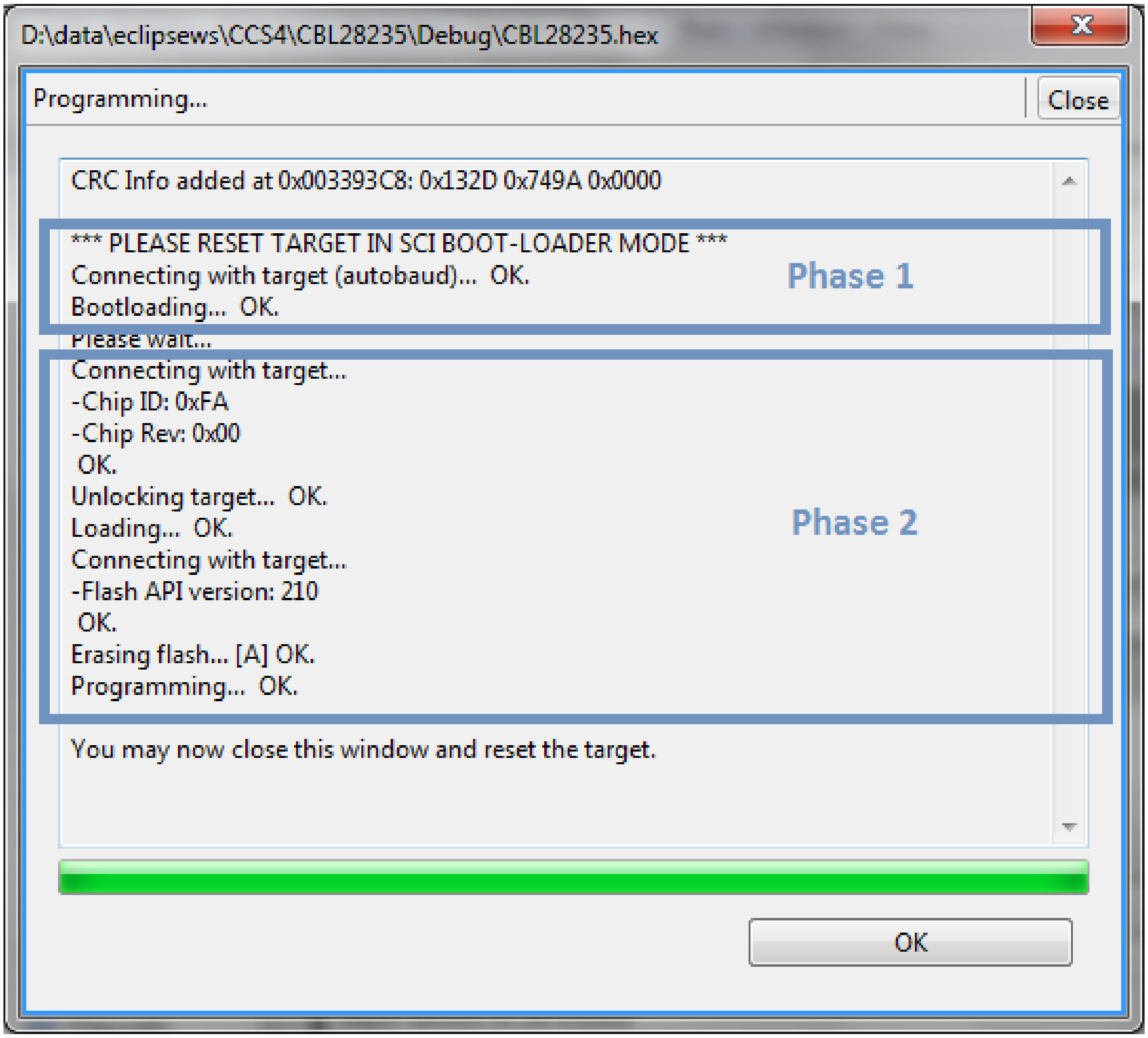
Data in OTP (range: 0x78000 ... 0x7800f)

3.8 Programming Sequence

The reflashing process is typically divided into two phases:

- Phase 1: Download of secondary bootloader (SBL)
- Phase 2: Execution of SBL, erasing of flash, download of application / flash programming

The screen capture below further illustrates the programming sequence for the serial mode in conjunction with TI's SCI bootloader. Please refer to the Appendix B on page 24 for more information on bootloaders.



```
CRC Info added at 0x003393C8: 0x132D 0x749A 0x0000

*** PLEASE RESET TARGET IN SCI BOOT-LOADER MODE ***
Connecting with target (autobaud)...  OK.
Bootloading...  OK.
Please wait...
Connecting with target...
-Chip ID: 0xFA
-Chip Rev: 0x00
 OK.
Unlocking target...  OK.
Loading...  OK.
Connecting with target...
-Flash API version: 210
 OK.
Erasing flash... [A] OK.
Programming...  OK.

You may now close this window and reset the target.
```

3.9 Extended Hex Files

The programming configuration, secondary bootloader, and contents of the binary file can be combined and saved as an "Extended Hex File" (*.ehx). This format is preferable over the raw binary file as it allows programming without requiring any manual configuration of the programmer options. An Extended Hex file can also be password protected. Thus, it is the ideal format for distributing programming files while also avoiding unauthorized use.

From the graphical user interface of the programmer an ehx file can be generated by clicking on the **Save as ehx...** button. The same can be done by calling C2Prog via its command line options, as below:

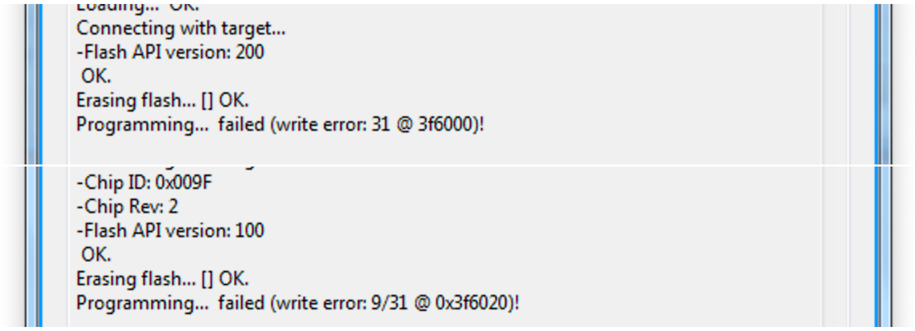
```
C2ProgShell.exe -create=test.ehx -target=28335_30MHz -bin=test.out
```

It is recommended that this command be configured as a post-build step in Code Composer Studio:

```
"C:\Program␣Files␣(x86)\C2Prog\C2ProgShell.exe" -create=${
    ↪ BuildArtifactFileBaseName}.ehx -target=28335_30MHz -bin=${
    ↪ BuildArtifactFileBaseName}.out
```

3.10 Error Codes

If an error occurs during programming, either a single error code or a pair of primary/secondary codes is displayed.



```
Loading... OK.
Connecting with target...
-Flash API version: 200
 OK.
Erasing flash... [] OK.
Programming... failed (write error: 31 @ 3f6000)!
```

```
-Chip ID: 0x009F
-Chip Rev: 2
-Flash API version: 100
 OK.
Erasing flash... [] OK.
Programming... failed (write error: 9/31 @ 0x3f6020)!
```

A singe number, or the primary code of a pair must be interpreted based on the type of MCU that is being programmed.

In case of 240x, 280x, 2802x, 2803x, 2805x, 2806x, 2823x and 2833x devices, the value corresponds to the error code reported by the TI flash API. Please refer to the relevant technical documentation for details.

For all other processors, and custom bootloaders licensed from CodeSkin, the single number, or primary code of a pair, indicates the following:

- 1: Unknown error
- 2: Feature not supported
- 3: Unlock error
- 4: Invalid size or alignment
- 5: Buffer overflow
- 6: Invalid address
- 7: Illegal sector
- 8: Erase error
- 9: Write error
- 10: Flash pump error
- 11: Flash FSM error

The secondary code of a pair identifies the flash API specific error code. When the primary code reads ”Flash FSM error”, the secondary code corresponds to the value of the FSM status. Please review the TI technical documentation for more details or contact CodeSkin for assistance.

3.11 Command Line Options

C2Prog can be launched from a command prompt (shell) with command-line options. This feature is available to facilitate the creation of ehx files as part of the code generation (for example, as a “final build step” in Code Composer Studio™). Users with a “professional” or “integration” C2Prog license can also program MCUs via the command line and extract hex files from ehx files.

The executable for this mode is **C2ProgShell.exe**.

C2Prog takes the following primary command line options:

-create=FILE_NAME	Creates ehx file
-program=EHX_FILE_NAME	Programs ehx file (licensed users only)
-extract=EHX_FILE_NAME	Extracts hex file from ehx file (licensed users only)

Primary options are augmented with a subset of the following secondary options:

-target=TARGET_ID	Selects the target – the target ID is composed of the target name, followed by an underscore “_” and the target option, for example “2812_30MHz”.
-bin=FILE_NAME	Selects binary file to be converted to an ehx file. For the extract command a hex-file name must be provided.
-keys=KEY1,KEY2, KEY3, …	Configures keys for unlocking flash (optional), KEYn is in 16-bit hex notation without ’0x’ prefix. In the case of 32-bit keys, each key is split into two 16-bit keys in big endian (BE) order. E.g. keys 0x01234567, 0x89ABCDEF, 0x11112222, 0x33334444 are specified as -keys= 0123,4567,89AB,CDEF,1111,2222,3333,4444.
-sectors=SECTOR_MASK	Configures which flash sectors are erased, where SECTOR_MASK is a hex number: -sectors=1: sector A -sectors=2: sector B -sectors=3: sectors A & B -sectors=A: sectors B & D If the “-sector” option is not used, then sectors to be erased are automatically detected.
-crc	Enables addition of CRC checksum (optional)
-otp	Allows OTP programming (optional)
-pass=PASS_PHRASE	Pass-phrase for extended hex-file
-baud=BAUDRATE	Baudrate for some protocols (such as CAN)
-ta=TARGET_ADDRESS	Target-address for multidrop protocols (such as CAN)
-sa=SOURCE_ADDRESS	Source address for multidrop protocols (such as CAN)
-port	Specifies communication port (for programming)
-progress	Enables the display of progress information (during programming)

The following command creates an extended hex file with password protection. All keys to unlock the flash are specified as 0x1234 and the sectors selected to be erased are A,B,C and D (hex 0xF = binary 1111).

```
C2ProgShell.exe -create=test.ehx -target="2811_30MHz" -bin=test.out -keys
   ↪ =1234,1234,1234,1234,1234,1234,1234,1234 -sectors=F -crc -pass="very␣
   ↪ secret"
```

The following command programs the flash using a password protected Extended Hex file:

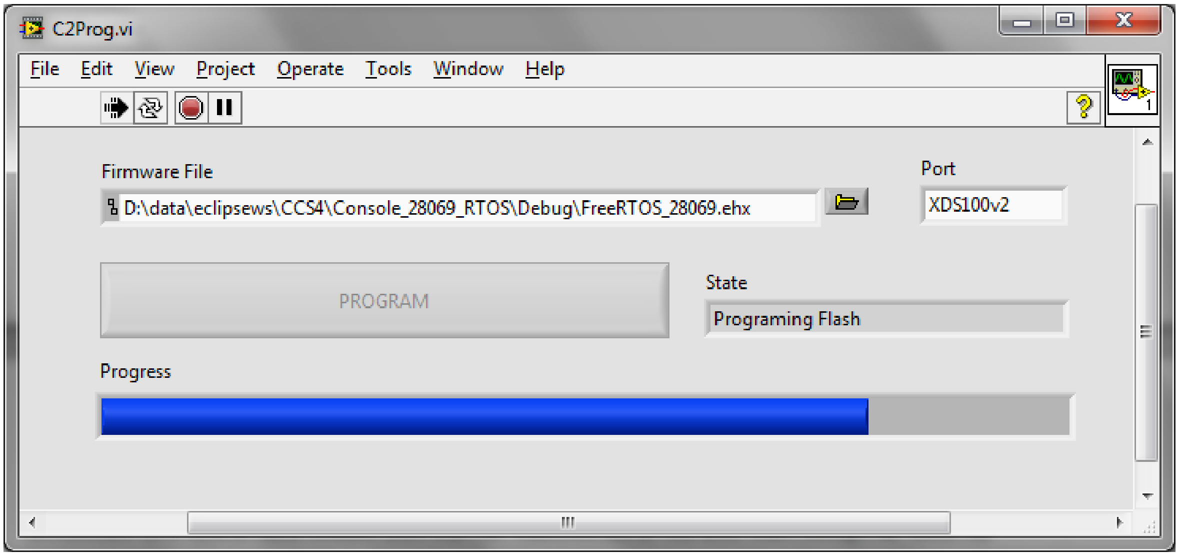
```
C2ProgShell.exe -program=test.ehx -pass="very␣secret" -port=COM1
```

This command extracts a hex file from an Extended Hex file:

```
C2ProgShell.exe -extract=test.ehx -pass="very␣secret" -bin=test.hex
```

3.12 DLL Interface

A powerful feature of C2Prog is that its core functionality is being exported via a 32-bit Dynamic-Link Library (DLL) for use by other applications, such as end-of-line (EOL) test-stands or field support tools [1]. With the C2Prog-DLL, flash programming capability can easily be added to custom applications generated by virtually any toolchain, including 32-bit NI LabView [2].



Firmware File: D:\data\eclipsews\CCS4\Console_28069_RTOS\Debug\FreeRTOS_28069.ehx

Port: XDS100v2

State: Programing Flash

The DLL is named “c2p.dll” and can be found in the “export” folder of the C2Prog installation. Below is a list of the principal function calls. Please refer to the Appendix D on page 26 for a more detailed description of the API.

- **c2pInitializeLibrary** — initializes the C2Prog environment
- **c2pProgram** — starts programming session (programs flash)
- **c2pGetProgressInfo** — retrieves information about the progress of the flash programming
- **c2pGetErrorDescription** — retrieves a textual description (string) for a given error code

---

☞ Currently, only an unmanaged 32-bit version of the DLL exists. Also note that the DLL cannot handle file paths containing non-English characters.

---

[1] The DLL interface is only available with the professional license of C2Prog.

[2] In LabView, the C2Prog DLL must be called from the GUI thread as the DLL is not thread-safe.

The function **c2pProgram** can be called as “blocking” or “non-blocking”. In blocking mode, the function will not return until the programming has completed (successfully, or with an error). In the non-blocking mode, the function returns immediately, and the progress of the programming (and its success) must be polled by means of the **c2pGetProgressInfo** call. This allows the application, which uses the C2Prog-DLL, to display progress information while the programming takes place.

⚠ It is important to understand that once a flash programming session has been initiated, it must be allowed to complete. Aborting the session in midstream may permanently damage the MCU.

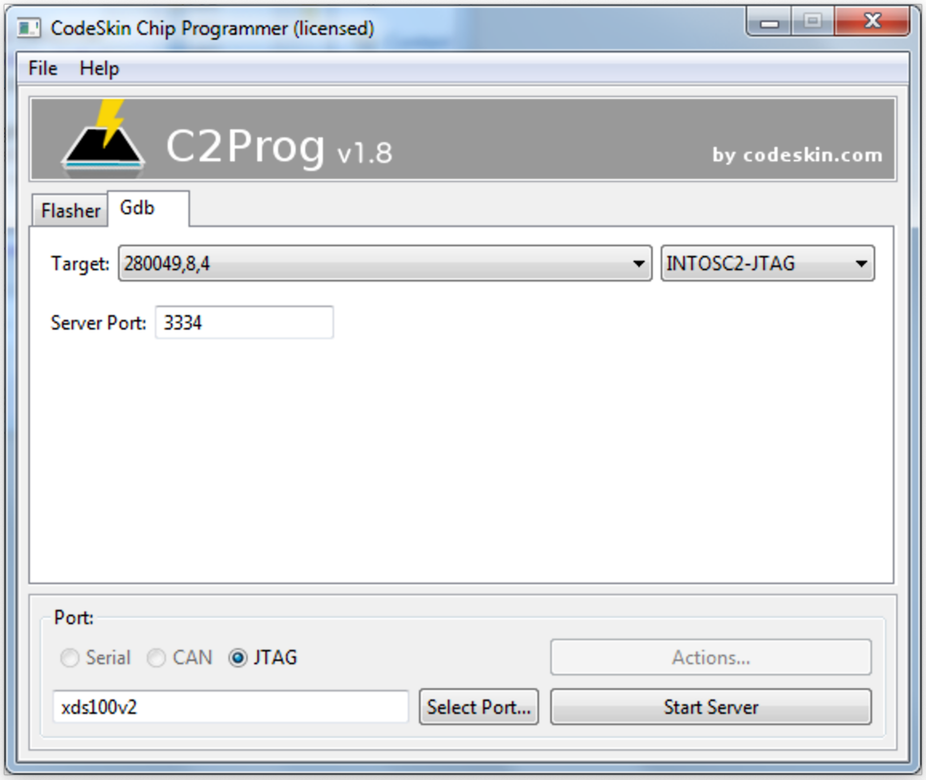
3.13 GNU Debug Server

For *primarily experimental* purposes, C2Prog includes a rudimentary GNU Debug Server stub.

A few notes about the server implementation:

- C28 cores have a 16-bit wide architecture. Memory access must therefore be made with an even number of bytes, where each pair of bytes is interpreted as a little endian 16-bit value.
- The server accepts multiple connections. This allows issuing a a blocking ’continue’ command and subsequently accessing the MCU, while it is running, through a second connection.

☞ The functionality of the Gdb server is still subject to change without notice. Please contact us if you wish to use this feature for important work.



Target: 280049,8,4 INTOSC2-JTAG

Server Port: 3334

Port: ○ Serial ○ CAN ● JTAG

xds100v2

Appendices

A License

Except where otherwise noted, all of the documentation and software included in the C2Prog package is copyrighted by CodeSkin, LLC.

Copyright (C) 2006-2019 CodeSkin, LLC. All rights reserved.

A free, limited-feature, Basic License is granted to anyone to use this software for any legal purpose, including commercial applications. Note, however, that C2Prog is not designed or suited for use in safety-critical environments, such as, but not limited to, life-support, medical and nuclear applications.

Users must review and accept the detailed licensing conditions as stated in the file `C2ProgLicense.pdf` located in the `doc` folder of the C2Prog installation.

C2Prog is using several open source software packages. For a comprehensive list of the libraries used, and their respective license conditions, please refer to **Help→About C2Prog**.

B About Bootloaders

When programming, C2Prog is interacting with a so called “bootloader” running on the target. This bootloader is often divided into two components:

- **Primary Bootloader (PBL)**: The primary bootloader is a small piece of code that is permanently programmed into the target and called immediately after reset. The primary bootloader can branch to the application code (if present) or receive the secondary bootloader (SBL) into RAM over the communication link and execute it. The primary bootloader typically also includes a security algorithm for unlocking the chip before the secondary bootloader can be loaded.

- **Secondary Bootloader (SBL)**: Contrary to the primary bootloader, the secondary bootloader is not permanently stored in the target. Instead, it is being loaded via the primary bootloader when needed. The SBL contains the flash programming algorithms. It erases the flash, receives the application code over the communication link, and programs the flash memory. Upon completion, the secondary bootloader can reset the chip.

There are several advantages to dividing the bootloader into two separate parts:

1. Since the primary bootloader does not include flash programming algorithms it can have a small footprint. Due to its low level of complexity it can be validated to a high level of confidence before it is shipped with a product.

2. The secondary bootloader contains the more complex algorithms. However, since it is not permanently programmed into the target, but rather distributed with the programming tool, it can be updated easily if needed.

3. Not having any flash programming algorithms permanently programmed in the target can be considered safer in the case of rogue behavior of the application code.

Almost all C2000™ MCUs ship with a primary bootloader programmed into the boot-ROM. While this bootloader supports several communication interfaces, only the SCI mode (RS-232) is of practical use in the field (the CAN implementation is too limited). CodeSkin has developed secondary RS-232 bootloaders for most C2000™ MCUs and distributes them with the C2Prog programming tool. The compiled versions of the secondary bootloaders are free; if so desired, the source code can be licensed for a fee.

CodeSkin also develops custom primary bootloaders that can be used in lieu of the TI version. They are licensed as source code, and allow for the implementation of customer specific features, such as servicing an external watchdog, and proprietary security/encryption algorithms. The CodeSkin primary bootloaders also support communication protocols other than RS-232. For example half-duplex RS-485, TCP/IP and CAN bus. Along with the primary bootloader, the source code of a matching secondary bootloader is provided.

C 32-bit CRC Algorithm

A basic implementation of a 32-bit CRC algorithm, optimized for code space, is provided in the listing below.

```
// 32-bit CRC lookup table (poly = 0x04c11db7)
uint32_t CRC32Lookup[16]={
0x00000000L, 0x04c11db7L, 0x09823b6eL, 0x0d4326d9L,
0x130476dcL, 0x17c56b6bL, 0x1a864db2L, 0x1e475005L,
0x2608edb8L, 0x22c9f00fL, 0x2f8ad6d6L, 0x2b4bcb61L,
0x350c9b64L, 0x31cd86d3L, 0x3c8ea00aL, 0x384fbdbdL
};

uint32_t CRC32StepNibble(uint16_t nibbleIn, uint32_t crc){
  uint16_t index;
  index = (uint16_t)(crc >> 28);
  crc = ((crc << 4) | (uint32_t)(nibbleIn)) ^ CRC32Lookup[index];
  return(crc);
}

uint32_t CRC32Step(uint16_t byteIn, uint32_t crc){
  uint16_t nibble;

  // first nibble
  nibble = (byteIn >> 4) & 0x0F;
  crc = CRC32StepNibble(nibble, crc);

  // second nibble
  nibble = (byteIn) & 0x0F;
  crc = CRC32StepNibble(nibble, crc);
  return(crc);
}
```

D C2Prog API

The following functions are provided by the C2Prog DLL. Note that the `c2p.dll` DLL must be loaded/unloaded using the Windows LoadLibrary / FreeLibrary functions. Function-pointers to the C2Prog API calls should be obtained by means of GetProcAddress. All API calls return a status information c2pStatus of type int. Use **c2pGetErrorDescription** to obtain an error description.

⚠ The C2Prog-DLL is not thread-safe and must therefore be called from one single thread only.

---

c2pStatus **c2pInitializeLibrary()**

Initializes the programming environment. Must be called by the application prior to using any other API functions.

Returns:

- zero (0), if call successful,
- error code, otherwise – use **c2pGetErrorDescription** for description of error

c2pStatus **c2pProgram**
(char* fileName, char* password, char* protocol, char* port, short wait)

Initiates flash programming. Note that only one flash programming session can be active at any time.

Parameters:

- fileName: Full path and name of ehx file
- password: Password to decrypt ehx file. If no password is used, provide an empty string (“”).
- protocol: Reserved for future use. The string “default” is recommended.
- port: Name of communication port
- wait: If set to 0, the function launches the programming session and returns immediately. Otherwise, the call blocks until the programming session terminates.

Returns:

- zero (0), if call successful,
- error code, otherwise – use **c2pGetErrorDescription** for description of error

Example:

```
   c2pProgram("D:\\data\\Firmware.ehx", "", "default", "XDS100v2", 1);
```

c2pStatus **c2pGetProgressInfo**
(int *state, double *progress, char* stateInfo, int infoStringMaxLen)

Obtains status information while programming session is active. This function is typically used after a non-blocking call to c2pProgram to display progress, status and error information. If a fault occurred during programming, the function returns an error code and the value of state indicates in which state the error occurred.

Parameters:

- state
    0. Idle / Done
    1. Establishing communication with PBL
    2. Downloading SBL
    3. Establishing communication with SBL
    4. Erasing flash
    5. Programming flash
    6. Finished
    7. Failed
- progress: Completion rate (0.5 = 50%, 1.0 = 100%)
- stateInfo: String describing state – memory allocated by caller
- infoStringMaxLen: Number of bytes allocated by caller to stateInfo string

Returns:

- zero (0), if call successful,
- error code, otherwise – use **c2pGetErrorDescription** for description of error

c2pStatus **c2pCloseProgrammingSession()**

Stops active programming session.

Warning: Closing an active programing session can leave the MCU in an unknown state. It his highly recommended that **c2pGetProgressInfo** be used instead to wait for the programming session to naturally terminate.
Returns:

- zero (0), if call successful,
- error code, otherwise – use **c2pGetErrorDescription** for description of error

c2pStatus **c2pExtract**(char* ehxFileName, char* password, char* hexFileName)

Obtains status information while programming session is active. This function is typically used after a non-blocking call to c2pProgram to display progress, status and error information. If a fault occurred during programming, the function returns an error code and the value of state indicates in which state the error occurred.

Parameters:

- ehxFileName: Full path and name of ehx file
- password: Password to decrypt ehx file. If no password is used, provide empty string (“”)
- hexFileName: Full path and name of hex file (note: existing file will be overwritten)

Returns:

- zero (0), if call successful,
- error code, otherwise – use **c2pGetErrorDescription** for description of error

c2pStatus **c2pPayloadHex**
(char* carrierHexFileName, int carrierDataWidth, char* payloadHexFileName,
int payloadDataWidth, long imageAddress, char* outHexFileName)

Converts the contents of a hex file into a payload image, and creates a new hex file which combines the contents of a carrier hex file with the payload image. For dual processors, this function can be used to stage a program image for core B into the flash memory of core A, that core A can transfer into core B. More details about the image format can be found after this description of the API calls on page 32.

Parameters:

- carrierHexFileName: Full path and name of hex file containing program data to which the image data will be added (note: this file is not modified by the function)
- carrierDataWidth: Data width (in bits) of carrier hex file
- payloadHexFileName: Full path and name of hex file that will be converted in to an image (note: this file is not modified by the function)
- payloadDataWidth: Data width (in bits) of payload hex file
- imageAddress: Address in carrier space at which image will be located
- outHexFileName: Full path and name of hex file to be generated, containing both the carrier and image data (note: existing file will be overwritten)

Returns:

- zero (0), if call successful,
- error code, otherwise – use **c2pGetErrorDescription** for description of error

c2pStatus **c2pGetErrorDescription**
(c2pStatus error, char* errorDescription, int errorStringMaxLen)

Obtains description for error code.

Parameters:

- error: Status returned by any of the API calls
- errorDescription: String describing error – memory allocated by caller
- errorStringMaxLen: Number of bytes allocated by caller to the errorDescription string

Returns:

- zero (0) if call successful,
- error code, otherwise

D.1 API Error Codes

0. NONE
1. OUT_OF_MEMORY – critical problem, report to CodeSkin
2. CANNOT_DETERMINE_JRE_PATH – critical problem, report to CodeSkin
3. CANNOT_LOAD_JVM_DLL – critical problem, report to CodeSkin
4. CANNOT_GET_JNI_CreateJavaVM_HANDLE – critical problem, report to CodeSkin
5. CANNOT_CREATE_JVM – critical problem, report to CodeSkin
6. CANNOT_LAUNCH_VM – critical problem, report to CodeSkin
7. CLASS_NOT_FOUND – critical problem, report to CodeSkin
8. METHOD_NOT_FOUND – critical problem, report to CodeSkin
9. UNABLE_TO_ATTACH_THREAD – critical problem, report to CodeSkin
10. UNABLE_TO_INVOKE – critical problem, report to CodeSkin
11. INVALID_HANDLE – critical problem, report to CodeSkin
12. LIBRARY_NOT_INITIALIZED – critical problem, report to CodeSkin
13. CANNOT_DETERMINE_C2P_PATH – invalid settings in ”c2p.config”
14. STRING_OVERFLOW – allocated string size too small
15. ARRAY_OVERFLOW – allocated array size too small
16. UNABLE_TO_ATTACH_NEWTHREAD – critical problem, report to CodeSkin
255. VM_NOT_LOADED – critical problem, report to CodeSkin
256. ERROR_UNKNOWN – critical problem, report to CodeSkin
1024. NO_ERROR
1025. UNKNOWN_ERROR – critical problem, report to CodeSkin
1026. CANNOT_OPEN_PORT – unable to open communication port
1027. NO_RESPONSE – no response from target
1028. INVALID_RESPONSE – invalid response from target
1029. OPERATION_TIMED_OUT – no response to issued command
1030. ERASE_ERROR – flash erase failed
1031. PROGRAM_ERROR – flash write failed
1032. UNLOCK_ERROR – unable to unlock security
1033. CANNOT_RESET – end of reflashing session resulted in error, likely failure of image integrity check
1034. CANNOT_INSTANTIATE_BOOTLOADER – ehx file not compatible with C2Prog version
1035. CANNOT_CONFIGURE_BOOTLOADER – ehx file not compatible with C2Prog version
1036. CANNOT_INSTANTIATE_PROGLOADER – ehx file not compatible with C2Prog version
1037. CANNOT_CONFIGURE_PROGLOADER – ehx file not compatible with C2Prog version
2048. NO_ERROR
2049. LICENSE_ERROR – invalid license
2050. NO_SUCH_TARGET – invalid target selection (”-target” command line option)
2051. LIC_CONDITIONS_ATTACHED – ehx has license conditions attached, can only be programmed from GUI
2052. FILE_IO_EXCEPTION – unable to read file
2053. CHECKSUM_ERROR – ehx has invalid checksum or is not properly signed for use with

integration license

2054. ENCRYPTION_ERROR – unable to decrypt ehx file
2055. PROG_IN_PROCRESS – programming session already in progress
2056. CRC_OR_SECTOR_VIOLATION – appended CRC or hex-file contents falls outside flash memory range
2057. UNKNOWN_ERROR – critical problem, report to CodeSkin

E Payload Image Format

The image created by the **c2pPayloadHex** command consist of a sequence of N Data Records followed by a Termination Record.

Firmware Image					
Data Record #0	Data Record #1	Data Record #2	...	Data Record #N-1	Termination
n0 bytes	n1 bytes	n2 bytes		nN-1 bytes	4 bytes

Data Records are formatted as follows:

Data Record		
Record Data Length k (measured in bytes)	Record Data	CRC
2 bytes	k bytes	2 bytes

The CRC is a 16-bit value calculated over entire Data Packet (k+2 bytes) (CCITT polynomial: $x^{16}+x^{12}+x^5+1$ initial value = $0x\mathrm{FFFF}$). All values are stored in Big Endian notation, starting with the most significant byte (MSB).

The Record Data contains image data as well as the address to which the data needs to be copied when the image is deployed (target address).

Record Data	
Target address	Data
4 bytes	k-4 bytes

The Termination Record has a length of 4 bytes:

Termination Record	
Record Data Length k (measured in bytes)	CRC
0x0000	0x1D0F