C2Prog User Manual

Version 2.0

July 4, 2022

Disclaimer

Contents

1 Introduction

C2Prog consists of a collection of tools for programming TI C2000™ MCUs.

Besides reflashing over JTAG, C2Prog also supports reflashing over RS-232, RS-485, TCP/IP and Controller Area Network (CAN). It is, therefore, well suited for deployment in the field where the JTAG port is typically not accessible.

Some salient features of C2Prog 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`).
- JSON-RPC server for using C2Prog from a CI/CD pipeline.
- Flexible and modular design allowing for customer specific solutions.
- Small footprint application.
- Cross-platform (Windows, macOS, Linux) binaries for Intel and ARM.

C2Prog includes the following modules:

- **C2Prog** — graphical user interface
- **c2p-cli** — headless command line interface
- **c2p-server** — JSON-RPC server
- **c2p-gdb** — GDB client with support for scripting
- **c2p-dll** — DLL implementing C2Prog API

☞ **C2Prog v2** represents a complete overhaul of C2Prog v1. The software has been rewritten in C++, with the goal of increasing the speed and portability of the application. Note that this release is still a preview and does not yet include the entirety of the v1 functionality.

---

2 Quick Start

2.1 Installation

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

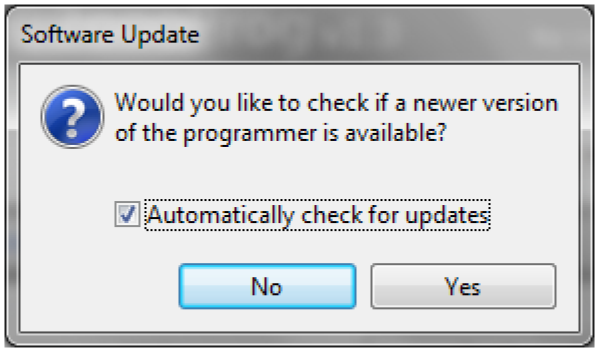
`https://www.codeskin.com/C2Prog.html`

For Windows, an installer **setup.exe** is provided. C2Prog can also be uninstalled from the control panel similar to other Windows software.

For macOS, the C2Prog application is distributed as a disk image. Simply mount the image and copy the application to a location of your choosing.

For Linux, the application files are contained in a zip archive that can be extracted in a suitable location.

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.



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:

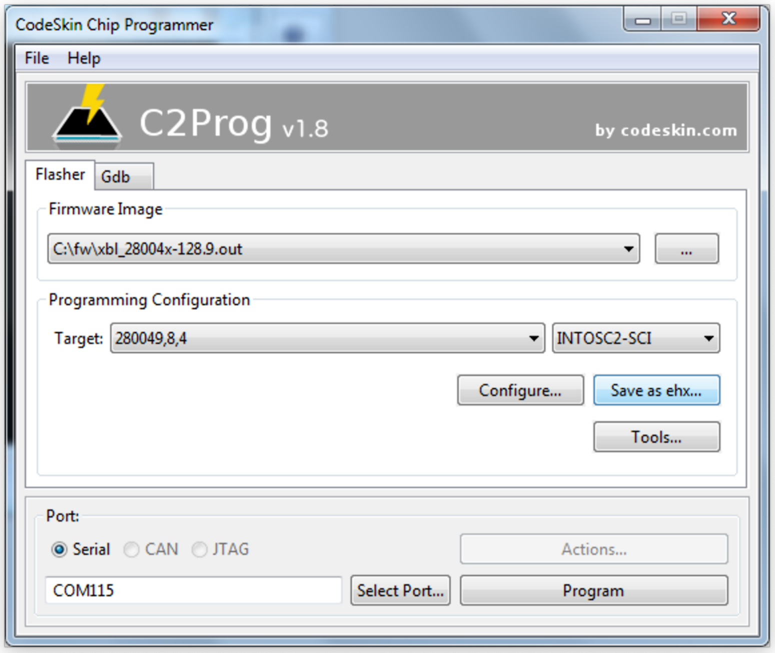
```
   hex2000 -romwidth 16 -memwidth 16 -i -o .\Debug\test.hex .\Debug\test.out
```

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.



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

Target: 280049,8,4 — INTOSC2-SCI

Port: (● Serial) COM115

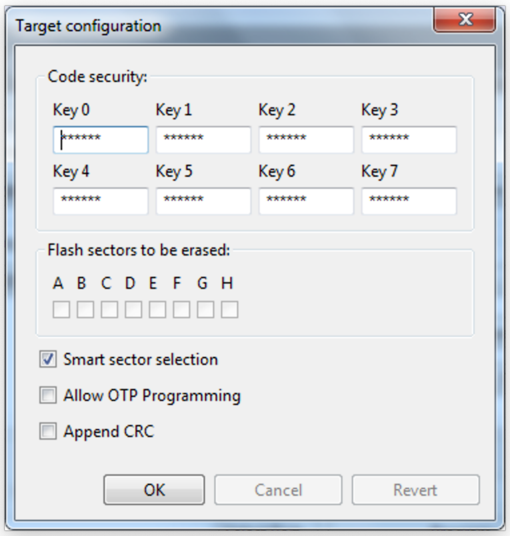
Activate the **Flasher** tab and select the **Firmware Image** (binary file) by means of the **...** popup menu.

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 10). Contact CodeSkin at info@code-skin.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.

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

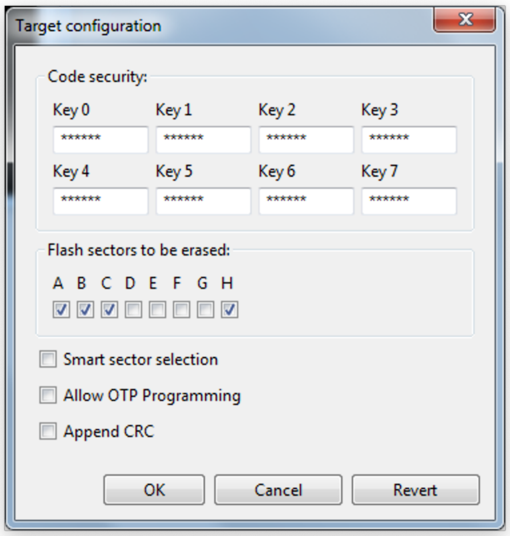
Code security:

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**
******	******	******	******

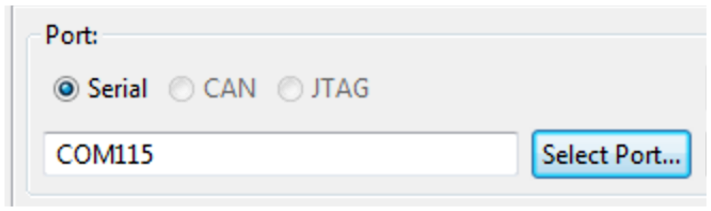
Flash sectors to be erased:

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:

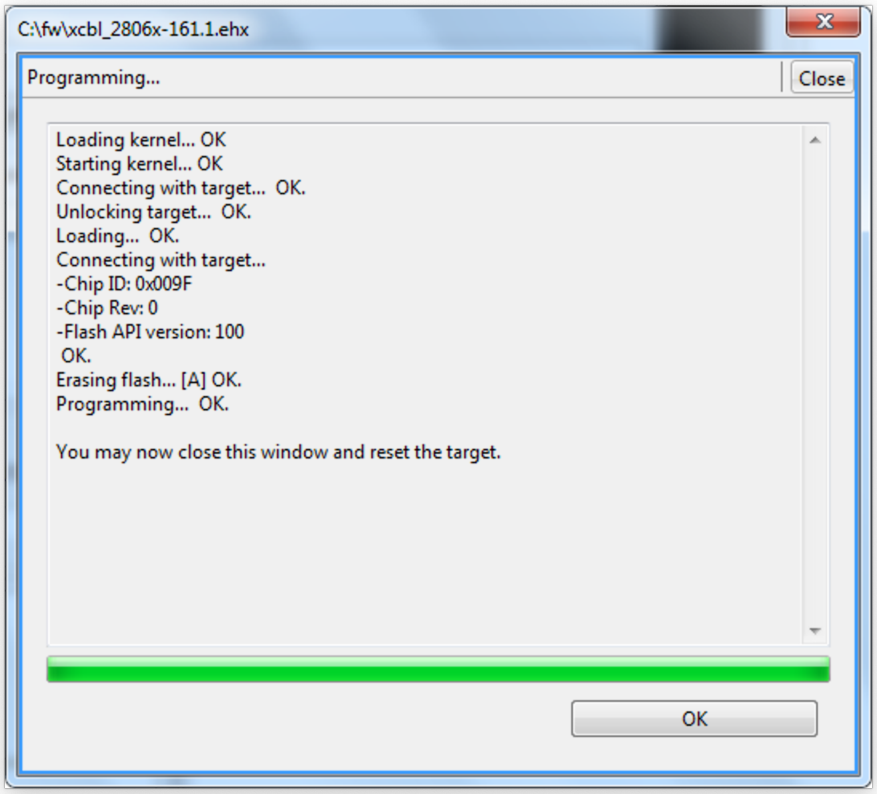
[✓ Serial] CAN JTAG

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.



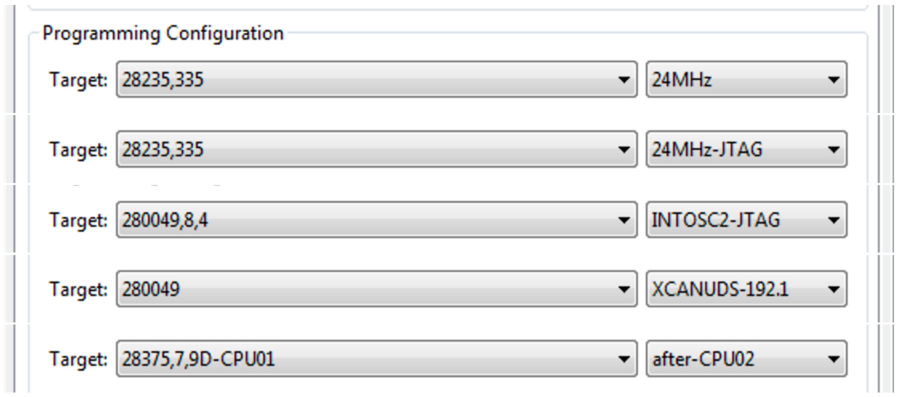
```
Loading kernel... OK
Starting kernel... OK
Connecting with target...  OK.
Unlocking target...  OK.
Loading...  OK.
Connecting with target...
-Chip ID: 0x009F
-Chip Rev: 0
-Flash API version: 100
 OK.
Erasing flash... [A] OK.
Programming...  OK.

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:



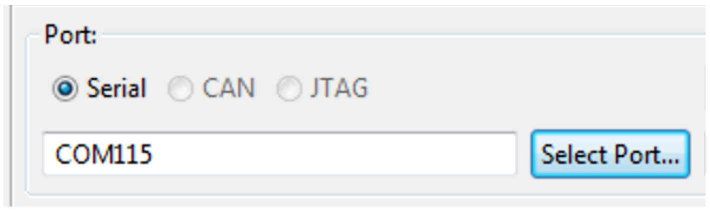
Target:	28235,335	24MHz
Target:	28235,335	24MHz-JTAG
Target:	280049,8,4	INTOSC2-JTAG
Target:	280049	XCANUDS-192.1
Target:	28375,7,9D-CPU01	after-CPU02

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.

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

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.

3.3 Programming over JTAG

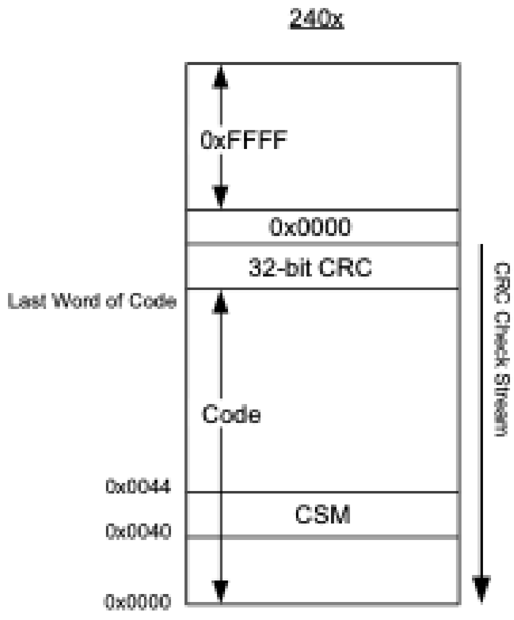
☞ This functionality will become available in a future 2.x release. Please use C2Prog v1.9 in the meantime.

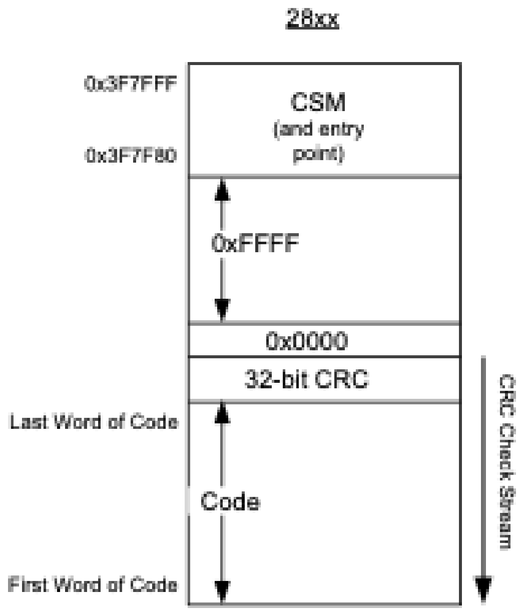
3.4 Programming over CAN

☞ This functionality will become available in a future 2.x release. Please use C2Prog v1.9 in the meantime.

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 24 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.

⚠ The security zone will not be truly protected until the corresponding PWDLOCK is set. Please make sure that you review the relevant documentation from TI.

3.6.1 DCSM

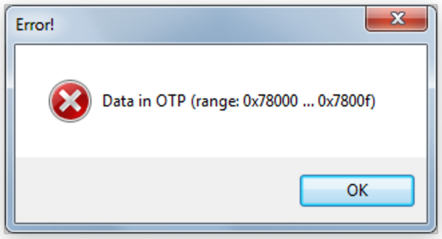
In case of MCUs with dual code security module (DCSM), C2Prog will automatically determine which zone must be unlocked in order to program a particular firmware image.

⚠ Do not set the PWDLOCK of a zone unless the sector assignment for that zone has been fully configured by either explicitly requesting, or not requesting, the zone for each sector. See TI technical documentation of GRABSECT registers.

Included with C2Prog are GDB scripts to analyze the CSM status of a particular device. See Section 3.14 on page 21 for more details on how to run such scripts.

3.7 OTP

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


Error!

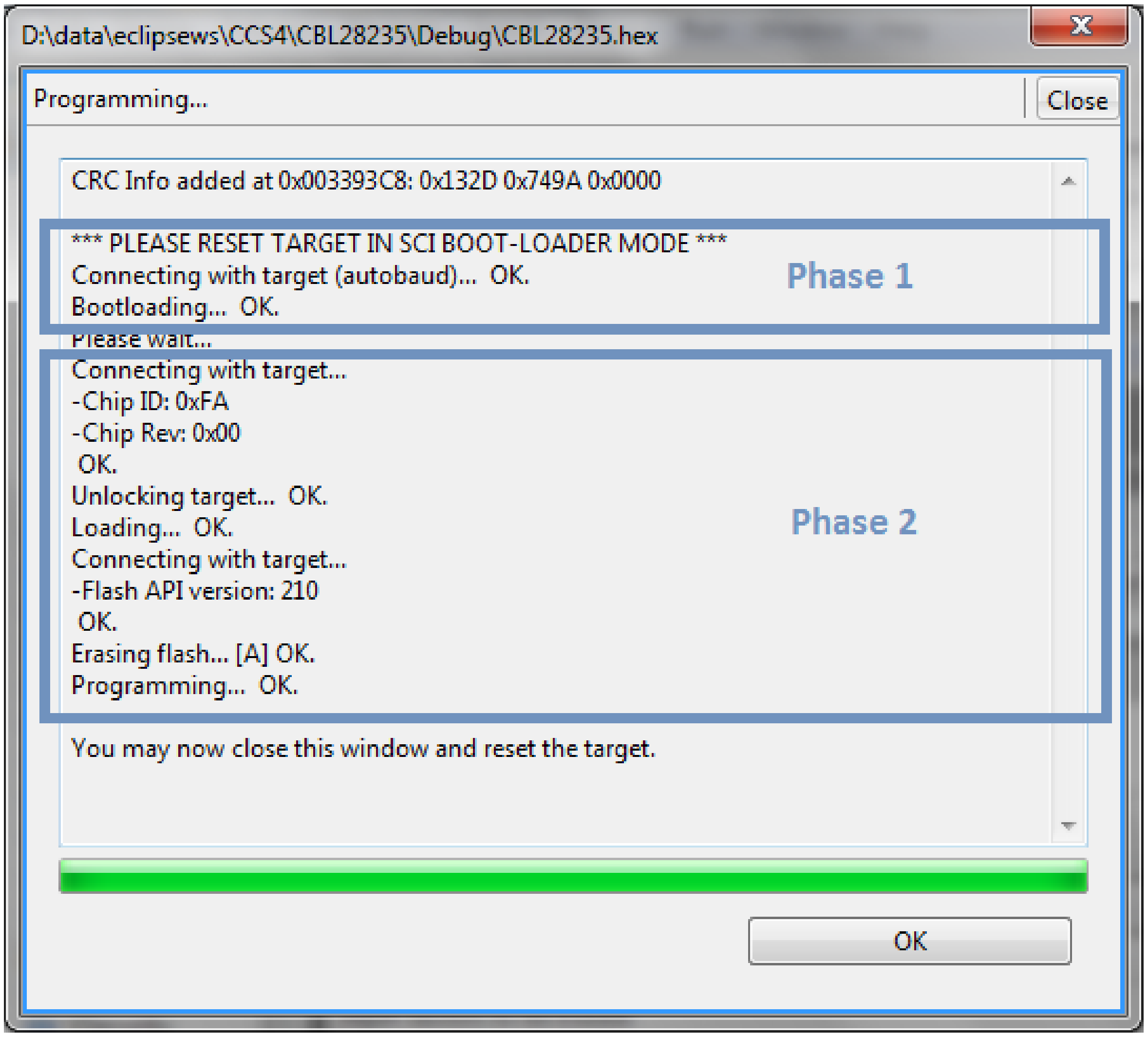
Data in OTP (range: 0x78000 ... 0x7800f)

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 23 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:

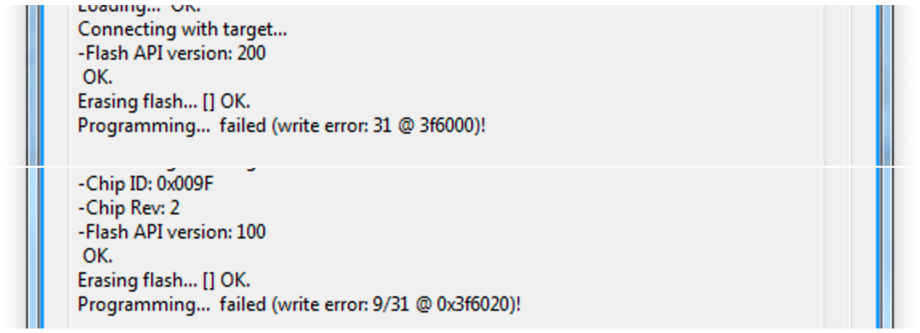
```
c2p-cli mkehx --target=28335_30MHz test.out
```

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

```
"C:\Users\joe\AppData\Local\Programs\C2Prog 2.x\c2p-cli.exe" mkehx --target
    ↪ =28335_30MHz ${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
- 12: OTP ECC write error
- 13: OTP data write 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.

The corresponding executable is **c2p-cli**.

c2p-cli accepts the following primary commands and options:

c2p-cli −−help	Display help
c2p-cli −−version	Display version information
c2p-cli mkehx [options]	Create ehx file
c2p-cli program [options]	Program ehx file (licensed users only)

The command for creating an ehx file is

```
c2p-cli mkehx --target=<target> [options] <binary file> [<ehx file>]
```

For example, the following command creates an extended hex file that is protected by a passphrase. 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).

```
c2p-cli mkehx --target=2811_30MHz c:\test.out --keys
    ↪ =1234,1234,1234,1234,1234,1234,1234,1234 --sector-mask=F --append-
    ↪ crc --passphrase="very secret"
```

The command for programming an ehx file is

```
c2p-cli program --port=<port> [options] <ehx file>
```

For example

```
c2p-cli program --port=COM5 test.ehx
```

mkehx 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".
−−keys=KEY1,KEY2,...	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.
−−sector-mask=SECTOR_MASK	Configures which flash sectors are erased, where SECTOR_MASK is a hex number: −−sector-mask=1: sector A −−sector-mask=2: sector B −−sector-mask=3: sectors A & B −−sector-mask=A: sectors B & D If the "−−sector-mask" option is not used, then sectors to be erased are automatically detected.
−−append-crc	Enables addition of CRC checksum (optional)
−−allow-otp	Allows OTP programming (optional)
−−passphrase=PASS_PHRASE	Passphrase for extended hex-file

program options	
−−port	Specifies communication port
−−passphase=PASS_PHRASE	Passphrase for extended hex-file
−−print-progress	Enables the display of progress information

3.12 JSON RPC Interface

The C2Prog functionality can be accessed via `JSON RPC`, a lightweight remote procedure call protocol.
The executable for launching the JSON RPC server is **c2p-server**.

```
c2p-server --rpc-port=<port>
```

For example

```
c2p-server --rpc-port=8080
```

A JSON RPC client can then connect to the C2Prog server and issue remote procedure requests.

3.12.1 Load command

The `load` method initiates a programming session.

```
{
  "method": "load",
  "params": {
    "file":"<ehx file path>",
    "port":"<port>",
    "pass":"<passphrase>"
  },
  "jsonrpc": "2.0",
  "id": 0
}
```

For example:

```
{
  "method": "load",
  "params": {
    "file":"c:\test.ehx",
    "port":"COM5",
    "pass":"very secret"
  },
  "jsonrpc": "2.0",
  "id": 0
}
```

The server response to the `load` command is formatted as follows:

```
{
  "result": {
    "info":"",
    "error":"<error description>",
    "progress":0.0
  },
  "jsonrpc": "2.0",
  "id": 0
}
```

A non-empty `error` string signals that the `load` command failed.

3.12.2 Get Status command

The `get-status` method can be used to query the status of on ongoing flashing session.

```
{
  "method": "get-status",
  "jsonrpc": "2.0",
  "id": 0
}
```

The response to `get-status` is identical to the `load` method response:

```
{
  "result": {
    "info":"<status description>",
    "error":"<error description>",
    "progress":<[0.0-1.0]>
  },
  "jsonrpc": "2.0",
  "id": 0
}
```

The programming session is complete once `progress` reaches `1.0`. If an error occurs, it is indicated my means of the `error` field.

3.12.3 Shutdown command

The server can be shut down by means of the `shutdown` method.

```
{
  "method": "shutdown",
  "jsonrpc": "2.0",
  "id": 0
}
```

3.13 GNU Debug Server

☞ This functionality will become available in a future 2.x release. Please use C2Prog v1.9 in the meantime.

3.14 GDB Client

C2Prog includes a GDB client **c2p-gdb** that can be used to interact with a GDB server, such as the C2Prog GDB stub, Segger J-Link GDB Server or `OpenOCD`.

c2p-gdb accepts the following primary commands and options:

c2p-gdb --help	Display help
c2p-gdb script [options] <filename>	Run Lua script
c2p-gdb load [options] <filename>	Load binary file

c2p-cli options	
--endian=[big]/[little]	Target endianess
--lausize=<lau size>	Size of least addressable unit
--url=<server url>	Server URL
--port=<server port>	Server port
--nvic=<nvic base>	NVIC base address (ARM only)
--server-cmd="<cmd>"	GDB server launch command
--server-start-delay=<delay>	Server startup delay [ms]

The following command connects to port 3333 of a GDB server and executes the script contained in **my_script.lua**:

```
./c2p-gdb script --port 3333 my_script.lua
```

Scripts are written in the Lua programing language using **gdb** methods to interact with the Gdb server.

```
-- reset target
gdb.restart()
-- configure boot from flash
gdb.write_memory(0xD00, {0x55aa, 0x0b}, 2)
-- start application
gdb.cont()
```

The next command connects to port 3333 of a GDB server and programs a target with the binary file **my_firmware.elf**. Note that providing the `--nvic` option for ARM processors will run the application automatically after it has been programmed.

```
./c2p-gdb load --port 3333 --nvic 0x8000000 ~/Desktop/my_firmware.elf
```

With the `--server-cmd` and `--server-start-delay` options it is possible to automatically launch the GDB server for the duration of the interaction with the GDB client.

```
./c2p-gdb load --server-cmd="~/opt/openocd/bin/openocd -s ~/opt/openocd/
    ↪ share/openocd/scripts -f board/stm32g431.cfg" --server-start-delay
    ↪ 1000 --port 3333 --nvic 0x8000000 ~/Desktop/my_firmware.elf
```

3.15 DLL Interface

☞ This functionality will become available in a future 2.x release. Please use C2Prog v1.9 in the meantime.

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) 2006-2022 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.

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.

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);
}
```