UG162: Simplicity Commander Reference Guide

This document describes how and when to use the Command-Line Interface (CLI) of Simplicity Commander. Only EFR32 is supported for this release. EM3xx is not supported at this time.

This document is intended for software engineers, hardware engineers, and release engineers. Silicon Labs recommends that you review this document to familiarize yourself with the CLI commands and their intended uses. You can refer to specific sections of this document to access operational information as needed. This document also includes examples so you can gain an understanding of Simplicity Commander in action.

This document is up-to-date with Simplicity Commander 0.15. See 6, **Revision History**, for a list of new features and commands for previous versions of the application.

KEY FEATURES

- Introduces Simplicity Commander.
- Adds new features and commands.
- Describes the file formats supported by Simplicity Commander.
- Includes detailed syntax of all Simplicity Commander commands and example command line inputs and outputs.

1 Introduction

Simplicity Commander is a single, all-purpose tool to be used in a production environment. It is invoked using a simple Command-Line Interface (CLI) that is also scriptable. Simplicity Commander enables customers to complete these essential tasks:

- Flash their own applications.
- Configure their own applications.
- Create binaries for production.

This release of Simplicity Commander supports only EFR32; EM3xx is not supported at this time. Simplicity Commander is designed to support the Silicon Labs Wireless STK platform.

The primary intended audience for this document is software engineers, hardware engineers, and release engineers who are familiar with programming the EFR32. This reference guide describes how to use the Simplicity Commander CLI. It provides general information on file formats supported by EFR32 and includes details on using the Simplicity Commander commands, options, and arguments. It also includes example command line inputs and outputs so you can gain a better understanding of how to use Simplicity Commander effectively.

2 File Format Overview

The EFR32 works with different file formats: .s37, .ebl, and .hex. Each file format serves a slightly different purpose. The file formats supported by Simplicity Commander are summarized below.

2.1 Motorola S-record (s37) File Format

Silicon Labs uses the Simplicity Studio as its Integrated Development Environment (IDE) and leverages the IAR Embedded Workbench for ARM platforms. This tool combination produces Motorola S-record files, s37 specifically, as its output.[1] In Silicon Labs development, an s37 file contains programming data about the built firmware and generally only represents a single piece of firmware—application firmware or bootloader firmware—but not both. An application image in s37 format can be loaded into a target EFR32 using the Simplicity Commander `flash` command. The s37 format can represent any combination of any byte of flash in the EFR32. The Simplicity Commander `convert` command can be used to convert the s37 format to ebl format for use with Ember bootloaders. This command can also be used to read multiple s37 files, ebl, and hex files; output an s37 file for combining multiple files into a single file; and modify individual bytes of a file.

2.2 Ember Bootloader (ebl) File Format

The Ember Bootloader (ebl) file format is generated by the Simplicity Commander `convert` command. The ebl file format can only represent an application image; it cannot be used to capture Simulated EEPROM token data (as described by *AN703, Using the Simulated EEPROM for the EM35x and Might Gecko (EFR32MG) SoC Platforms*), firmware for a bootloader, or data contained outside of the main flash block.

The ebl file format is designed to be an efficient and fault-tolerant image format for use with the Ember bootloader to upgrade an application without the need for special programming devices. The bootloader can receive an ebl file either over-the-air (OTA) or via a supported peripheral interface, such as a serial port, and reprogram the flash in place.

Although the ebl file format is intended for use with a bootloader, the Simplicity Commander `flash` command is also capable of directly programming an ebl image. This file format is generally used in later stage development, and for upgrading manufactured devices in the field. The standalone bootloader should never be loaded onto the device as an ebl image. Use the s37 file format when loading the bootloader itself. Refer to UG103.6, Application Development Fundamentals: Security, for more details about the proprietary ebl file format.

2.3 Intel HEX-32 File Format

Production programming uses the standard Intel HEX-32 file format. The normal development process for EFR32 chips involves creating and programming images using the s37 and ebl file formats. The s37 and ebl files are intended to hold applications, bootloaders, manufacturing data, and other information to be programmed during development. The s37 and ebl files, though, are not intended to hold a single image for an entire chip. For example, it is often the case that there is an s37 file for the bootloader, an s37 file for the application, and an s37 file for manufacturing data. Because production programming is primarily about installing a single, complete image with all the necessary code and information, the file format used is Intel HEX-32 format. While s37 and hex files are functionally the same—they simply define addresses and the data to be placed at those addresses—Silicon Labs has adopted the conceptual distinction that a single hex file contains a single, complete image often derived from multiple s37 files. The Simplicity Commander `convert` command can be used to read multiple hex files, and ebl and s37 files; output a hex file for combining multiple files into a single file; and modify individual bytes of a file.

Note: Simplicity Commander is capable of working identically with s37 and hex files. All functionality that can be performed with s37 files can be performed with hex files. Ultimately, with respect to production programming, Simplicity Commander `flash` command allows the developer to load a variety of sources onto a physical chip. The `convert` command can be used to merge a variety of sources into a final image file and modify individual bytes in that image if necessary.

[1] See http://en.wikipedia.org/wiki/S_record for more information on Motorola S-record file format.

Table 1 summarizes the inputs and outputs for the different file formats used by Simplicity Commander.

Table 1. File Format Summary

	Inputs					Outputs				
	ebl	s37	hex	bin	chip	ebl	s37	hex	bin	chip
flash	X	X	X	X						X
readmem					X		X	X	X	
convert	X	X	X	X		X	X	X	X	

3 General Information

3.1 Command Line Syntax

To execute Simplicity Commander commands, start a Windows command window, and change to the Simplicity Commander directory. The general command line structure in Simplicity Commander looks like this:

```
commander [command] [options][arguments]
```

where:

- `commander` is the name of the tool
- `command` is one of the commands supported by Simplicity Commander, such as, `flash`, `readmem`, `convert`, etc. The command-specific help provides additional information on each command.
- `option` is a keyword that modifies the operation of the command. Options are preceded `--` (double dash) as described in "Command Line Structure" and for each command. Some commands have single-character short versions which are preceded by – (single dash). Refer to the command-specific help for the single-dash shorthands.
- `argument` is an item of information provided to Simplicity Commander when it is started. An argument is commonly used when the command takes one or more input files.
- square brackets indicate *optional* parameters as in this example: `commander flash [filename(s)] [options]`
- angle brackets indicate *required* parameters as in this example: `commander readmem –-output <filename>`

3.2 General Options

3.2.1 Help (--help)

Displays help for all Simplicity Commander commands and command-specific help for each command.

Command Line Usage

```
$ commander –-help
```

Command Line Usage Output

Simplicity Commander help displays a list of all Simplicity Commander commands as shown in Figure 1.

```
C:\SiliconLabs\Simplicity Commander>commander --help

Usage: commander [command] [options]

Simplicity Commander

Each command listed below has its own set of options and arguments.
Run 'commander <command> --help' to get specific help and usage descriptions for
 each command.

Options:
  -?, -h, --help  Displays this help.
  -v, --version   Displays version information.

Arguments:
  command         The command to execute

Commands:
  adapter         Adapter commands.
  convert         Convert or combine one or more input files to one output file.
  device          Device commands.
  ebl             Encrypt, decrypt and other handling for EBL files.
  extflash        External SPI flash commands.
  flash           Write data to the target flash.
  readmem         Read memory from a device.
  tokendump       Read and dump tokens from a device or an image file.
  tokenheader     Generate a C header file from a custom token group.
  verify          Verify the current flash contents.
DONE
```

Figure 1. Simplicity Commander Help

To display help on a specific Simplicity Commander command, enter the name of the command followed by `–-help.`

Command Line Input Example

```
$ commander flash --help
```

Command Line Output Example

Simplicity Commander displays help for the flash command in Figure 2.

```
C:\SiliconLabs\Simplicity Commander>commander flash --help

Usage: commander flash [filename(s)] [options]
Write one or more files to the target flash.

Options:
  -?, -h, --help                          Displays this help.
  -v, --version                           Displays version information.
  --device, -d <device>                   The device, device family or platform to
                                          target. Examples of strings that are
                                          understood: "EFR32MG1P233F256GM48",
                                          "EFR32MG", "EFR32", "EFR32F256". Required

                                          for some operations.
  --force                                 Force operation. This will convert
                                          non-fatal errors to warnings, allowing
                                          the process to continue.
  --serialno, -s <serial number>          J-Link serial number.
  --ip <IP>                               IP Address.
  --speed <speed in kHz>                  Debug interface speed.
  --tif <SWD|JTAG|C2>                     Target debug interface.
  --address <address>                     Address to flash to. Not applicable for
                                          hex or s37 files which contain address
                                          information.
  --halt                                  Leave the target halted after flashing.
  --masserase                             Supply this to do a mass erase of the
                                          entire chip before flashing. Otherwise
                                          only affected pages are erased.
  --noverify                              Don't verify contents written to flash
                                          (verification is enabled by default).
  --patch, -p <address:data[:length]>     Patch memory contents.
                                          Data is interpreted as an unsigned
                                          integer. The optional length parameter
                                          can be used to define the number of bytes

                                          write, up to 8.
  --token <TOKEN_NAME:value>              Single token with its new value.
  --tokenfile <filename>                  File describing tokens to write.
  --tokengroup <tokengroup>               Which set of tokens to use. Supported:
                                          znet

Arguments:
  flash
  filename(s)                             File(s) to flash.

DONE
```

Figure 2. Simplicity Commander Flash Command Help

3.2.2 Version (--version)

Displays the version information for Simplicity Commander, J-Link DLL, and EMDLL, and a list of detected USB devices. If you use this option in conjunction with another command or command/option, Simplicity Commander displays this extra information before any command is executed.

Command Line Usage

```
$ commander –-version
```

Command Line Usage Output

Simplicity Commander displays version information in Figure 3.

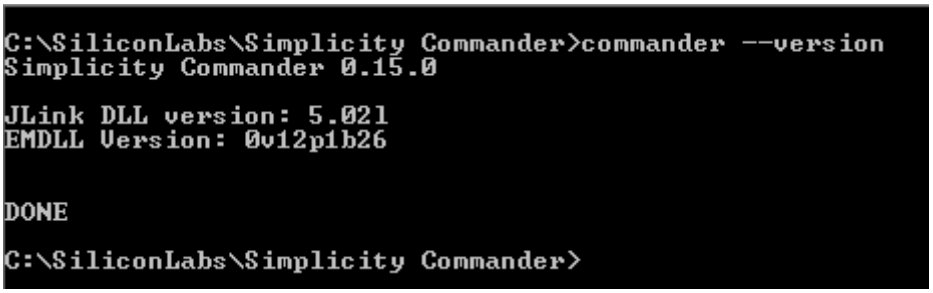


Figure 3. Simplicity Commander Version Information

3.2.3 Device (--device <part number>)

Specifies a target device for the command. If this option is supplied, no auto-detection of the target device is used. In some cases, such as when using `convert` with the --`token` option, this option is required.

For convenience, Simplicity Commander attempts to parse the `--device` option so that a complete part number is normally not required as a command input. For example, Simplicity Commander interprets `commander --device EFR32` to mean that the selected device is an EFR32, which has implications regarding the memory layout and available features of this specific device. As another example, Simplicity Commander interprets `--device EFR32F256` as an EFR32 with 256 kB flash memory.

Using a complete part number such as `--device EFR32MG1P233F256GM48` is always supported and recommended.

Command Line Usage Example

```
$ commander convert --device EFR32F256 --outfile image.ebl image.s37
```

3.2.4 J-Link Connection Options

Use the following options to select a J-Link device to connect to and use for any operation that requires a connection to a kit or debugger. You can connect over IP (using the `–-ip` option) or over USB (using the `--serialno` option) as shown in the following examples. You can use only one of these options at a time. If no option is provided, Simplicity Commander attempts a connection to the only USB connected J-Link adapter.

Command Line Usage

```
$ commander <command> –-serialno <J-Link serial number>
```

Command Line Input Example

```
$ commander adapter probe –-serialno 440050184
```

Command Line Usage

```
$ commander <command> –-ip <IP address>
```

Command Line Input Example

```
$ commander adapter probe –-ip 10.7.1.27
```

3.2.5 Graphical User Interface

Displays a Graphical User Interface (GUI) for laboratory use of Simplicity Commander. The GUI can be used in the lab for such typical tasks as:

- Flashing device images
- Upgrading Silicon Labs kit firmware and configuration
- Setting device lock features

Command Line Usage

```
$ commander
```

4 EFR32 Custom Tokens

4.1 Introduction

Simplicity Commander supports defining custom token groups for reading and writing. Custom tokens work just like manufacturing tokens, but the definition and location of the tokens is configurable to suit different requirements.

Any custom token definition files must be placed in a specific `tokens` folder for Simplicity Commander to find and parse it. The location of this file is slightly different depending on the operating system used:

On Windows and Linux, the `tokens` folder is included in the zip file and is placed alongside the executable in the installation directory.

On Mac OS X, the folder named `~/Library/SimplicityCommander/tokens/` is generated automatically when running `commander` on the command line for the first time. Running `commander --help`, for example, is enough to ensure that the folder with files is created.

Inside this `tokens` folder, there is a file named `tokens-example-efr32.json`. This file provides an example of the token types and locations currently supported by Simplicity Commander.

4.2 Creating Custom Token Groups

To define a custom token group, copy `tokens-example-efr32.json` to a new file in the same directory using the naming convention `tokens-<groupname>-efr32.json.`

For example: `tokens-myapp-efr32.json`

To verify that Simplicity Commander sees the new file, run

```
commander.exe tokendump –-help
```

The name of your token group (for example, "myapp") should be listed as a supported token group like this:

`--tokengroup <tokengroup>` which set of tokens to use. Supported: myapp, znet

4.3 Defining Tokens

Each token in the JSON file has the following properties:

Property	Description
Name	The name of the token, which is used as an identifier when dumping or writing tokens.
Page	The named memory region to use for the token. See **Memory Regions**.
Offset	The offset in number of bytes from the start of the memory region at which to place the token.
sizeB	The size of the token in bytes. ▪ A token of size 1 is interpreted as an unsigned 8-bit integer. ▪ A token of size 2 is interpreted as an unsigned 16-bit integer. ▪ A token of size 4 is interpreted as an unsigned 32-bit integer. ▪ Any other size is interpreted as a byte array of the given size.
Description	A plain text description of the token. This property is currently only used for documentation of the JSON file.

4.4 Memory Regions

The following values are valid data in the "page" option:

USERDATA

The data in the user data page is **not** erased via a mass erase (`commander.exe flash --masserase`, `commander.exe masserase`, or when disabling debug lock). It can, however, be erased by a specific page erase (located at address 0x0FE00000 with size 2 kB on EFR32 devices).

LOCKBITSDATA

The lock bits page is used by the chip itself to configure flash write locks, debug lock, AAP lock, and so on. However, the last 1.5 kB of this page is unused by the device itself, and has the important property that it is erased in a mass erase event

The lock bits page is located at address 0x0FE04000 with size 2 kB on EFR32 devices. Tokens in this page must use an offset of at least 0x200; otherwise, collisions with chip functionality can occur.

4.5 Using Custom Token Files

To use a custom token file, run Simplicity Commander with a `--tokengroup` option corresponding to the name of the JSON file. For example, if the file was named `tokens-myapp-efr32.json`, use this option:

```
--tokengroup myapp
```

To create a text file useful as input to the `flash` or `convert` commands, the easiest way is to start by dumping the current data from a device.

For example:

```
commander tokendump -s 440050148 --tokengroup myapp --outfile mytokens.txt
```

`mytokens.txt` can then modified to have the desired content, and then used when flashing devices or creating images in this way:

```
commander flash -s 440050148 --tokengroup myapp --tokenfile mytokens.txt
```

To be able to read the custom token data from an application, Simplicity Commander provides the `tokenheader` command, which generates a C header file that can be included in an application. See Section 5.4.4, **Generate C Header Files from Token Groups** for details.

5 Simplicity Commander Commands

This section includes the following information for using each Simplicity Commander command:

- Command Line Usage
- Command Line Input Example
- Command Line Output Example

In cases where the Command Line Usage is the same as the Command Line Input Example, only the former is included.

The Simplicity Commander commands are organized in the following categories:

- Device Flashing Commands
- Flash Verification Command
- Memory Read Commands
- Token Commands
- Convert and Modify File Commands
- Dump File Contents Command
- EBL Commands
- Kit Utility Commands
- Device Erase Commands
- Device Lock and Protection Commands
- Device Utility Commands
- External SPI Flash Commands

5.1 Device Flashing Commands

The commands in this section all require a working debug connection for communicating with the device. You would normally always use one of the J-Link connection options when running the `flash` command, but it is intentionally left out of most of the examples to keep them short and concise.

5.1.1 Flash Image File

Flashes the image in the specified filename to the target device, starting at the specified address. The affected bytes will be erased before writing. If the image contains any partial flash pages, these pages will be read from the device and patched with the image contents before erasing the page and writing back. After writing, the affected flash areas are read back and compared. Finally, the chip is reset using a pin reset, making code execution start. The debugger to connect to is indicated by the J-Link serial number (`--serialno` option).

Command Line Usage

```
$ commander flash <filename> --address <address> --serialno <serial number>
```

Command Line Input Example

```
$ commander flash blink.bin --address 0x0 --serialno 440012345
```

Connects to the J-Link debugger with serial number 440012345 and flashes the image in blink.bin to the target device, starting at address 0.

Command Line Output Example

```
Flashing blink.s37.
Flashing 2812 bytes, starting at address 0x00000000
Resetting...
Uploading flash loader...
Waiting for flashloader to become ready...
Erasing flash...
Flashing...
Verifying written data...
Resetting...
Finished!
DONE
```

5.1.2 Flash Using IP Address without Verification and Reset

Flashes the image in the specified filename to the target device, using the IP address specified. The data in flash is not verified after flashing, and the device is left halted after flashing.

Command Line Usage

```
$ commander flash <filename> --ip <IP> --halt --noverify>
```

Command Line Input Example

```
$ commander flash blink.s37 --ip 10.7.1.27 --halt --noverify
```

Flashes the image in blink.s37 to the target device, using the IP address 10.7.1.27. The data in flash is not verified after flashing, and the device is left halted after flashing.

Command Line Output Example

```
Flashing blink.s37.
Flashing 2812 bytes, starting at address 0x00000000
Resetting...
Uploading flash loader...
Waiting for flashloader to become ready...
Erasing flash...
Flashing...
Finished!
DONE
```

5.1.3 Flash Several Files

Flashes the images to the target device. Any overlapping data is considered an error.

Command Line Usage

```
$ commander flash <filename> <filename>
```

Command Line Input Example

```
$ commander flash blink.s37 userpage.hex
```

Flashes the images in blink.s37 and userpage.hex to the target device.

Command Line Output Example

```
Adding file blink.s37...
Adding file userpage.hex...
Flashing 2812 bytes, starting at address 0x00000000
Resetting...
Uploading flash loader...
Waiting for flashloader to become ready...
Erasing flash...
Flashing...
Verifying written data...
Finished!
Flashing 2048 bytes, starting at address 0x0fe00000
Resetting...
Uploading flash loader...
Waiting for flashloader to become ready...
Erasing flash...
Flashing...
Verifying written data...
Resetting...
Finished!
DONE
```

5.1.4 Patch Flash

Writes the specified byte(s) to the flash. The affected pages will be read from the device and patched with this data before erasing the page and writing back. When you use the `--patch` option, the patch memory data is interpreted as an unsigned integer. The optional `length` argument can be used to define the number of bytes, up to 8 bytes. If no length is specified, the default is to patch 1 byte.

Command Line Usage

```
$ commander flash --patch <address>:<data>[:length]
```

Command Line Input Example

```
$ commander flash --patch 0x120:0xAB --patch 0x3200:0xA5A5:2
```

Writes the specified bytes 0xAB to address 0x120 and 0xA5A5 to address 0x3200. The affected pages will be read from the device and patched with this data before erasing the page and writing back.

Command Line Output Example

```
Patching 0x00000120 = 0xAB...
Patching 0x00003200 = 0xA5A5...
Flashing 2048 bytes, starting at address 0x00000000
Resetting...
Uploading flash loader...
Waiting for flashloader to become ready...
Erasing flash...
Flashing...
Verifying written data...
Finished!
Flashing 2048 bytes, starting at address 0x00003000
Resetting...
Uploading flash loader...
Waiting for flashloader to become ready...
Erasing flash...
Flashing...
Verifying written data...
Resetting...
Finished!
DONE
```

5.1.5 Patch Using Input File

Flashes the specified application while simultaneously patching the image file and the flash of the device. If a filename is inside the file, these bytes are patched before writing the image

Command Line Usage

```
$ commander flash <filename> –-patch <address>:<data>[:length] –-patch <address>:<data>[:length]
```

Command Line Input Example

```
$ commander flash blink.s37 --patch 0x123:0x00FF0001:4 --patch 0x0FE00004:0x00
```

Flashes the blink application while simultaneously patching the image file and the flash of the device. Because `0x123` is inside the file, these bytes are patched before writing the image. Additionally, the user page will be read from the device and patched with this data before erasing the page and writing back.

Command Line Output Example

```
Flashing blink.s37.
Patching 0x00000123 = 00FF0001...
Patching 0x0FE00004 = 00...
Flashing 4096 bytes, starting at address 0x00000000
Resetting...
Uploading flash loader...
Waiting for flashloader to become ready...
Erasing flash...
Flashing...
Verifying written data...
Finished!
Flashing 2048 bytes, starting at address 0x0fe00000
Resetting...
Uploading flash loader...
Waiting for flashloader to become ready...
Erasing flash...
Flashing...
Verifying written data...
Finished!
DONE
```

5.1.6 Flash Tokens

This section describes how to flash one or more tokens from text file(s) and/or command line options with their new values. Manufacturing tokens are the only token type supported by Simplicity Commander; simulated EEPROM tokens are not supported. For more information on manufacturing tokens, see *AN961, Bringing Up Custom Nodes for the Mighty Gecko and Flex Gecko Families*.

The `--tokengroup` option defines which group of tokens is used. Simplicity Commander currently has built-in support for the `znet` token group.

Silicon Labs recommends generating a token file from a device or image file using the `tokendump` command and then making modifications to this file for use with the `--tokenfile` option.

Command Line Usage

```
$ commander flash --tokengroup <token group> –-token <TOKEN_NAME:value> –-tokenfile <filename>
```

Command Line Input Example

```
$ commander flash --tokengroup znet --token TOKEN_MFG_STRING:”IoT Inc”
```

Set the token `MFG_STRING` to have the value `IoT Inc`. The `TOKEN_` prefix is optional, that is, `TOKEN_MFG_STRING` and `MFG_STRING` are equivalent.

Command Line Input Example

```
$ commander flash --tokengroup znet --tokenfile tokens.txt
```

Sets the tokens specified in tokens.txt. All tokens in the file are processed, and if a duplicate is found, it will be treated as an error.

Command Line Input Example

```
$ commander flash --tokengroup znet --tokenfile tokens.txt --token TOKEN_MFG_STRING:”IoT Inc”
```

Sets the tokens specified in tokens.txt. Additionally, sets the `MFG_STRING` to the value given. All files and tokens specified on the command line are processed, and if a duplicate is found, it will be treated as an error.

Depending on the operating system and shell being used, some escapes may be needed to correctly specify a string. For example, on the command line in a Windows 7 Professional Command Prompt window, execute the following command:

```
commander flash --tokengroup znet --token "TOKEN_MFG_STRING:\"IoT Inc\""
```

Command Line Output Example

```
Flashing 2048 bytes to 0x0fe00000
Resetting...
Uploading flash loader...
Waiting for flashloader to become ready...
Erasing flash...
Flashing...
Verifying written data...
Resetting...
Finished!
DONE
```

5.2 Flash Verification Command

The `verify` command verifies the contents of a device against a set of files, tokens, and/or patch options without writing anything to the flash. It works just like the verification step of the `flash` command, but without actually flashing first. For example, the `verify` command can be used to verify that the application on a microcontroller is what you expect it to be.

Command Line Usage

All options and examples for the `flash` command also apply to the `verify` command, except for the `--halt`, `--masserase`, and `--noverify` options that do not apply to the `verify` command.

```
$ commander verify [filename] [filename ...] [patch options] [token options]
```

Command Line Input Example

```
$ commander verify myimage.hex
```

Command Line Output Example

```
Parsing file myimage.hex...
Verifying 52000 bytes at address 0x00000000...OK!
Verifying 2048 bytes at address 0x0fe00000...OK!
DONE
```

5.3 Memory Read Commands

The `readmem` command reads data from a device and can either store it to file or print it in human-readable format. The location and length to be read from the device is defined by the `--range` and `--region` options. You can combine one or more ranges and regions to read and combine several different areas in flash to one file.

Note: Like `flash`, the commands in this section all require a working debug connection for communicating with the device. One would normally always use one of the J-Link connection options when running `readmem`, but this is left out of the examples to keep them short and concise.

The `--range` option supports two different range formats:

- The first is `<startaddress>:<endaddress>`, for example, `--range 0x4000:0x6000`. The range is non-inclusive, meaning that all bytes from 0x4000 up to and including 0x5FFF are read out.
- The second is `<startaddress>:+<length>`, which takes an address to start reading from, and a number of bytes to read. For example, the equivalent command line input to the previous example is `--range 0x4000:+0x2000`.

The `--region` option takes a named flash region with an @ prefix. Valid regions for use with the `--region` option are listed below.

EFM32, EZR32, EFR32: `@mainflash`, `@userdata`, `@lockbits`, `@devinfo`

5.3.1 Print Flash Contents

Specifies the range of memory to read from flash and prints data.

Command Line Usage

```
$ commander readmem –-range <startaddress>:<endaddress>
```

OR

Command Line Usage

```
$ commander readmem –-range <startaddress>:+<length>
```

Command Line Input Example

```
$ commander readmem --range 0x100:+128
```

Reads 128 bytes from flash starting at address 0x100 and prints it to standard out.

Command Line Output Example

```
Reading 128 bytes from 0x00000100...
{address:  0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F}
00000100: 12 F0 40 72 11 00 DF F8 C0 24 90 42 07 D2 DF F8
00000110: BC 24 90 42 03 D3 5F F0 80 72 11 00 01 E0 00 22
00000120: 11 00 DF F8 84 26 12 68 32 F0 40 72 0A 43 DF F8
00000130: 78 36 1A 60 70 47 80 B5 00 F0 90 FC FF F7 DD FF
00000140: 01 BD DF F8 70 16 09 68 08 00 70 47 38 B5 DF F8
00000150: 4C 06 00 F0 9F F9 05 00 ED B2 28 00 07 28 05 D0
00000160: 08 28 07 D1 00 F0 7C FC 04 00 0B E0 FF F7 E9 FF
00000170: 04 00 07 E0 40 F2 25 11 DF F8 3C 06 00 F0 B0 FC
DONE
```

5.3.2 Dump Flash Contents to File

Reads the contents of the specified user page and stores it in the specified filename. File format will be auto-detected based on file extension (.bin, .hex, or .s37).

Command Line Usage

```
$ commander readmem –-region <@region> --outfile <filename>
```

Command Line Input Example

```
$ commander readmem --region @userdata --outfile userpage.hex
```

Reads the contents of the region named userdata and stores it in an output file named userpage.hex.

Command Line Output Example

```
Reading 2048 bytes from 0x0fe00000...
Writing to userpage.hex...
DONE
```

5.4 Token Commands

The `tokendump` command generates a text dump of token data. It can take as input either a (set of) files using the same command line options as the `convert` command, or a microcontroller using the same command line options as the `readmem` command.

The output of `tokendump` can either be printed to standard output or written to an output file using the `--outfile` option. The file written when using the `--outfile` option is suitable for modification and re-use as input to the `flash`, `verify`, or `convert` commands using the `--tokenfile` option.

`tokendump` always requires a token group to be selected with the `--tokengroup` option. A token group is a defined set of tokens for a specific stack or application. Simplicity Commander only supports the `znet` token group.

Manufacturing tokens are the only token type supported by Simplicity Commander; simulated EEPROM tokens are not supported. For more information on manufacturing tokens, see *AN961, Bringing Up Custom Nodes for the Mighty Gecko and Flex Gecko Families*.

5.4.1 Print Tokens

Command Line Usage

```
$ commander tokendump --tokengroup <token group> [--token <token name>]
```

Command Line Input Example

```
$ commander tokendump --tokengroup znet --token TOKEN_MFG_STRING --token TOKEN_MFG_EMBER_EUI_64
```

Reads the selected tokens from the device and prints it to stdout.

Command Line Output Example

```
#
# The token data can be in one of three main forms: byte-array, integer, or string.
# Byte-arrays are a series of hexadecimal numbers of the required length.
# Integers are BIG endian hexadecimal numbers.
# String data is a quoted set of ASCII characters.
#
MFG_STRING      : "IoT_Inc"
# MFG_EMBER_EUI_64: F0B2030000570B00

DONE
```

5.4.2 Dump Tokens to File

This example works just like section 4.3.1, Print Tokens, except that the output is written to a file suitable for use with the `--tokenfile` option (`flash`, `verify`, and `convert` commands).

Command Line Usage

```
$ commander tokendump --tokengroup <token group> [--token <token name>] --outfile <filename>
```

Command Line Input Example

```
$ commander tokendump --tokengroup znet --outfile tokens.txt
```

Reads all tokens from the device and outputs it to the file named tokens.txt.

Command Line Output Example

```
Writing tokens to tokens.txt...
DONE
```

5.4.3 Dump Tokens from Image File

If an input file is given to the `tokendump` command, the input is read from one or more files instead of reading from a device.

In this case, the `--device` option must be provided, because token locations can be different from one device family to another.

Command Line Usage

```
$ commander tokendump <filename> --tokengroup <token group> --device <device> [--outfile <filename>]
```

Command Line Input Example

```
$ commander tokendump blink.hex --tokengroup znet --device EFR32MG1P --outfile tokens.txt
```

Command Line Output Example

```
Parsing file blink.hex...
DONE
```

5.4.4 Generate C Header Files from Token Groups

The `tokenheader` command generates a simple header file based on a custom token group. The generated header file contains pre-processor defines that specify the location and size of each token.

See 4, **EFR32 Custom Tokens** for details on custom tokens.

Command Line Usage

```
$ commander tokenheader --tokengroup <group name> --device <target device> <filename>
```

Command Line Input Example

```
$ commander tokenheader --tokengroup myapp --device EFR32MG1P233F256 my_tokens.h
```

Command Line Output Example

```
Writing token header file: my_tokens.h
DONE
```

5.5 Convert and Modify File Commands

The `convert` command performs image file conversion and manipulation. It supports the following actions:

- Conversion between file formats
- Merging several image files
- Extracting subsets of images
- Patching bytes
- Setting token data

The `convert` command can either write its output to a file or print it to standard out in human-readable format, similar to the `readmem` command. When writing to a file, the file format is auto-detected based on the file extension used.

The `convert` command works off-line without any J-Link/debug connection. The command is device-agnostic, except when working with tokens or ebl files. In this case, you must use the `--device` option.

Command Line Usage

```
$ commander convert [infile1] [infile2 …] [options]
```

5.5.1 Combine Two Files

Converts two files with different file formats into one specified output file.

Command Line Usage

```
$ commander convert <filename> <filename> [--address <address>] --outfile <filename>
```

Command Line Input Example

```
$ commander convert blink.bin userpage.hex --address 0x0 --outfile blinkapp.s37
```

Combines blink.bin and userpage.hex to blinkapp.s37. The address option is used to set the start address of the .bin file, since bin files doesn't contain any addressing information. If more than one .bin file is supplied, the same start address is used for all. If this is not desirable, consider converting the bin files to s37 or hex in a separate preparation step.

Command Line Output Example

```
Parsing file blink.bin...
Parsing file userpage.hex...
Writing to blinkapp.s37...
DONE
```

5.5.2 Define Specific Bytes

Like the `flash` command, the `convert` command supports the `--patch` option for setting arbitrary unsigned integers at any address.

Command Line Usage

```
$ commander convert [filename] --patch <address>:<data>[:length] [--outfile <filename>]
```

Command Line Input Example

```
$ commander convert blink.s37 --patch 0x0FE00000:0x12345:4 --outfile blink.hex
```

Converts blink.s37 to hex format, while simultaneously defining the first four bytes of the user page to 0x00012345. This works just like `flash blink.s37 --patch 0x0FE00000:0x12345:4`, but works against a file instead of writing to a device flash.

Command Line Output Example

```
Parsing file blink.s37...
Patching 0x0FE00000 = 0x00012345...
Writing to blink.hex...
DONE
```

5.5.3 Define Tokens

Like the `flash` command, the `convert` command supports the `--tokengroup`, `--token` and `--tokenfile` options for setting token data while doing file conversion.

Command Line Usage

```
$ commander convert [filename] --tokengroup <token group> [--tokenfile <filename>] [--token <token
name>:<token data>] [--device <device>] [--outfile <filename>]
```

Command Line Input Example

```
$ commander convert blink.s37 --tokengroup znet --tokenfile tokens.txt --device EFR32MG1P --outfile
blink.hex
```

Converts blink.s37 to hex format, while simultaneously defining the tokens defined in tokens.txt and on the command line. Works just like the corresponding options with `flash`, but writes to file instead of flash.

Command Line Output Example

```
Parsing file blink.s37...
Writing to blink.hex...
DONE
```

5.5.4 Dump File Contents

Like the `readmem` command, the `convert` command will print its output in human-readable format to standard out if no output file is given.

Command Line Usage

```
$ commander convert <filename> [--address <bin file start address>]
```

Command Line Input Example

```
$ commander convert blink.bin --address 0x0 userpage.hex
```

If the `--outfile` option is not used, the data is printed to stdout instead of writing to file.

Command Line Output Example

```
Parsing file blink.bin...
Parsing file userpage.hex...
{address:  0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F}
00000000: 10 04 00 20 B5 0A 00 00 57 08 00 00 8B 0A 00 00
00000010: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000020: 00 00 00 00 00 00 00 00 00 00 00 00 97 0A 00 00
00000030: 00 00 00 00 00 00 00 00 D1 0A 00 00 13 06 00 00
00000040: D3 0A 00 00 D5 0A 00 00 D7 0A 00 00 D9 0A 00 00
00000050: DB 0A 00 00 DD 0A 00 00 DF 0A 00 00 E1 0A 00 00
00000060: E3 0A 00 00 E5 0A 00 00 E7 0A 00 00 E9 0A 00 00
00000070: EB 0A 00 00 ED 0A 00 00 EF 0A 00 00 F1 0A 00 00
<shortened data for documentation>
00000ac0: C5 0A 00 00 C0 46 C0 46 C0 46 C0 46 FF F7 CA FF
00000ad0: FE E7 FE E7 FE E7 FE E7 FE E7 FE E7 FE E7 FE E7
00000ae0: FE E7 FE E7 FE E7 FE E7 FE E7 FE E7 FE E7 FE E7
00000af0: FE E7 FE E7 00 36 6E 01 00 80 00 00
{address:  0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F}
0fe00000: 45 23 01 00 FF FF FF FF FF FF FF FF FF FF FF FF
0fe00010: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0fe00020: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
<shortened data for documentation>
0fe007e0: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0fe007f0: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
DONE
```

5.6 EBL Commands

5.6.1 Print EBL Information

Parses and prints EBL information from the specified ebl file.

Command Line Usage

```
$ commander ebl print <filename>
```

Command Line Input Example

```
$ commander ebl print nodetest.ebl
```

Command Line Output Example

```
Found EBL Tag = 0x0000, length  140, [EBL Header]
  Version:     0x0201
  Signature:   0xE350 (Correct)
  Flash Addr:  0x00004000
  AAT CRC:     0x53BC1F4E
  AAT Size:    128 bytes
    HalAppBaseAddressTableType
      Top of Stack:       0x20006980
      Reset Vector:       0x000121F9
      Hard Fault Handler: 0x00012125
      Type:               0x0AA7
      HalVectorTable:     0x00004100
    Full AAT Size:    172
    Ember Version:    5.7.0.0
    Ember Build:      0
    Timestamp:        0x561E581F (Wed Oct 14, 2015 13:26:55 UTC [+0100])
    Image Info String:''
    Image CRC:        0x2ACE0C5B
    Customer Version: 0x00000000
    Image Stamp:      0xF4271F50BA2E2FBA

Found EBL Tag = 0xFD03, length 1924, [Erase then Program Data]
  Flash Addr: 0x00004080
Found EBL Tag = 0xFD03, length 2052, [Erase then Program Data]
  Flash Addr: 0x00004800

(32 additional tags of the same type and length.)

Found EBL Tag = 0xFD03, length 1772, [Erase then Program Data]
  Flash Addr: 0x00015000
Found EBL Tag = 0xFC04, length    4, [EBL End Tag]
  CRC: 0xDBC82DA5

The CRC of this EBL file is valid (0xdebb20e3)

File has 0 bytes of end padding.
Calculated image stamp matches value found in AAT.
DONE
```

5.6.2 EBL Key Generation

Generates a keyfile to be used for encryption or decryption and outputs the keyfile to the specified filename.

Command Line Usage

```
$ commander ebl keygen --keyfile <filename>
```

Command Line Input Example

```
$ commander ebl keygen --keyfile key.txt
```

Command Line Output Example

```
Using /dev/random for random number generation
Gathering sufficient entropy... (may take up to a minute)...
DONE
```

5.6.3 EBL File Encryption

Encrypts an EBL file using a keyfile generated by `ebl keygen` and writes the output to the specified filename.

Command Line Usage

```
$ commander ebl encrypt <filename> --keyfile <filename> --outfile <filename>
```

Command Line Input Example

```
$ commander ebl encrypt nodetest.ebl --keyfile key.txt --outfile nodetest.ebl.encrypted
```

Command Line Output Example

```
Unencrypted  input file:  nodetest.ebl
Encrypt output file:      nodetest.ebl.encrypted
Randomly generating nonce
Using /dev/random for random number generation
Gathering sufficient entropy... (may take up to a minute)...
Created ENCRYPTED ebl image file
DONE
```

5.6.4 EBL File Decryption

Decrypts an encrypted EBL file and writes the output to the specified filename. The keyfile must be the same as was used for encrypting the encrypted EBL file.

Command Line Usage

```
$ commander ebl decrypt <filename> --keyfile <filename> --outfile <filename>
```

Command Line Input Example

```
$ commander ebl decrypt nodetest.ebl.encrypted --keyfile key.txt --outfile nodetest.ebl
```

Command Line Output Example

```
Unencrypted output file:  nodetest.ebl
Encrypt  input file:      nodetest.ebl.encrypted
MAC matches. Decryption successful.
Created DECRYPTED ebl image file
DONE
```

5.7 Kit Utility Commands

5.7.1 Firmware Upgrade

Updates the application running on the board controller on the kit to a new version provided in an .emz file by Silicon Labs.

Command Line Usage

```
$ commander adapter fwupgrade --serialno <J-Link serial number> <filename>
```

Command Line Input Example

```
$ commander adapter fwupgrade -s 440050184 S1015B_wireless_stk_firmware_package_0v14p0b435.emz
```

Command Line Usage Output

```
Checking manifest...
Checking if target is in bootloader...
Waiting for kit to restart...
Package is usable
Deleting previous firmware...
Installing files...
Resetting target...
Waiting for kit to restart...
Finished!
DONE
```

5.7.2 Kit Information Probe

Retrieves information about a connected kit. Lists information about the kit part number and name, connected boards, and firmware version.

The options `--kit`, `--boards`, and `--firmware` limit the output to just kit information, board list, or firmware information, respectively.

Command Line Usage

```
$ commander adapter probe --serialno <J-Link serial number> [--kit] [--boards] [--firmware]
```

Command Line Input Example

```
$ commander adapter probe --serialno 440050184
```

Command Line Usage Output

```
Kit Information:
=====================================
Kit Name        : EFR32 Mighty Gecko 2400/915 MHz Dual Band Wireless Starter Kit
Kit Part Number : WSTK6002A Rev. A00
J-Link Serial   : 440050184
Debug Mode      : MCU

Firmware Information:
=====================================
FW Version      : 0v14p0b435

Board List:
=====================================
Name            : Wireless Starter Kit Mainboard
Part Number     : BRD4001A Rev. A01
Serial Number   : 152607557

Name            : EFR32MG 2400/915 MHz 19.5 dBm Dual Band Radio Board
Part Number     : BRD4150B Rev. B00
Serial Number   : 151300035
DONE
```

5.7.3 Adapter Reset Command

This command resets the adapter itself, causing a restart. The `adapter reset` command is usually not required during normal operation.

An error about “Communication timed out” may occur because the adapter sometimes restarts before it has time to reply to the command.

Command Line Usage

```
$ commander adapter reset
```

Command Line Input Example

```
$ commander adapter reset
```

Command Line Output Example

```
Communication timed out: Requested 76 bytes, received 0 bytes !
DONE
```

5.8 Adapter Debug Mode Command

This command sets or reads the current debug mode of the adapter. The supported debug modes are typically IN, OUT, MCU, and OFF. See the quick start guide for your kit for a description of the debug modes it supports.

Command Line Usage

```
$ commander adapter dbgmode [mode]
```

Command Line Input Example

```
$ commander adapter dbgmode MCU
```

Command Line Output Example

```
Setting debug mode to MCU...
DONE
```

5.9 Device Erase Commands

5.9.1 Erase Chip

Executes a mass erase for devices where it is supported. On EFM32G and EFM32TG, all pages are erased instead, which is significantly slower.

Command Line Usage

```
$ commander device masserase
```

Command Line Usage Output

```
Erasing chip...
DONE
```

5.9.2 Erase Region

Erases a named region. For more information on the `–-region` option, see section 4.2, Device Memory Commands.

Command Line Usage

```
$ commander device pageerase –-region <@region>
```

Command Line Input Example

```
$ commander device pageerase --region @userdata
```

Command Line Output Example

```
Erasing range 0x0fe00000 - 0x0fe00800
DONE
```

5.9.3 Erase Pages in Address Range

Erases all flash pages affected by the given memory range. If the given range doesn't match page boundaries, it will be extended to always erase entire pages.

Command Line Usage

```
$ commander device pageerase –-range <startaddress>:<endaddress>
```

Command Line Input Example

```
$ commander device pageerase –-range 0x200:0x6000
```

Erases all flash pages 0 to 11 or 0x0000 to 0x5FFF (assuming a page size of 2 kB).

Command Line Output Example

```
Erasing range 0x00000000 - 0x00006000
DONE
```

5.10 Device Lock and Protection Commands

5.10.1 Debug Lock

Locks access to the debug interface of the device.

Command Line Usage

```
$ commander device lock –-debug enable
```

Command Line Usage Output

```
Locking debug access...
DONE
```

5.10.2 Debug Unlock

Unlocks access to the debug interface of the device. This triggers a mass erase if the device was locked before.

Command Line Usage

```
$ commander device lock –-debug disable
```

Command Line Usage Output

```
ERROR: Could not get MCU information
Removing all locks/protection...
Unlocking debug access (triggers a mass erase)...
DONE
```

5.10.3 Write Protect Flash Ranges

Protects all flash pages affected by the given memory range from any writes or erases. If the given range doesn't match page boundaries, it will be extended to always protect entire pages.

Command Line Usage

```
$ commander device protect --write --range <startaddress>:<endaddress>
```

Command Line Input Example

```
$ commander device protect --write --range 0x0:0x4000
```

Protects all flash pages in the first 16 kB from being erased or written to. Useful for protecting a bootloader from being modified by buggy application code, for example.

Command Line Output Example

```
Write protecting range 0x00000000 - 0x00004000
DONE
```

5.10.4 Write Protect Flash Region

Protects all flash pages in the named region from being written to or erased.

Command Line Usage

```
$ commander device protect --write --region @<region>
```

Command Line Input Example

```
$ commander device protect --write --region @mainflash
```

Protects the entire main flash from being written to or erased.

Command Line Output Example

```
Write-protecting all pages in main flash.
DONE
```

5.10.5 Disable Write Protection

Disables write protection for all pages.

Command Line Usage

```
$ commander device protect --write --disable
```

Command Line Output Example

```
Disabling all write protection...
DONE
```

5.11 Device Utility Commands

5.11.1 Device Information Command

Shows detailed information about the target device.

Command Line Usage

```
$ commander device info
```

Command Line Usage Output

```
Part Number    : EFR32MG1P233F256GM48
Die Revision   : A0
Production Ver : 0
Flash Size     : 256 kB
SRAM Size      : 32 kB
Unique ID      : 000b57000003b2f0
DONE
```

5.11.2 Device Reset Command

Resets a device using a pin reset.

Command Line Usage

```
$ commander device reset
```

Command Line Usage Output

```
Resetting chip...
DONE
```

5.11.3 Device Recovery Command

Tries to recover a device that has lost debug access due to misconfiguration of clocks, GPIO pins, or similar. Recovery is not supported on all devices, and in some cases requires the kit corresponding to the device you want to recover, for example, an EFM32TG STK to recover an EFM32TG device.

Command Line Usage

```
$ commander device recover
```

Command Line Usage Output

```
Recovering "bricked" device...
DONE
```

5.12 External SPI Flash Commands

Simplicity Commander supports reading, writing, and erasing data on an external SPI flash on a limited selection of boards and devices. The following configurations are currently supported:

- The integrated SPI flash on EFR32MG1x632 and EFR32MG1x732 devices
- The MX25 SPI flash on EFR32 radio boards

5.12.1 Erase External SPI Flash Command

Use this command to erase data on an external flash. By default, the erased range is read back to verify that it was actually erased. This blank check can be disabled by including the `--noverify option`.

The `extflash erase` command always erases complete sectors. Any sector overlapping with the range provided will be erased. All currently supported flash devices have a sector size of 4096 bytes. For example, erasing with option `--range 0xE00:0x1100` will effectively erase the first two sectors (equivalent to `--range 0x0:0x2000`).

Command Line Usage

```
$ commander extflash erase --range <range expression> [--noverify]
```

Command Line Input Example

```
$ commander extflash erase --range 0x1000:0x3000
```

Command Line Output Example

```
Erasing 8192 bytes from 0x00001000 on external flash.
Resetting target...
Uploading flashloader...
Erasing external flash...
Verifying written data...
Waiting for flashloader to become ready...
Reading from external flash...
DONE
```

5.12.2 Read External SPI Flash Command

Use this command to read from external flash.

Command Line Usage

```
$ commander extflash read --range <range expression>
```

Command Line Input Example

```
$ commander extflash read --range 0x0:+0x20
```

Command Line Output Example

```
Reading 32 bytes from 0x00002000 on external flash.
Resetting target...
Uploading flashloader...
Waiting for flashloader to become ready...
Reading from external flash...
{address:  0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F}
00002000: 48 65 6C 6C 6F 20 57 6F 72 6C 64 21 0A FF FF FF
00002010: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
DONE
```

5.12.3 Write External SPI Flash Command

Use this command to write to external flash.

Any existing content in the affected flash sectors will be erased before writing.

In contrast to the `flash` command for internal flash, the `extflash write` command always flashes the raw content of the given file. If, for example, an S-record file is provided, the ASCII content of the file is written; the S-record format is not parsed and written to the addresses specified in the file.

Command Line Usage

```
$ commander extflash write <filename> --address <start address>
```

Command Line Input Example

```
$ commander extflash write myfile.txt --address 0x2000
```

Command Line Output Example

```
Flashing 13 bytes to 0x00002000 on external flash.
Resetting target...
Uploading flashloader...
Waiting for flashloader to become ready...
Erasing external flash...
Writing to external flash...
Verifying written data...
Waiting for flashloader to become ready...
Reading from external flash...
DONE
```

6 Revision History

The following subsections summarize the new features of Simplicity Commander by version number.

6.1 Version 0.15

2016-04-27

- Added commands:
  - `extflash`
  - `adapter reset`
  - `adapter dbgmode`

6.2 Version 0.14

2016-02-05

- Added commands:
  - `device lock`
  - `device protect`
  - `device pageerase`
  - `device recover`

6.3 Version 0.13

Not released

- Added `tokenheader` command.

6.4 Version 0.12

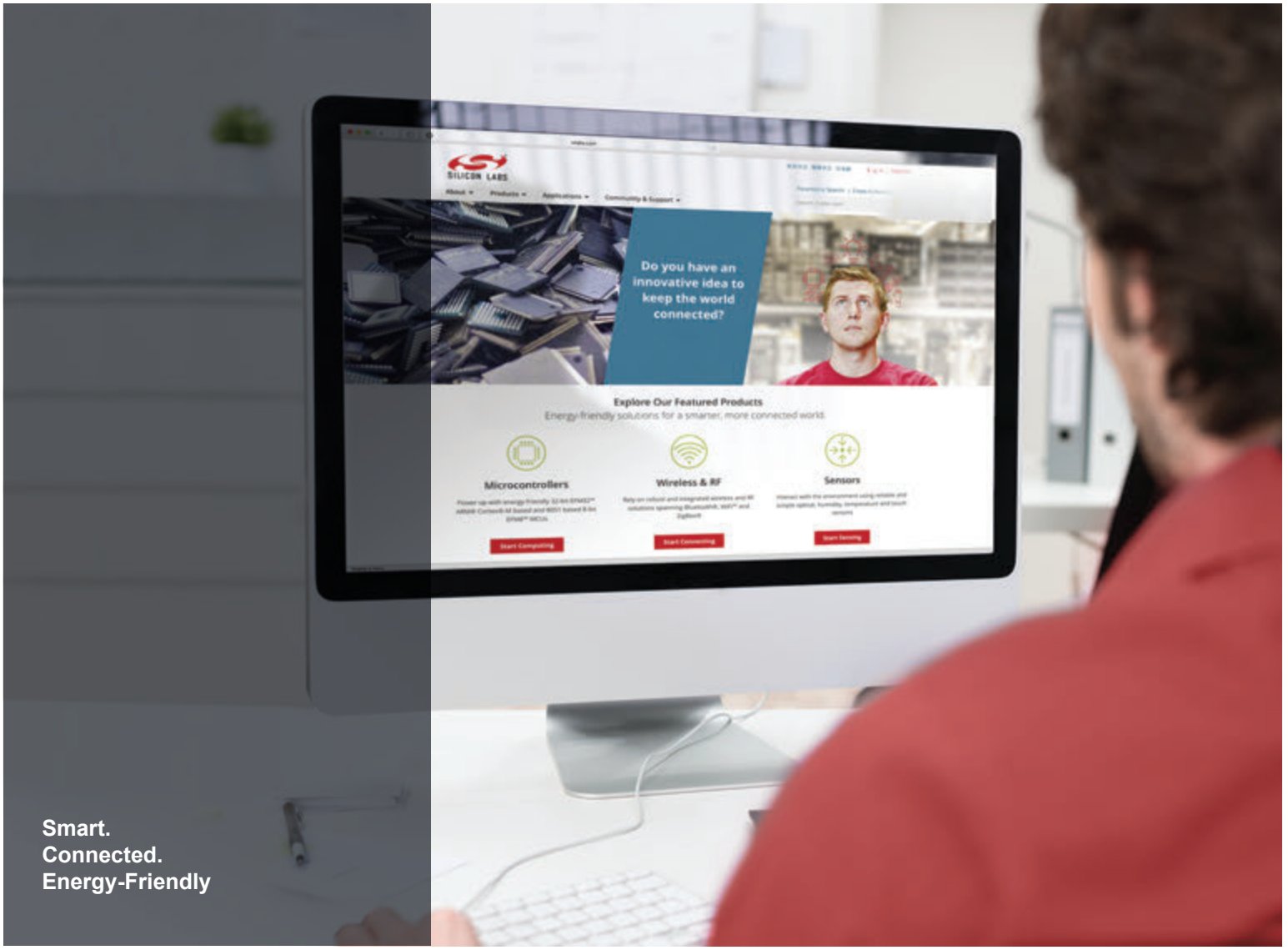
2016-01-20

- Added support for EFR32 custom tokens.

6.5 Version 0.11

2016-01-15

Initial release.



Products
www.silabs.com/products

Quality
www.silabs.com/quality

Support and Community
community.silabs.com

SILICON LABS®

Silicon Laboratories Inc.
400 West Cesar Chavez
Austin, TX 78701
USA

http://www.silabs.com