# Simplicity Commander


- Introduction
- File Format Overview
- General Information
- EFR32 Custom Tokens
- Security Overview
- Simplicity Commander Commands
  - Introduction
  - Configure Commands
  - Device Flashing Commands
  - Flash Verification Command
  - Memory Read Commands
  - Tokens Command
  - Convert and Modify File Commands
  - EBL Commands
  - GBL3 Commands
  - GBL4 Commands
  - Kit Utility Commands
  - Device Erase Commands
  - Device Lock and Protection Commands
  - Device Utility Commands
  - External SPI Flash Commands
  - Advanced Energy Monitor Commands
  - Serial Wire Output Read Commands
  - NVM3 Commands
  - CTUNE Commands
  - Security Commands
  - Util Commands
  - OTA Commands
  - Post-Build Command
  - RPS Commands
  - VUART Commands
  - RTT Commands
  - Serial Commands
  - Manufacturing Commands


Copyright © 2026 Silicon Laboratories. All rights reserved.

- VCOM Commands
- Completion Commands
- LittleFS Commands
- Batch Operations
  - Introduction
  - Configuring Devices
  - Defining Actions
  - Applying Debug Connection Options
  - Executing a Batch Job
  - Importing and Exporting a Batch Job


Copyright © 2026 Silicon Laboratories. All rights reserved.

# Introduction

NOTE: This site replaces *UG162: Simplicity Commander Reference Guide*. Further updates to this user guide will be provided here.

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.
- Communicate with the target device

Simplicity Commander is designed to support the Silicon Labs Wireless STK and STK platforms.

The primary intended audience for this document is software engineers, hardware engineers, and release engineers who are familiar with programming EFM32, EFR32, EM3xx, and SiWx91x devices. EFM8 MCU families are not supported at this time. This reference guide describes how to use the Simplicity Commander CLI. It provides general information on file formats supported by Simplicity Commander and the Silicon Labs bootloaders, 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. This guide is up-to-date with Simplicity Commander version 1.23.0.

Copyright © 2026 Silicon Laboratories. All rights reserved.

File Format Overview

# File Format Overview

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

## 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. For more information on Motorola S-record file format, see https://en.wikipedia.org/wiki/SREC_(file_format). 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 supported target device using the Simplicity Commander `flash` command. The s37 format can represent any combination of any byte of flash in the device. The Simplicity Commander `convert` command can also be used to read multiple s37 files and hex files; output an s37 file for combining multiple files into a single file; and modify individual bytes of a file.

## Update Image File Formats

An update image file provides an efficient and fault-tolerant image format for use with Silicon Labs bootloaders to update an application without the need for special programming devices. Two image formats are supported: Gecko Bootloader (GBL) format for use with the Silicon Labs Gecko Bootloader introduced for use with EFR32 devices and Ember Bootloader (EBL) format for use with legacy Ember bootloaders. See *UG103.6: Application Development Fundamentals: Bootloading* for more details about these image file formats and bootloader use with different platforms.

Update image files are generated by the Simplicity Commander `gbl create` or `ebl create` command. These formats can only represent firmware images; they cannot be used to capture Simulated EEPROM token data (as described by *AN703: Using Simulated EEPROM Version 1 and Version 2 for the EM35x and EFR32 Series 1 SoC Platforms*). GBL upgrade files may contain data that gets flashed outside the main flash.

Bootloaders can receive an update image file either over-the-air (OTA) or via a supported peripheral interface, such as a serial port, and reprogram the flash in place. Update image files are generally used in later stage development and for upgrading manufactured devices in the field.

During development, bootloaders should be loaded onto the device using the .s37 or .hex file format. If the Gecko Bootloader with support for in-field bootloader upgrades is used, it is possible to perform a bootloader upgrade using a GBL update image. For other bootloaders or file formats, do not attempt to load a bootloader image onto the device as an update image.

## 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,

Copyright © 2026 Silicon Laboratories. All rights reserved.

complete image often derived from multiple s37 files. You can use the Simplicity Commander `convert` command to read multiple hex files 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.

The following table summarizes the inputs and outputs for the different file formats used by Simplicity Commander.

| | Inputs | | | | | Outputs | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | ebl | s37 | hex | bin | chip | ebl | s37 | hex | bin | chip | rps |
| flash | | X | X | X | | | | | | X | |
| readmem | | | | | X | | X | X | X | | |
| convert | | X | X | X | | | X | X | X | | |
| ebl create | | X | X | X | | X | | | | | |
| ebl parse | X | | | | | | X | X | X | | |
| rps create | | X | X | X | | | | | | | X |
| mfg917 | | X | X | X | | | X | X | X | | |

Copyright © 2026 Silicon Laboratories. All rights reserved.

General Information

# General Information

## Installing Simplicity Commander

You can install Simplicity Commander using Simplicity Studio or by downloading one of the following standalone versions and then completing the installation:

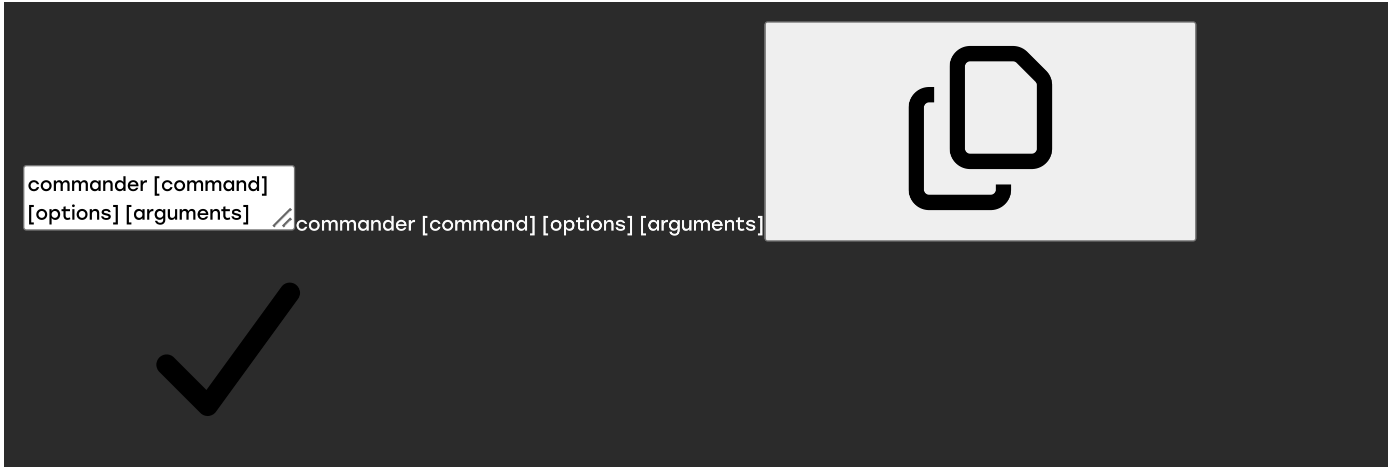
https://www.silabs.com/documents/public/software/SimplicityCommander-Linux.zip

https://www.silabs.com/documents/public/software/SimplicityCommander-Mac.zip

https://www.silabs.com/documents/public/software/SimplicityCommander-Windows.zip

## 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:




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 with `--` (double dash) as described 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>`

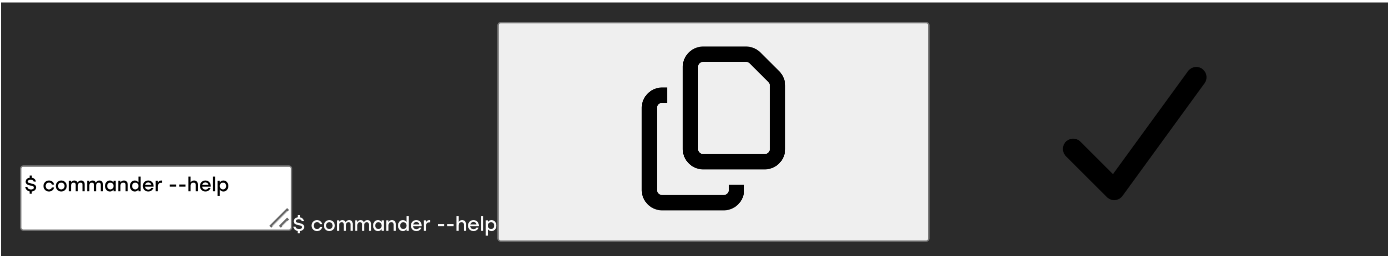
## General Options

### Help ( `--help` )

Copyright © 2026 Silicon Laboratories. All rights reserved.

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

Command Line Syntax




Command Line Usage Output

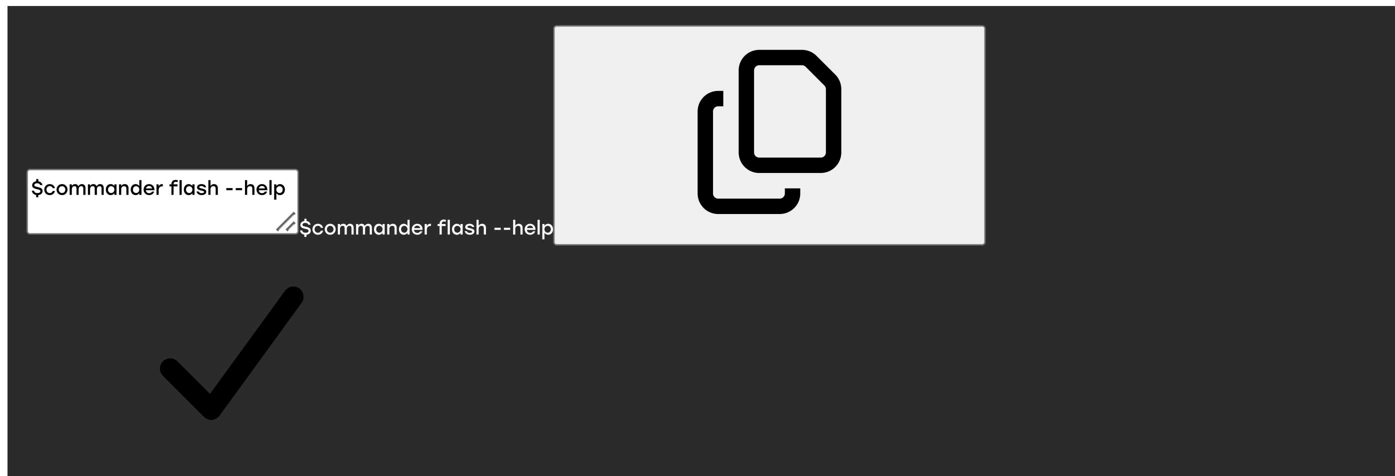
Simplicity Commander help displays a list of all Simplicity Commander commands. The following figure is an example.

Command Prompt

```
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.
  aem             AEM (Advanced Energy Monitor) 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.
  gbl             Encrypt, decrypt and other handling for GBL files.
  readmem         Read memory from a device.
  swo             SWO commands.
  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
```

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

Command Line Input Example




Command Line Output Example

Simplicity Commander displays help for the flash command in the following figure.

Command Prompt

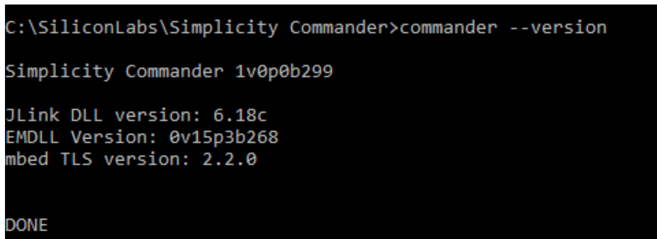
```
C:\SiliconLabs\Simplicity Commander>commander flash --help

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.
  --irpre <IR length>                  JTAG: Total length of instruction
                                       registers of all devices closer to TDI
                                       than the addressed ARM device.
  --drpre <Data bits>                  JTAG: Total number of data bits closer
                                       to TDI than the addressed ARM device.
  --address <address>                  Address to flash to. Not applicable for
                                       hex or s37 files which contain address
                                       information.
  --halt                               Leave the target halted after flashing.
                                       By default the device is reset by a pin
                                       reset after flashing.
  --masserase                          Supply this to do a mass erase of the
                                       entire main flash 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
```

## Version ( `--version` )

Copyright © 2026 Silicon Laboratories. All rights reserved.

Command Line Usage Output

Simplicity Commander displays version information. The following figure is an example.



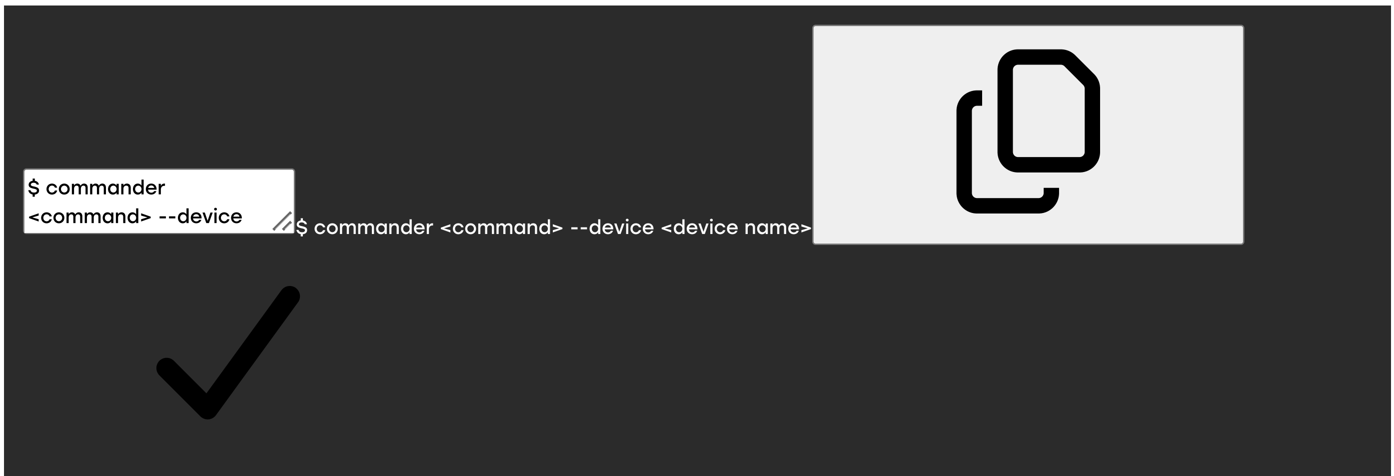

## Device ( `--device <device name>` )

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.

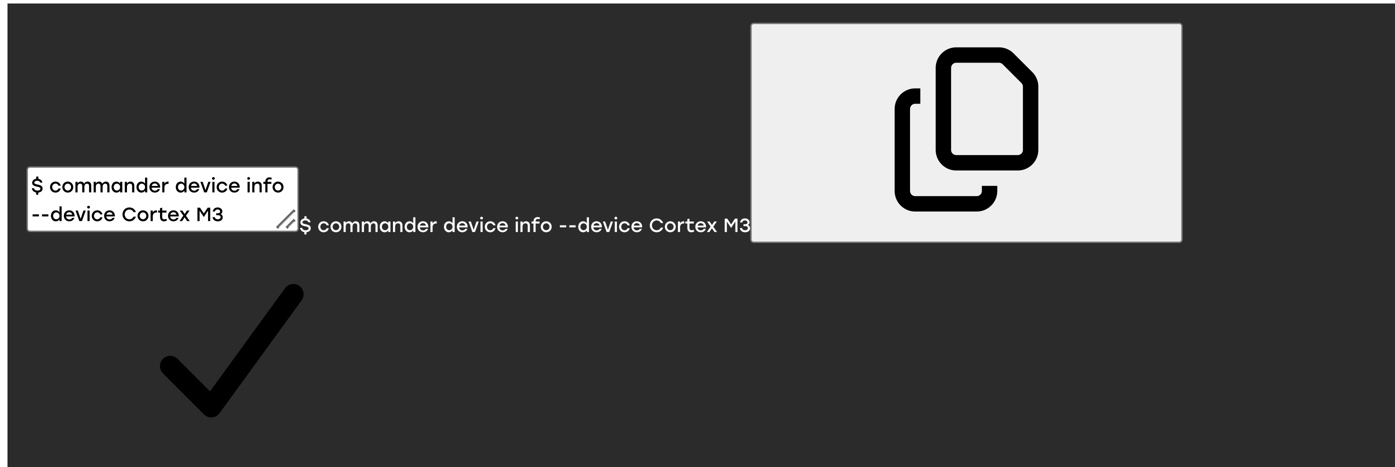
Using a complete part number such as `--device EFR32MG1P233F256GM48` is always supported and recommended.

Command Line Syntax




Command Line Input Example

Copyright © 2026 Silicon Laboratories. All rights reserved.




## 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), over USB (using the `--serialno` option), or you can provide the serial port name or device file (using the `--identifybyserialport` 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.

> Note: Providing the `--identifybyserialport` option only lets Simplicity Commander use the serial port name to *identify* the corresponding J-Link device; Simplicity Commander will still connect to the J-Link device over USB (similarly to when you provide the `--serialno` option).

Command Line Syntax

Command Line Input Example

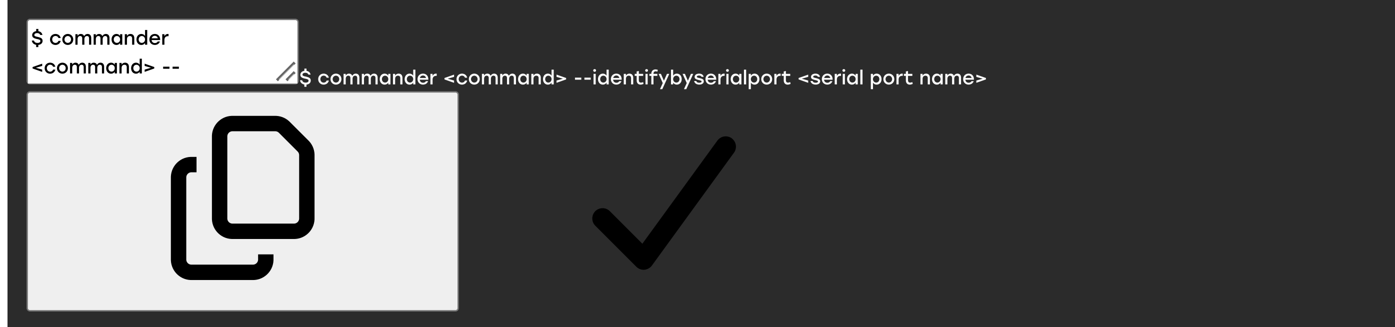
Copyright © 2026 Silicon Laboratories. All rights reserved.

Command Line Syntax

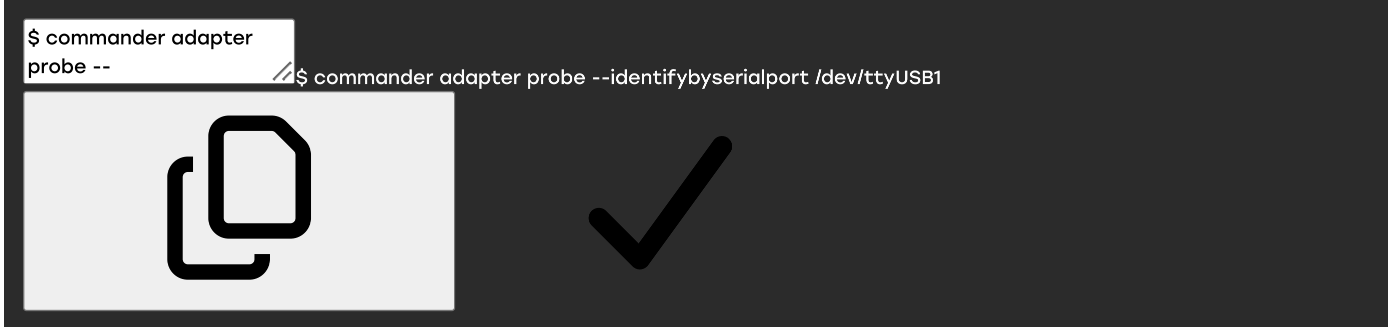
Command Line Input Example

Command Line Syntax

Copyright © 2026 Silicon Laboratories. All rights reserved.




Command Line Input Example




## Debug Interface Configuration

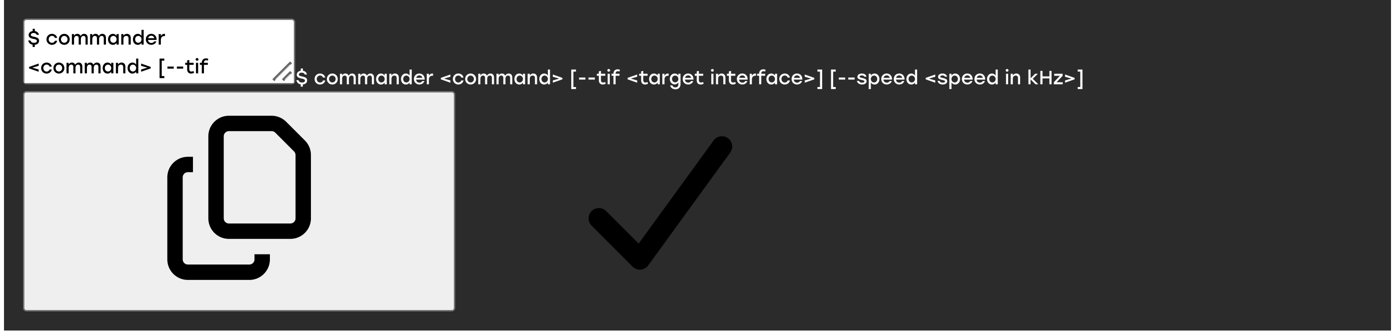
Use the `--tif` and `--speed` options to configure the target interface and clock speed when connecting the debugger to the target device.

Simplicity Commander supports using Serial Wire Debug (SWD) or Joint Test Action Group (JTAG) as the target interface. All currently supported Silicon Labs hardware works with SWD, while some can also be used with JTAG. Custom hardware may require JTAG to be used.

The maximum clock speed available typically depends on the debug adapter, the target device, and the physical connection between the two. Silicon Labs kits typically support speeds up to 1000 – 8000 kHz, depending on the kit model. If the selected clock speed is higher than what the adapter supports, the clock speed will fall back to using the highest speed it does support. You may want to select a lower clock speed if the debug connection is unstable or not working at all when working with custom hardware with longer debug cables or when the electrical connections are less than ideal.

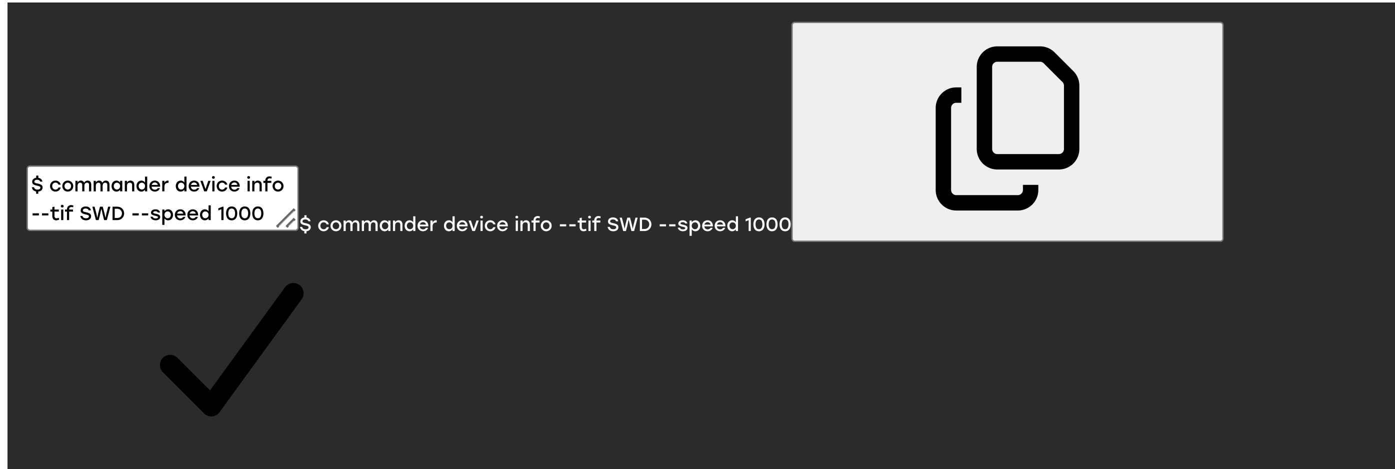
If the `--tif` and `--speed` options are not used, the default configuration is SWD and 4000 kHz.

Command Line Syntax



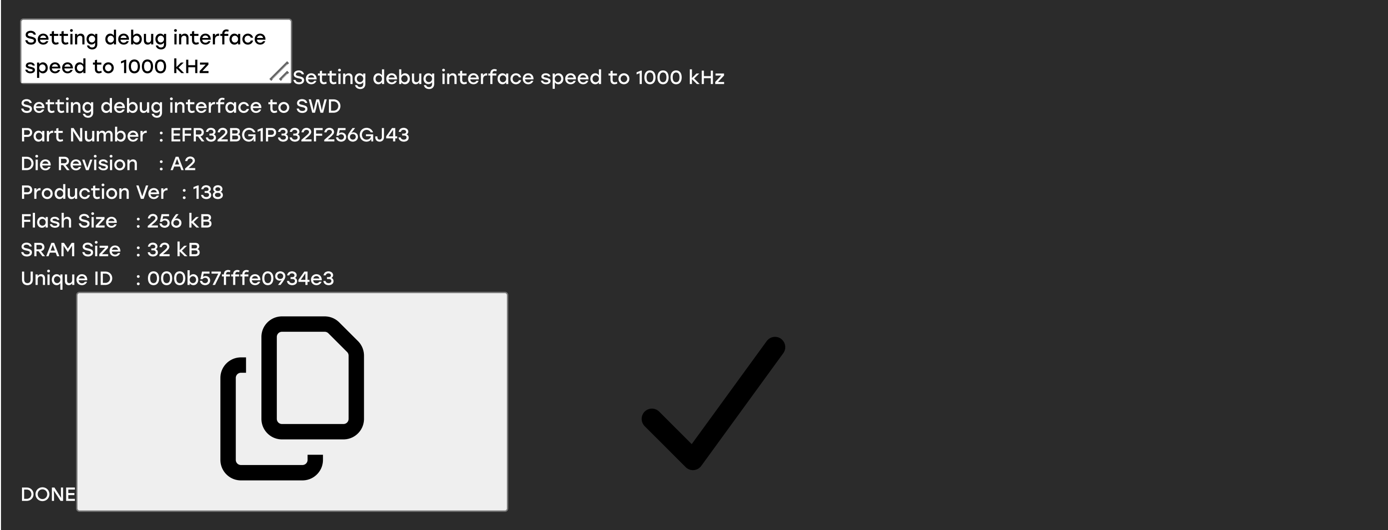

Command Line Input Example

Copyright © 2026 Silicon Laboratories. All rights reserved.



```
$ commander device info --tif SWD --speed 1000
```

Command Line Output Example



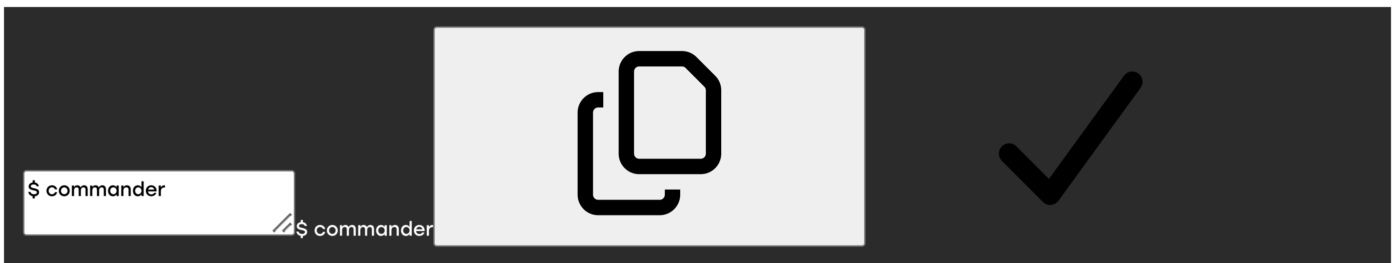
```
Setting debug interface speed to 1000 kHz
Setting debug interface to SWD
Part Number  : EFR32BG1P332F256GJ43
Die Revision    : A2
Production Ver  : 138
Flash Size   : 256 kB
SRAM Size   : 32 kB
Unique ID    : 000b57fffe0934e3
DONE
```

## 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
- Accessing the kit's Admin console
- Communicating with the target device via multiple protocols, including:
  - SEGGER Real Time Transfer (RTT)
  - Serial Wire Output (SWO)
  - Virtual UART (VUART)
  - Virtual COM (VCOM)

Command Line Syntax



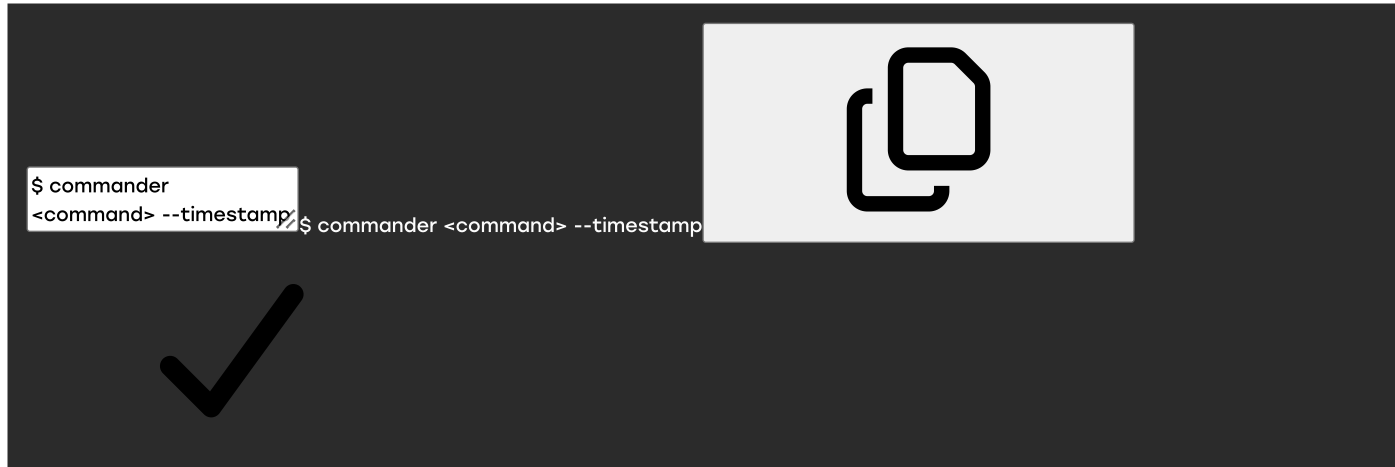
```
$ commander
```

Copyright © 2026 Silicon Laboratories. All rights reserved.

## Timestamp ( `--timestamp` )

Add a timestamp to the Simplicity Commander output.

Command Line Syntax



```
$ commander <command> --timestamp
```

Command Line Usage Output

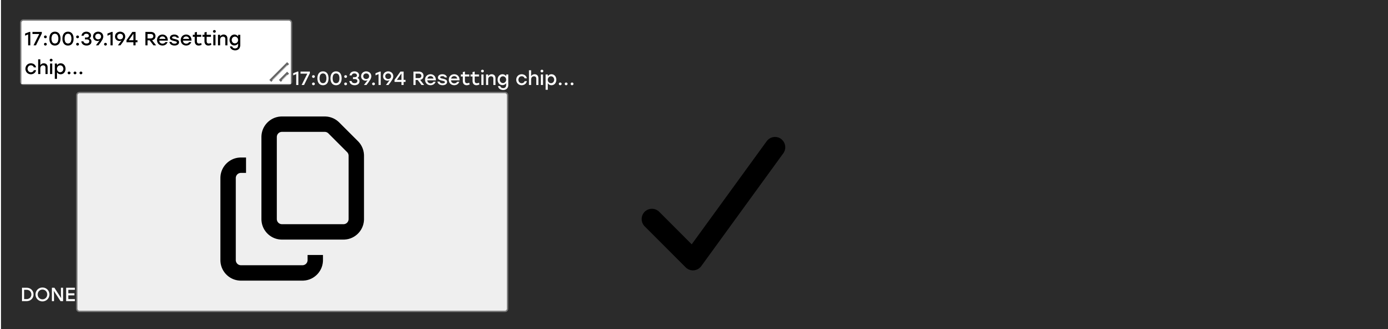
Display a timestamp to all output from Simplicity Commander.

Command Line Input Example

```
$commander device reset --timestamp
```

Command Line Output Example

Simplicity Commander displays the timestamp for the device reset command.



```
17:00:39.194 Resetting chip...
DONE
```

# Output and Exit Status

Copyright © 2026 Silicon Laboratories. All rights reserved.

The exit status of Simplicity Commander can take on a few different values. Whenever an operation completed successfully, Simplicity Commander's exit status is 0 (zero). Any error will cause the exit status to be non-zero.

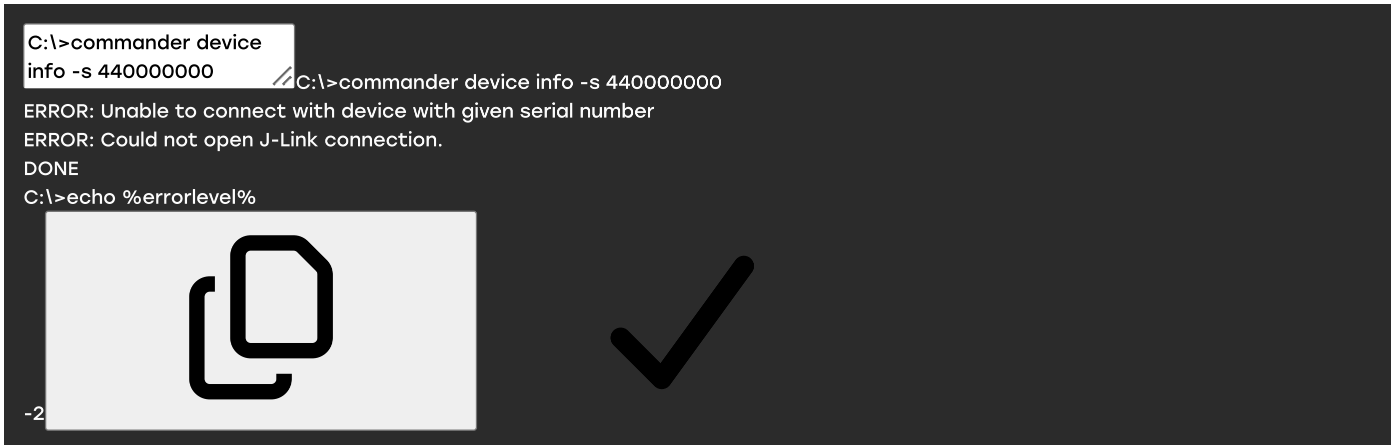
Simplicity Commander defines the following exit status codes.Exit StatusDescription0No error occured-1Input error. For example, this could be a missing command line option, non-existent command, or an invalid filename.-2Run time error. Used whenever anything goes wrong when executing the command. Examples include not being able to connect to a debug adapter or flash verification failed.Note: Some operations systems present the exit status as an unsigned integer. On these systems, -1 will be interpreted as 255, -2 as 254, and so on.

The operating system itself may create other exit codes if the application crashes. These will always be non-zero and are out of the control of Simplicity Commander.

All errors and potential error conditions are indicated in Simplicity Commander's output in addition to the exit status. All errors are displayed with the prefix "ERROR:". All warnings are displayed with the prefix "WARNING:".

Any output from Simplicity Commander will always end with "DONE". This does not indicate that the operation was successful, merely that execution has finished.

Example of an error in Windows follows.



```
C:\>commander device info -s 440000000
ERROR: Unable to connect with device with given serial number
ERROR: Could not open J-Link connection.
DONE
C:\>echo %errorlevel%
-2
```

Copyright © 2026 Silicon Laboratories. All rights reserved.

EFR32 Custom Tokens

# Custom Tokens

## 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.

There are two different ways for Simplicity Commander to find and use custom token definition files. For Simplicity Commander to treat the custom token file in the same way as a regular token group, the file must be placed in a specific location as described in Custom Token Groups below.

The other option is to use the `--tokendefs` command line option instead of the `--tokengroup` option. With this method, Simplicity Commander uses a token definition file in an arbitrary location, for example, under revision control. For more information, see Using Custom Token Files in Any Location.

## Custom Token Groups

For Simplicity Commander to treat custom token files like regular token groups, the file must be placed in a specific `tokens` folder and the filename must follow a special syntax.

The location and initialization of the tokens folder depends 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 in the user's home directory 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.

The syntax of the filename is `tokens-<group name>-<architecture>.json`. `<group name>` is the name of the custom token group and can be any string. `<architecture>` is a string describing which devices the token definitions apply to. The following table lists the supported architecture strings.

| Architecture | Devices |
| --- | --- |
| `efr32` | All Series 1 EFR32 devices |
| `efr32xg2` | All Series 2 EFR32 devices |
| `em3xx` | All EM3xx devices |
| `efm32` | All EFM32 devices (Series 0 and 1) |
| `ezr32` | All EZR32 devices |
| `sixx3` | All Sixx3 devices (Series 3) |

For example, to define the token group myapp for EFR32 Series 1 devices, the filename would be `tokens-myapp-efr32.json`.

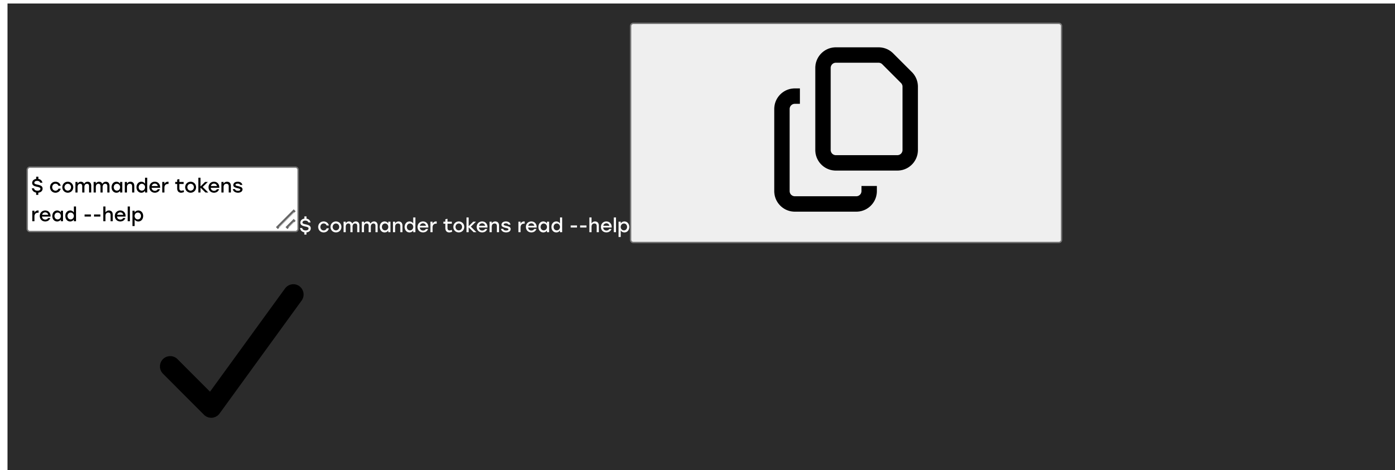
## Creating Custom Token Groups

Copyright © 2026 Silicon Laboratories. All rights reserved.

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

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

To verify that Simplicity Commander sees the new file, run:




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

# 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. |
| sizeB | The size of the token in bytes.<br>• A token of size 1 is interpreted as an unsigned 8-bit integer.<br>• A token of size 2 is interpreted as an unsigned 16-bit integer.<br>• A token of size 4 is interpreted as an unsigned 32-bit integer.<br>• Any other size is interpreted as a byte array of the given size. |
| string | Optional boolean. If this property is `true`, the token is interpreted as a zero terminated ASCII<br>string instead of a byte array. The maximum string length is `sizeB - 1` because one byte is reserved for the zero terminator. |
| description | A plain text description of the token. This property is currently only used for documentation of the JSON file. |

## Additional properties for manufacturing tokens

| Property | Description |
| --- | --- |
| page | The named memory region to use for the token. For more information, see Memory Regions below. |
| offset | The offset in number of bytes from the start of the memory region at which to place the token. |

## Additional properties for static tokens

| Property | Description |
| --- | --- |

Copyright © 2026 Silicon Laboratories. All rights reserved.

| Property | Description |
| --- | --- |
| kind | The kind of the token. This determines how the token is stored and accessed. Valid kinds are: `STATIC_SECURE` and `STATIC_DEVICE`. If this property is omitted from the json file, it defaults to `STATIC_SECURE`. |
| key | A hexadecimal number that identifies the token. We recommend using a key larger than 0x600 to avoid collisions with any Silicon Labs predefined tokens. |

## Memory Regions

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

USERDATA

The user data page is a separate flash page intended for persistent data and configuration. The user data page is not erased when disabling debug lock. It can, however, be erased by a specific page erase.

The user data page is located at address 0x0FE00000. It is 2 kB on Series 1 EFR32 devices and 1 kB on Series 2 EFR32 devices.

LOCKBITSDATA

On Series 1 EFR32 devices, 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 when disabling debug lock. A regular mass erase by the MSC—typically by executing the `commander device masserase` or `commander flash --masserase` command—does not erase the lock bits page.

The lock bits page is located at address 0x0FE04000 with size 2 kB on Series 1 EFR32 devices. Tokens in this page must use an offset of at least 0x200 on these devices; otherwise, collisions with chip functionality can occur.

On Series 2 EFR32 devices, there is no physical lock bits page. Instead, the LOCKBITSPAGE region is defined to be the first 2 kB of the last flash page in the main flash block. This maintains backwards compatibility, while still ensuring that any data in this region is erased when the device is erased during debug unlock.

## Token Kinds

For Sixx3 devices, there is no concept of memory regions. Instead, the tokens are defined with a key and a token kind. The key is a hexadecimal number that identifies the token, and the token kind is a string that describes the type of the token. The following token kinds are supported:

- `STATIC_SECURE`: This is the main KLV chain where almost all tokens should be placed. These tokens are stored in a dedicated flash range which is defined in the linker file. This memory range must be provided when reading/writing these tokens. Static secure tokens are encrypted by default.
- `STATIC_DEVICE`: This KLV chain is placed in SE-managed OTP. It has the benefit of being persistent through any kind of flash erase. However, the storage space and the number of writes/erases are very limited (252 B for the entire chain and 100 writes respectively). We therefore strongly recommend using static secure tokens instead.

## Token File Format Description

A token file declares what values are programmed for manufacturing tokens on the chip. Lines are composed of one of the following forms:

Copyright © 2026 Silicon Laboratories. All rights reserved.

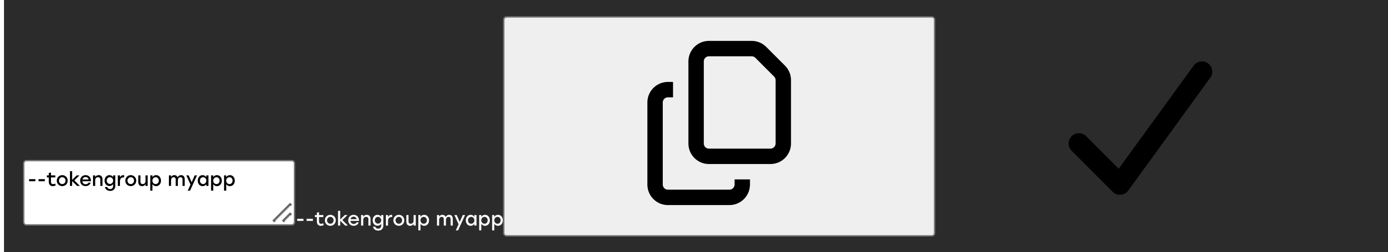
```
<token-name> : <data>
<token-name> : !ERASE!
```

Follow these guidelines when using a token file:

- Omitted tokens are left untouched and not programmed on the chip.
- Token names are case insensitive.
- All integer values are interpreted as hexadecimal numbers in BIG-endian format and must be prefixed with '0x'.
- Blank lines and lines beginning with # (hashtag) are ignored.
- Byte arrays are given in hexadecimal format without a leading '0x'.
- Specifying !ERASE! for the data sets that token to all 0xFF.
- 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 that must be prefixed with '0x'.
- String data is a quoted set of ASCII characters.

## Using Custom Token Files

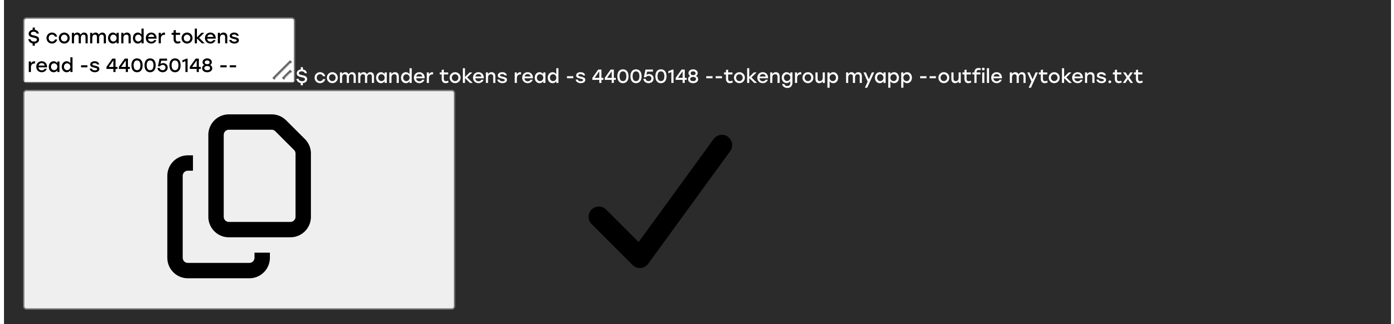
Refer to Introduction for a definition of custom token files and where they should be located for Simplicity Commander to find them automatically. To use a custom token file located in the `tokens` folder, run Simplicity Commander with a `--tokengroup` option corresponding to the name of the JSON file. 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 `tokens write` or `convert` commands, the easiest way is to start by dumping the current data from a device. For static tokens, adding the `--includeall` option to the `tokens read` command includes all static tokens defined in the file in the output, even if they are not present on the device.

For example:



```
$ commander tokens read -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:

Copyright © 2026 Silicon Laboratories. All rights reserved.

This command reads all static tokens defined in the `myapp` token group from the device, including those not present on the device, and writes them to `mytokens.txt`.

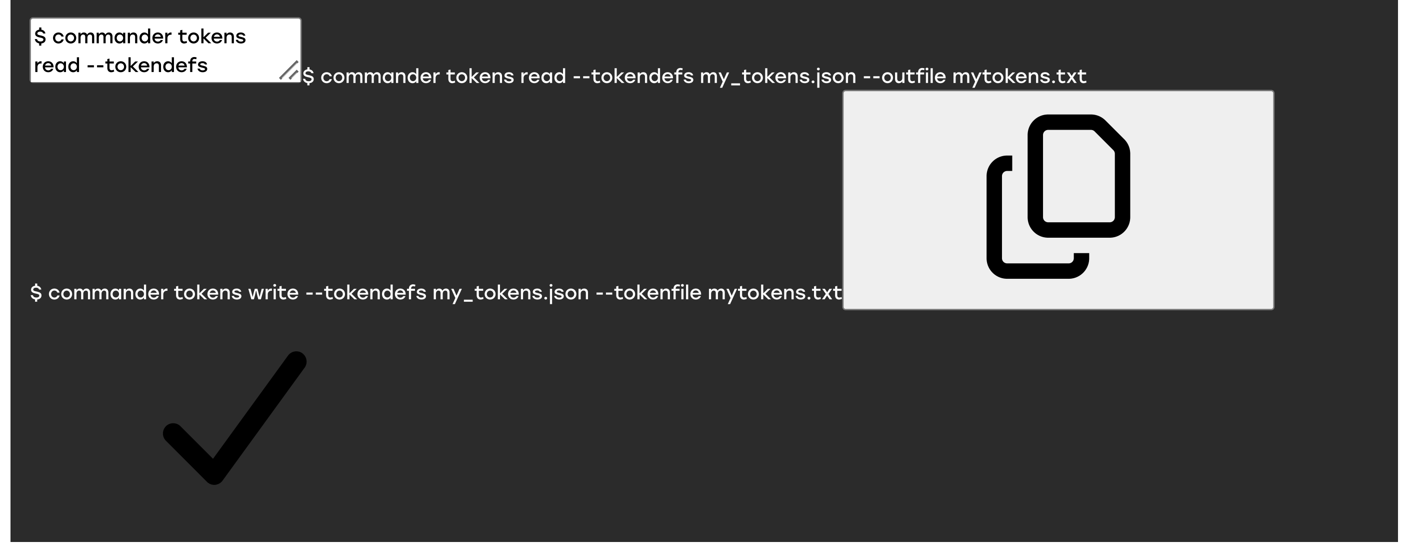
To be able to read the custom token data from an application, Simplicity Commander provides the `tokenheader` command, which generates a C header file that can be included in an application. See Generate C Header Files from Token Groups for details.

## Using Custom Token Files in Any Location

In some cases, it is more convenient to have the custom token definitions file somewhere in the file system (for example, if it is placed under revision control). Simplicity Commander supports this functionality with the `--tokendefs` option which refers to a JSON file anywhere in the file system. Use it instead of the `--tokengroup` option.

For example:

Copyright © 2026 Silicon Laboratories. All rights reserved.




Copyright © 2026 Silicon Laboratories. All rights reserved.

# Security Overview

This chapter describes essential security features in Simplicity Commander.

## Security Store

Security Store is the location where all files generated and used by the security commands in Simplicity Commander are stored. You can find the path to Security Store with the `commander security getpath` command. Unless the `--nostore` option is used with security commands, Simplicity Commander will store all keys, certificates, and configuration files seen in Security Store. Descriptions of the files appear below.

- access_certificate.bin: certificate delegating permission to unlock debug access of a device.
- archive folder: folder used to store all outdated files (for example, all files in the challenge folder are moved here when a challenge is rolled).
- cert_key.pem: private key used to sign unlock token.
- cert_pubkey.pem: public key used in certificate. Public key corresponding to cert_key.pem.
- certificate_authorization.json: configuration file used to define authorizations given by access certificate. May be edited.
- challenge_xxx folder: folder used to store files related to a challenge.
    - unlock_payload_xxx.bin: payload used to unlock secure debug access.
    - unlock_command_to_be_signed_dd_mm_yyyy.bin: command token that needs to be signed with cert_key.pem
- command_key.pem: private command key used to sign access certificate.
- command_pubkey.pem: public command key stored on device. Public key corresponding to command_key.pem.
- user_configuration.json: configuration file used in write config. May be edited.

When running the `commander security unlock` command, Simplicity Commander will use all available files to attempt to unlock the debug access. If anything is missing, you will be asked to provide the file as an option to the command. The file will then be stored in Security Store, unless the `--nostore` option is used.

## Access Certificate

An access certificate is used to delegate access to a single device to another key, which is called a certificate key. This scheme supports security models where the command key is kept in a secure location, while the certificate key can be used with more lenient security practices.

The access certificate contains the serial number of the device it applies to, a description of what actions it gives access to, and the public certificate key. An outline of the access certificate is illustrated in the following figure.

The device serial number uniquely identifies each device. It can be displayed by executing the `commander security status` command. The certificate_authorizations.json file sets the authorizations for the certificate. The current version of Simplicity Commander does not support any modifications to the authorization file, but it will be available in future versions. The private certificate key corresponding to the public certificate key in the certificate is used to generate a signature required to unlock debug access. For more information, see Challenge and Command Signing. The certificate is authenticated by signing it with the private command key corresponding to the public command key written to the device. The signing of the certificate may be done by passing an unsigned certificate to a Hardware Security Module (HSM) containing the private key or by providing the private key to Simplicity Commander (that is, for development) using the `--command-key` option.

Copyright © 2026 Silicon Laboratories. All rights reserved.

Device Serial Number

Authorization

Certificate Public Key

Access Certificate Signature
Signed by Command private key

## Challenge and Command Signing

The part of the data that needs to be signed to create a valid unlock command is called the *challenge*. Secure Engine generates this random data. It remains unchanged until it is updated to a new random value by the security rollchallenge command.

By updating the challenge, any existing command signatures are effectively invalidated because part of the data the signature encompasses has changed. This allows the owner of the device to give debug access to someone else for a limited amount of time.

A command signature is created by signing a binary containing the data fields in yellow in the following figure; Simplicity Commander sets the unlock command ID, command parameters, and the security challenge using the private key corresponding to the public key in the access certificate.

The security gencommand command creates a file containing these elements, but does not include the signature. If the certificate private key is not available to the user, the signature must be obtained from another party–for example, an HSM. If the user possesses the certificate private key, Security Commander can create the signed unlock command using the security unlock command. By passing the command signature and the access certificate to the Debug Challenge interface, the debug interface is temporarily unlocked until the next power-on or pin reset.

Unlock Command ID

Command parameters

Device challenge

Unlock Command Signature
Signed by Certificate Private key

Copyright © 2026 Silicon Laboratories. All rights reserved.

# Simplicity Commander Commands

# Simplicity Commander Commands

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

- Command Line Syntax
- Command Line Input Example
- Command Line Output Example

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

The Simplicity Commander commands are organized in the following categories:

- Configure Commands
- Device Flashing Commands
- Flash Verification Command
- Memory Read Commands
- Tokens Commands
- Convert and Modify File Commands
- EBL Commands
- GBL Commands
- Kit Utility Commands
- Device Erase Commands
- Device Lock and Protection Commands
- Device Utility Commands
- External SPI Flash Commands
- Advanced Energy Monitor Commands
- Serial Wire Output Read Commands
- NVM3 Commands
- CTUNE Commands
- Security Commands
- Util Commands
- OTA Commands
- Post-Build Command
- RPS Commands
- VUART Commands
- RTT Commands
- Serial Commands
- Manufacturing Commands
- VCOM Commands
- Completion Commands
- LittleFS Commands

Copyright © 2026 Silicon Laboratories. All rights reserved.

Configure Commands

# Configure Commands

You can configure certain Certain Simplicity Commander settings by using the `configure` command. These settings persist across Commander runs and updates.

## Logging Overview

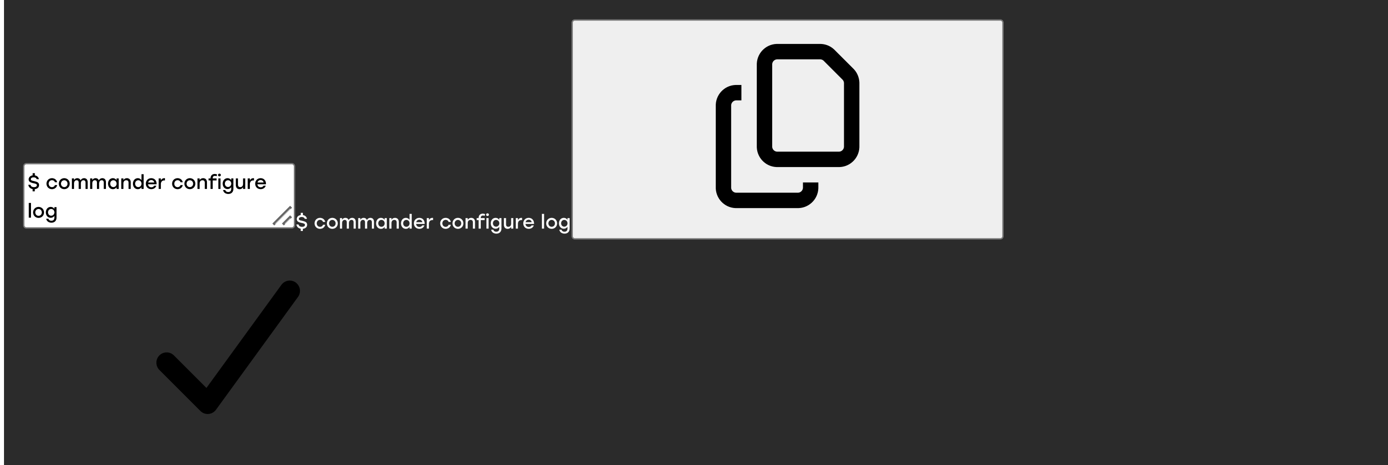
You can use Simplicity Commander to log all Commander invocations in a user-defined log file by using the `configure log` commands. Each entry in the log includes the following parameters:

- Timestamp in the form `YYYY-MM-DD hh:mm:ss.zzz`
- The full path to the Commander executable
- The command being executed, exactly as Commander receives it from the command line

## Get Current Logging Status

To view the the current logging status, use the `configure log` command.

Command Line Syntax

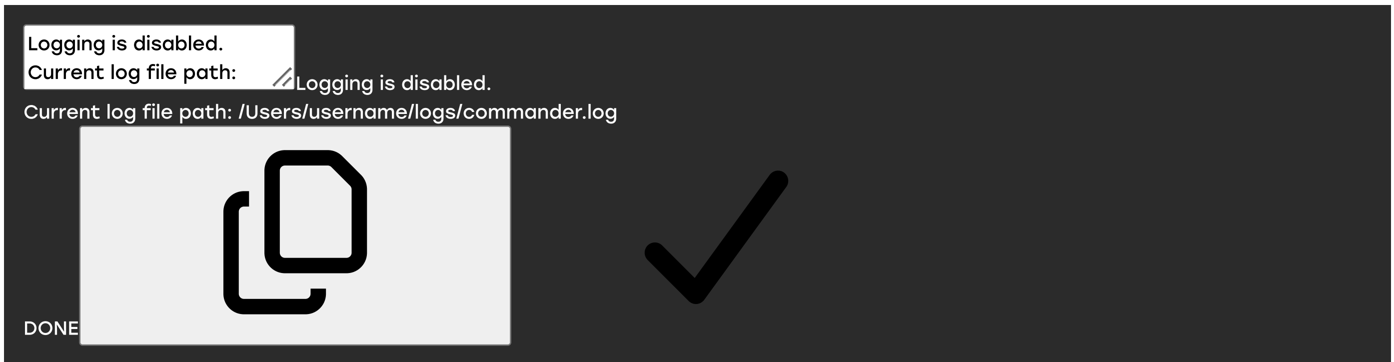



Command Line Input Example

This example gets the current logging status.

Copyright © 2026 Silicon Laboratories. All rights reserved.

Command Line Output Example



```
Logging is disabled.
Current log file path: /Users/username/logs/commander.log
DONE
```

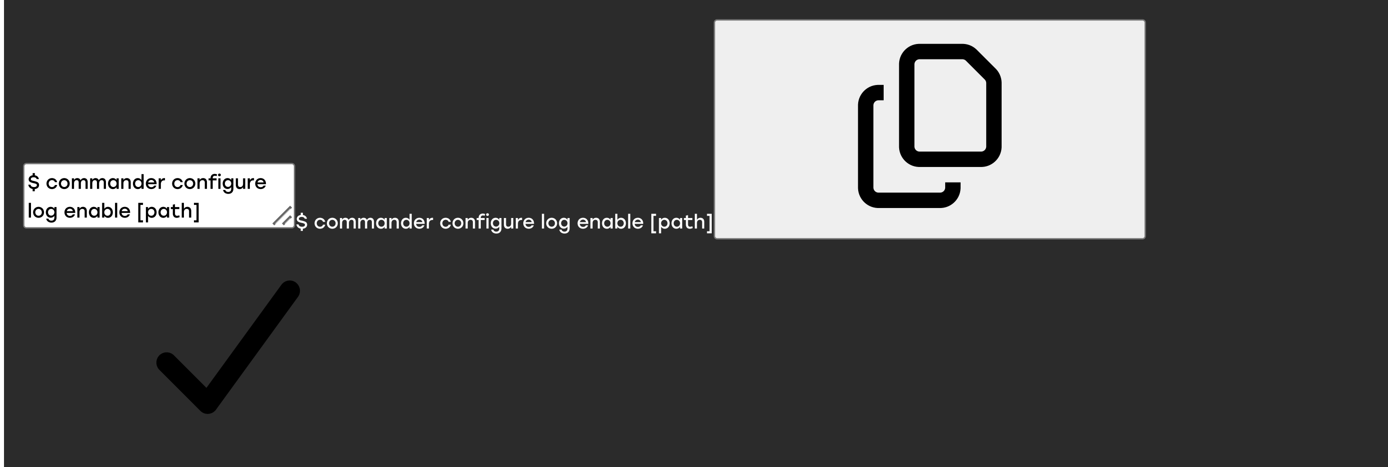
## Enable Logging

To enable logging, use the `configure log enable` command.

- If you provide a file name, Commander uses that file for the log output file.
- If no file name is provided, Commander uses the last known output file name.
- If logging previously hasn't been enabled, Commander uses `~/.silabs/commander/commander.log` as the default file name.

Commander will create the path to the log file if it does not already exist. If the log file itself does not exist when logging is enabled, it will be created upon the next invocation of Commander.

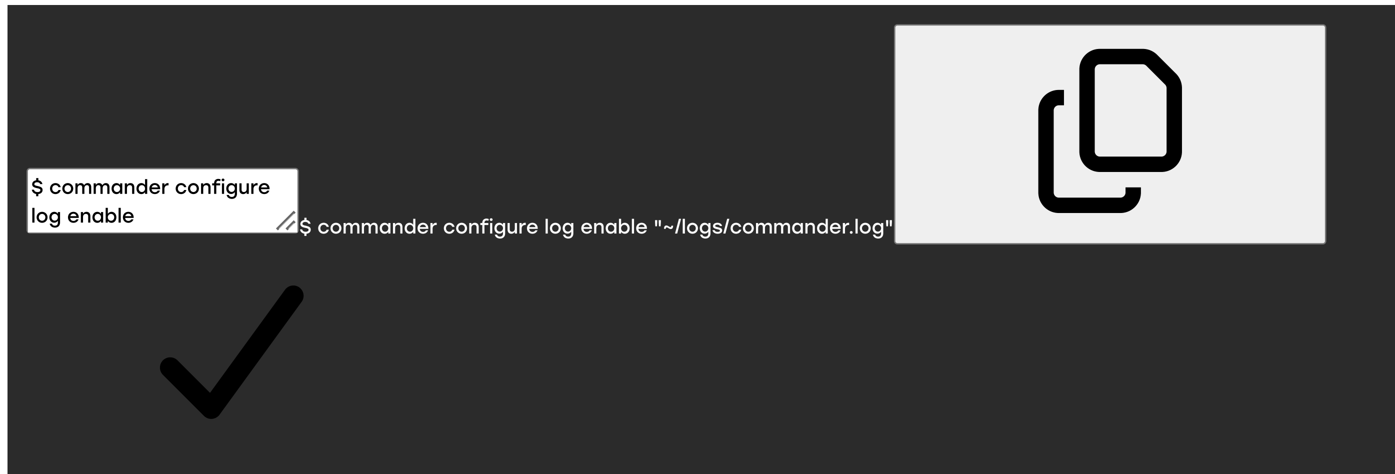
Command Line Syntax



```
$ commander configure log enable [path]
```

Command Line Input Example

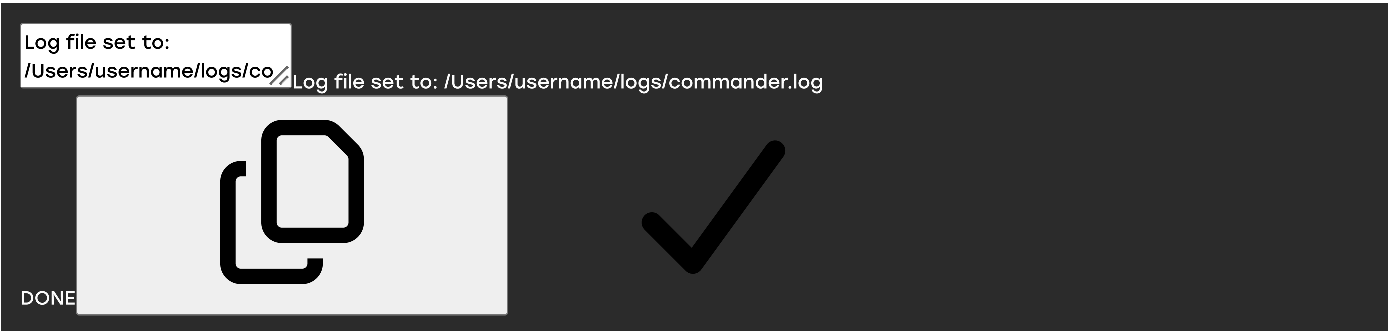
Copyright © 2026 Silicon Laboratories. All rights reserved.



```
$ commander configure log enable "~/logs/commander.log"
```

This example enables logging and sets `~/logs/commander.log` as the output log file.

Command Line Output Example

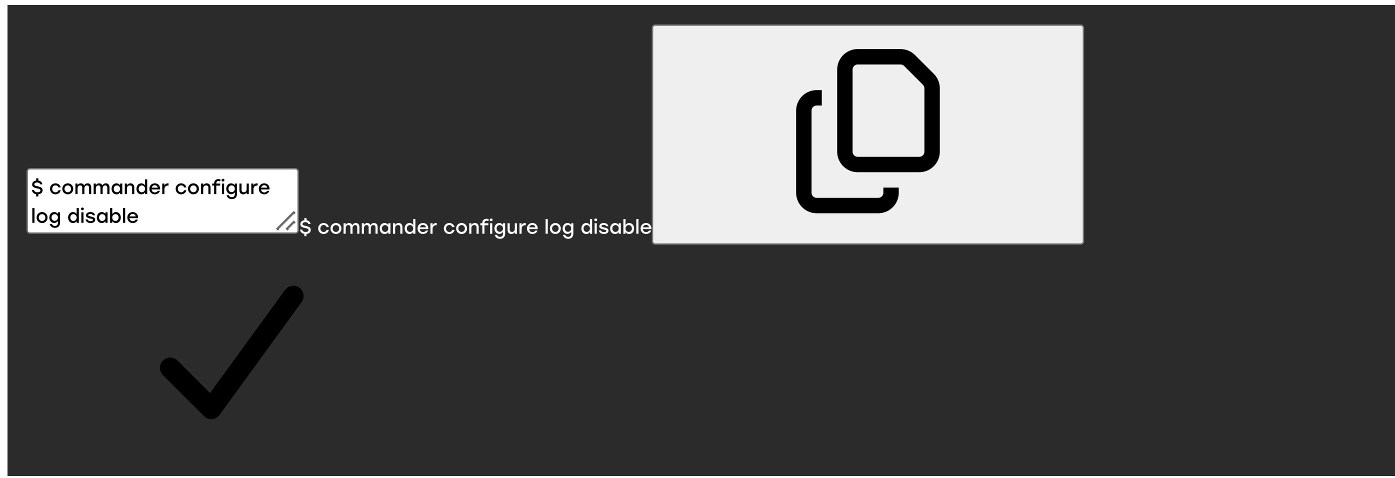


```
Log file set to: /Users/username/logs/commander.log
DONE
```

## Disable Logging

To disable logging, use the `configure log disable` command. Reenabling logging will use the last known output filename.

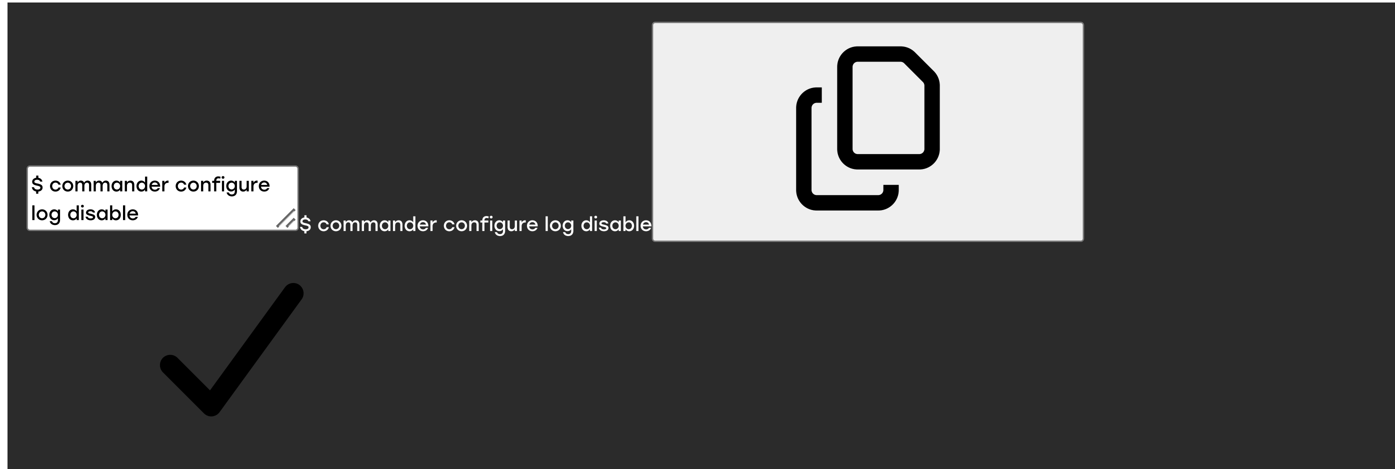
Command Line Syntax



```
$ commander configure log disable
```

Command Line Input Example

Copyright © 2026 Silicon Laboratories. All rights reserved.



```
$ commander configure log disable
```

This example disables logging.

Command Line Output Example

```
Logging disabled.
DONE
```

Copyright © 2026 Silicon Laboratories. All rights reserved.

# Device Flashing Commands

# 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.

## Flash Image File

Flashes the image in the specified filename to the target device, starting at the specified address. The address value is interpreted as a hexadecimal number. 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). The `--binary` option can be used to interpret all file types as flat binaries, bypassing any parsing of GBL, S-record, or Intel Hex files. For example, you can use this to test firmware upgrade using an internal storage bootloader. The `--include-section` and `--exclude-section` options can be used when flashing an Elf file.

Command Line Syntax

```
$ commander flash <filename> --address <address> --serialno <serial number> [--binary --include-section <section> --exclude-section <section>]
```

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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

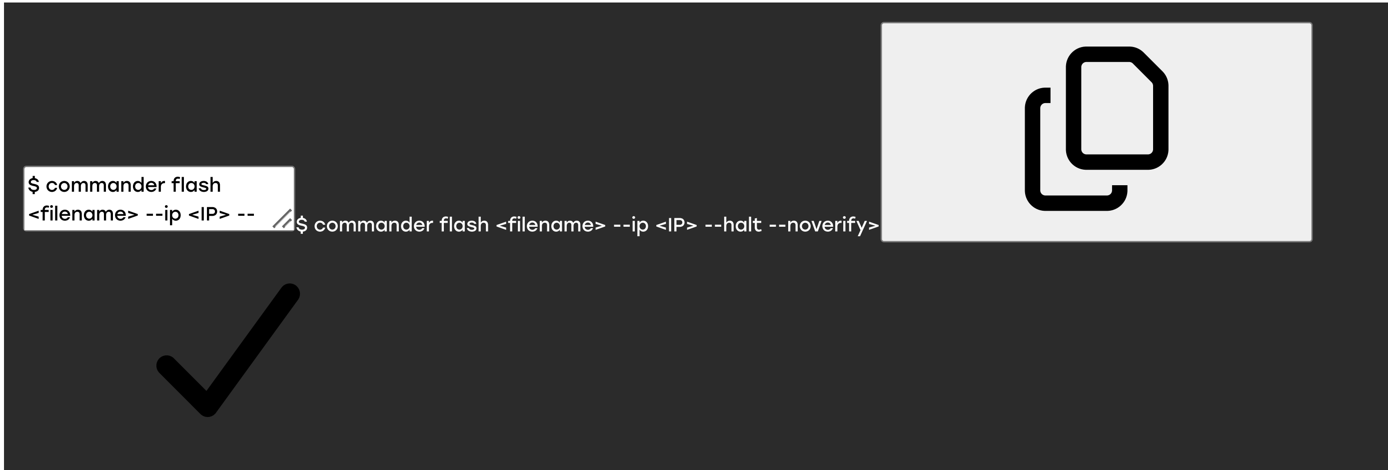
Command Line Output Example

```
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
```

## 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 Syntax



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

Command Line Input Example

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

Copyright © 2026 Silicon Laboratories. All rights reserved.

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.

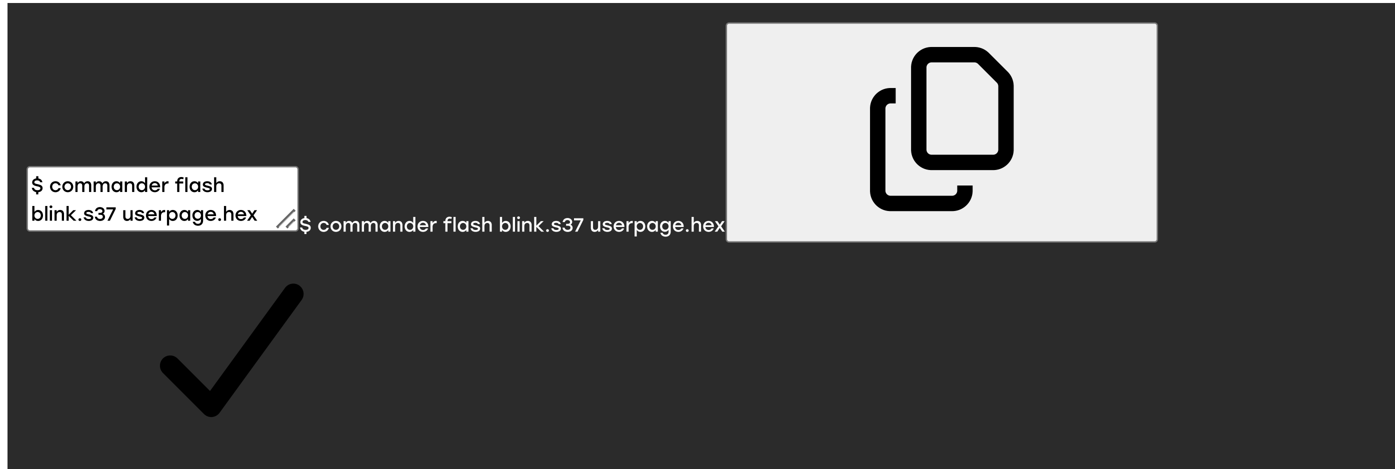
Command Line Output Example

## Flash Several Files

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

Command Line Syntax

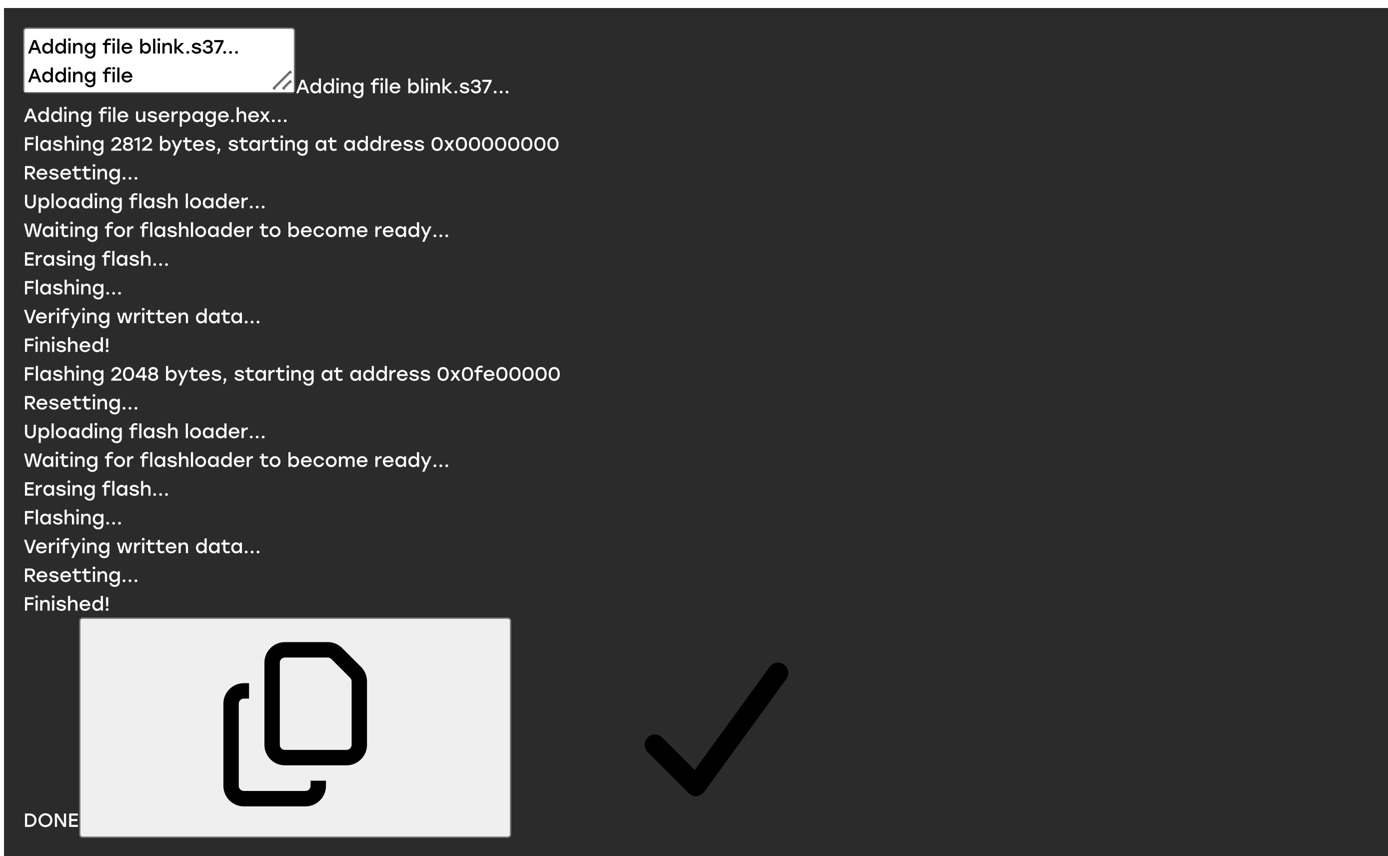
Command Line Input Example




Copyright © 2026 Silicon Laboratories. All rights reserved.

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
```

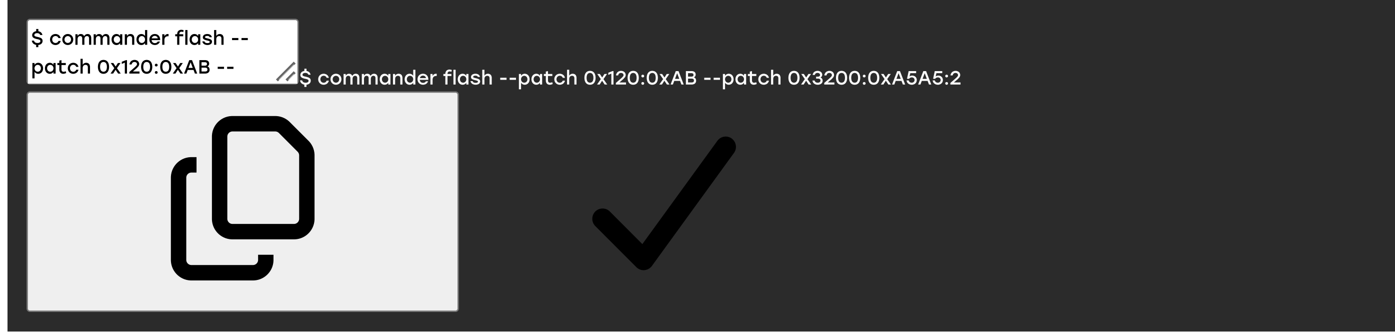
## Patch Flash

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 Syntax

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

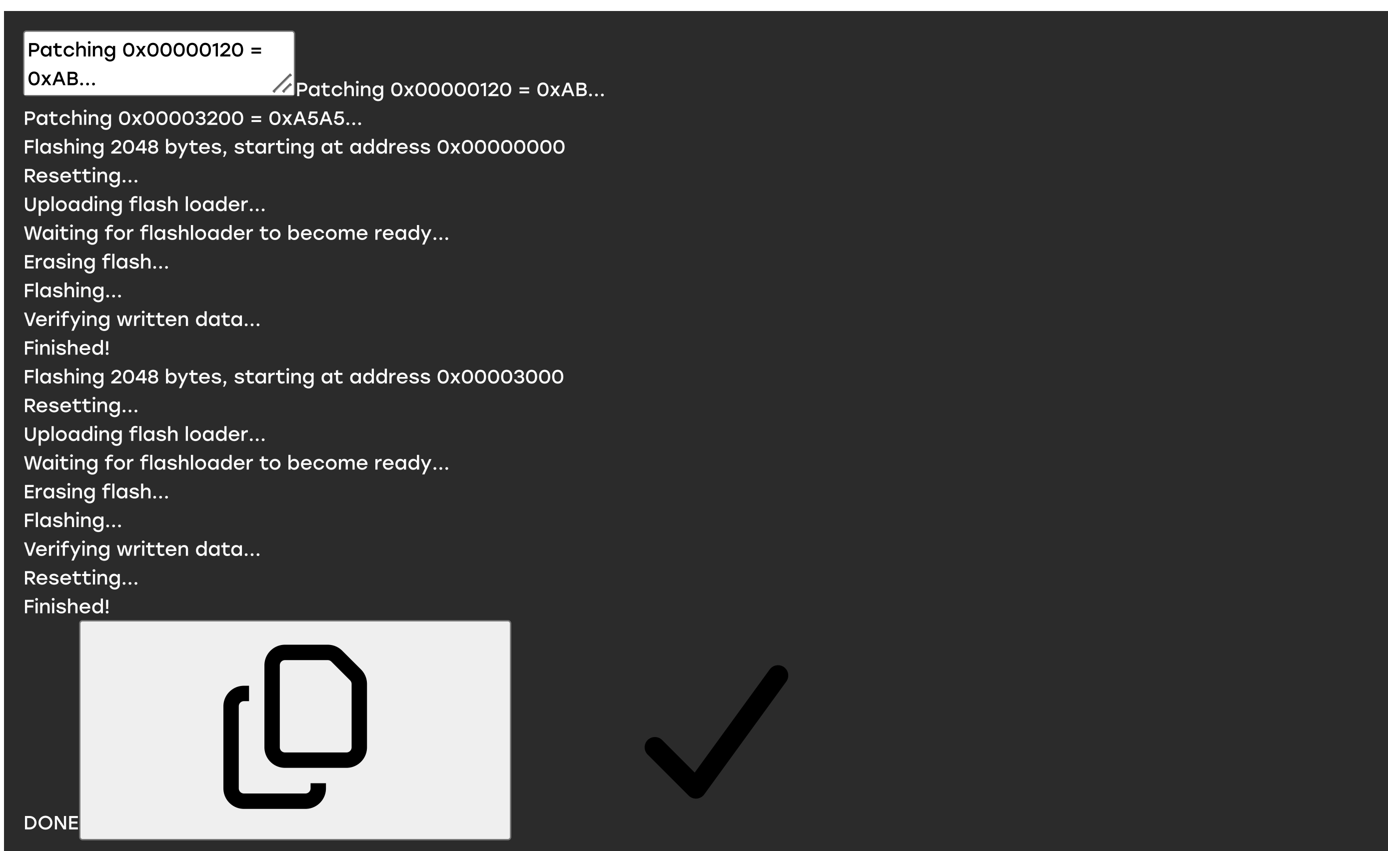
Command Line Input Example



```
$ 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
```

## 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.

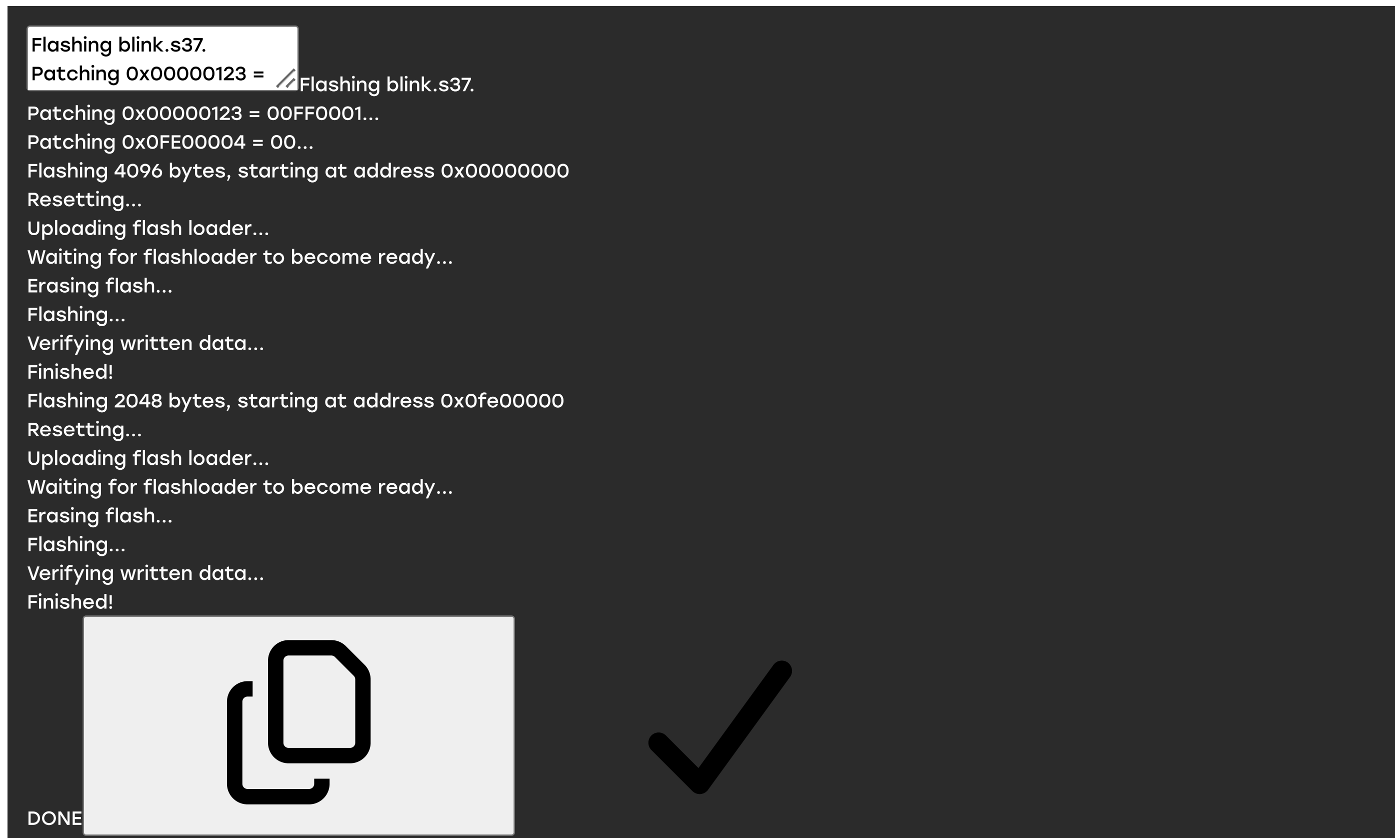
Command Line Syntax

Copyright © 2026 Silicon Laboratories. All rights reserved.

Command Line Input Example

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




## Flash Tokens

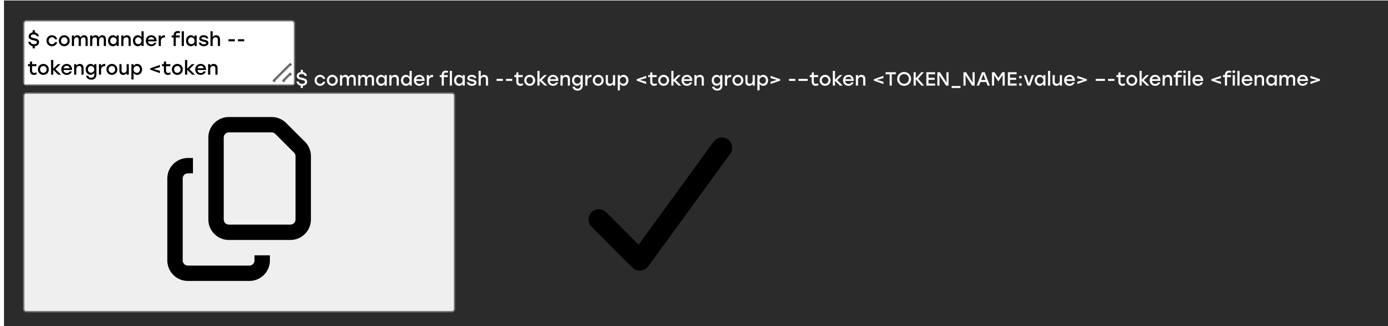
Copyright © 2026 Silicon Laboratories. All rights reserved.

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 EFR32MG and EFR32FG Families*.

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

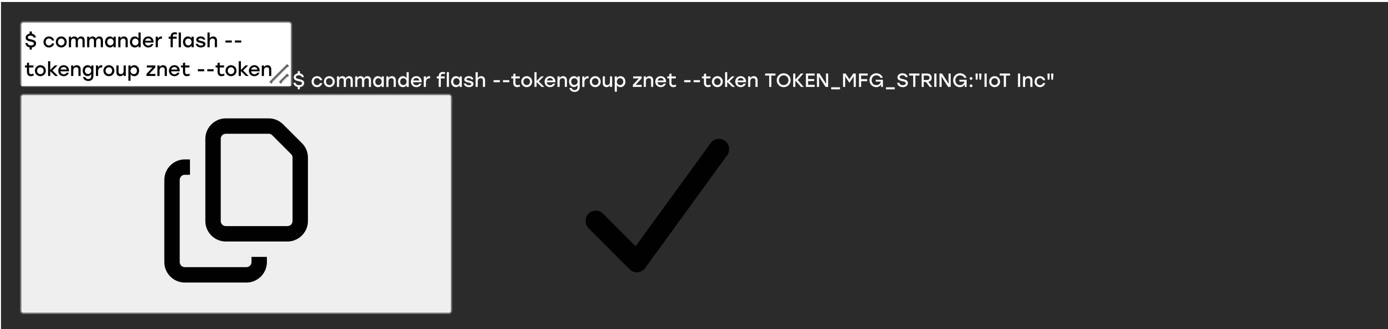
Silicon Labs recommends generating a token file from a device or image file using the `tokendump` command and then making modifications to this file for use with the `--tokenfile` option.

Command Line Syntax



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

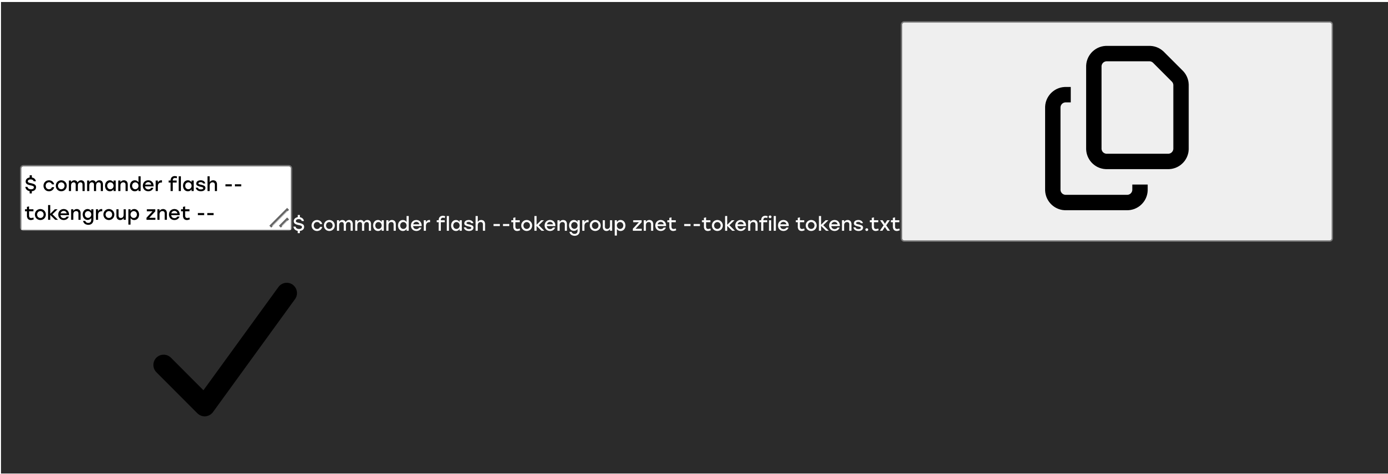
Command Line Input Example



```
$ 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

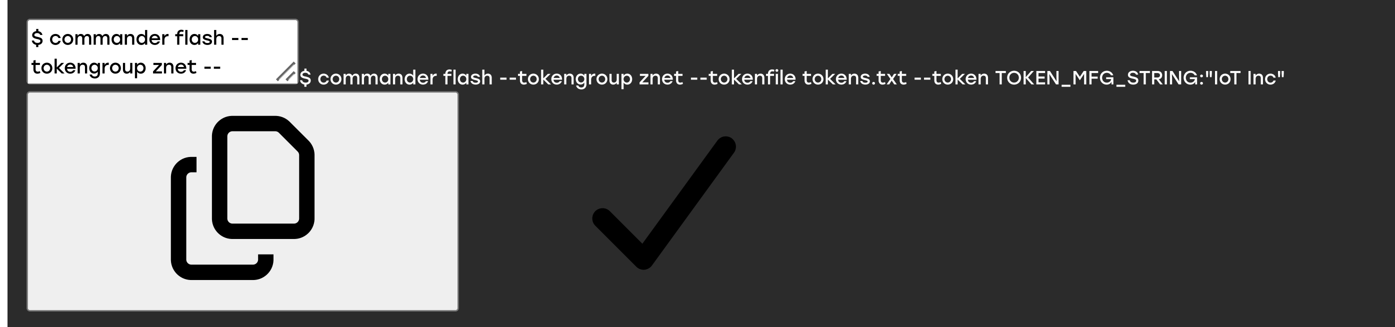


```
$ commander flash --tokengroup znet --tokenfile tokens.txt
```

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.

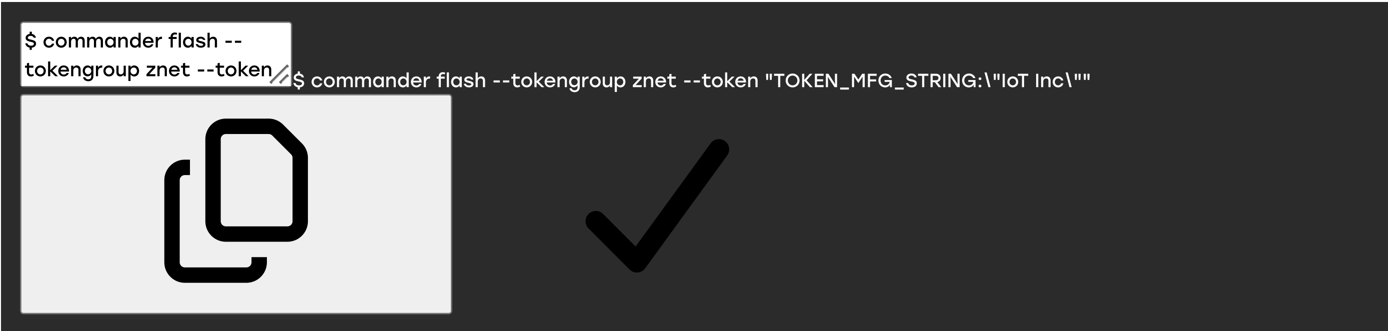
Command Line Input Example

Copyright © 2026 Silicon Laboratories. All rights reserved.

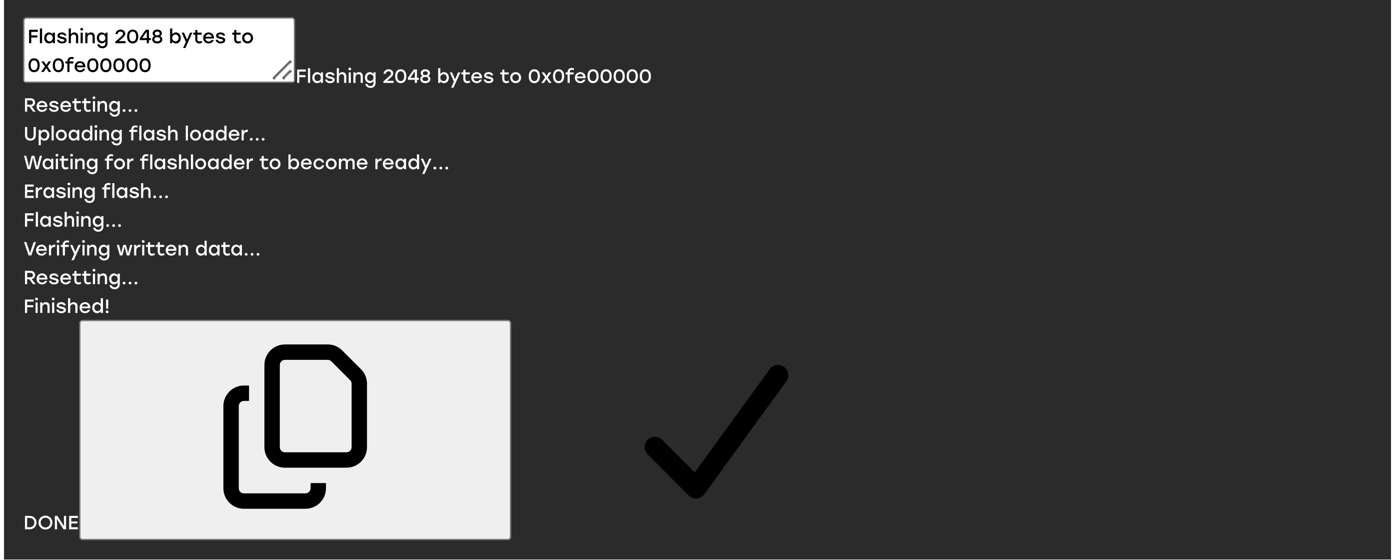



Sets the tokens specified in tokens.txt. Additionally, sets the `MFG_STRING` to the value given. All files and tokens specified on the command line are processed, and if a duplicate is found, it will be treated as an error.

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:




Command Line Output Example




Copyright © 2026 Silicon Laboratories. All rights reserved.

# Flash Verification Command

# Flash Verification Command

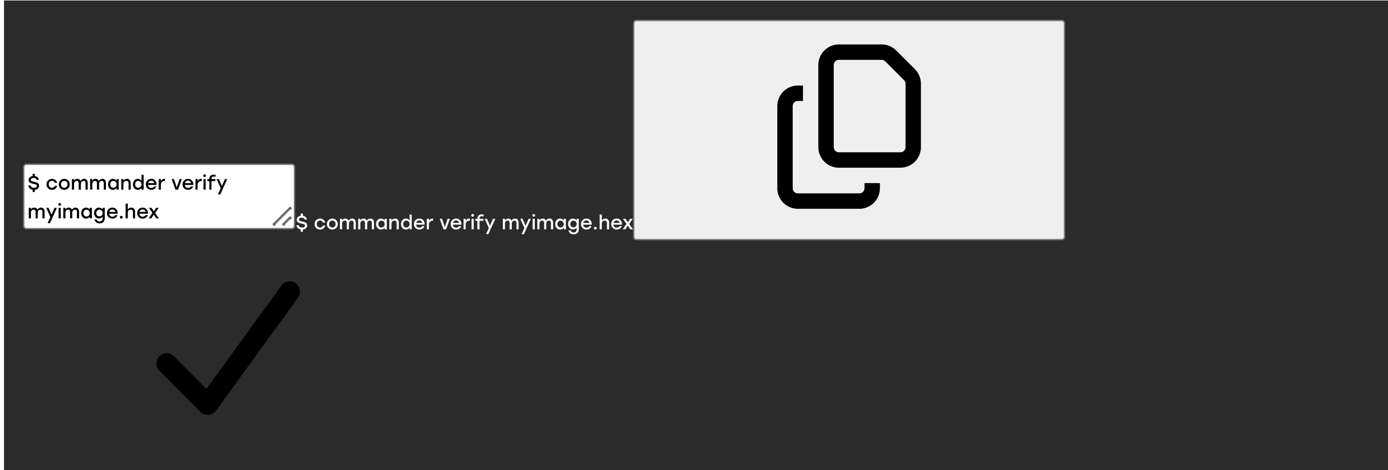
The `verify` command verifies the contents of a device against a set of files, tokens, and/or patch options without writing anything to the flash. It works just like the verification step of the `flash` command, but without actually flashing first. For example, the `verify` command can be used to verify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

Command Line Syntax

All options and examples for the `flash` command also apply to the `verify` command. The exceptions are 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

Copyright © 2026 Silicon Laboratories. All rights reserved.

Copyright © 2026 Silicon Laboratories. All rights reserved.

# Memory Read Commands

# 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. You 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.

Series 0 EFM32, EZR32, EFR32: `@mainflash`, `@userdata`, `@lockbits, @devinfo`

Series 1 EFM32, EFR32: `@mainflash`, `@userdata`, `@lockbits`, `@devinfo`, `@bootloader`

Series 2 EFM32, EFR32: `@mainflash`, `@userdata`, `@devinfo`

EM3xx: `@mfb`, `@cib`, `@fib`

## Print Flash Contents

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

Command Line Syntax

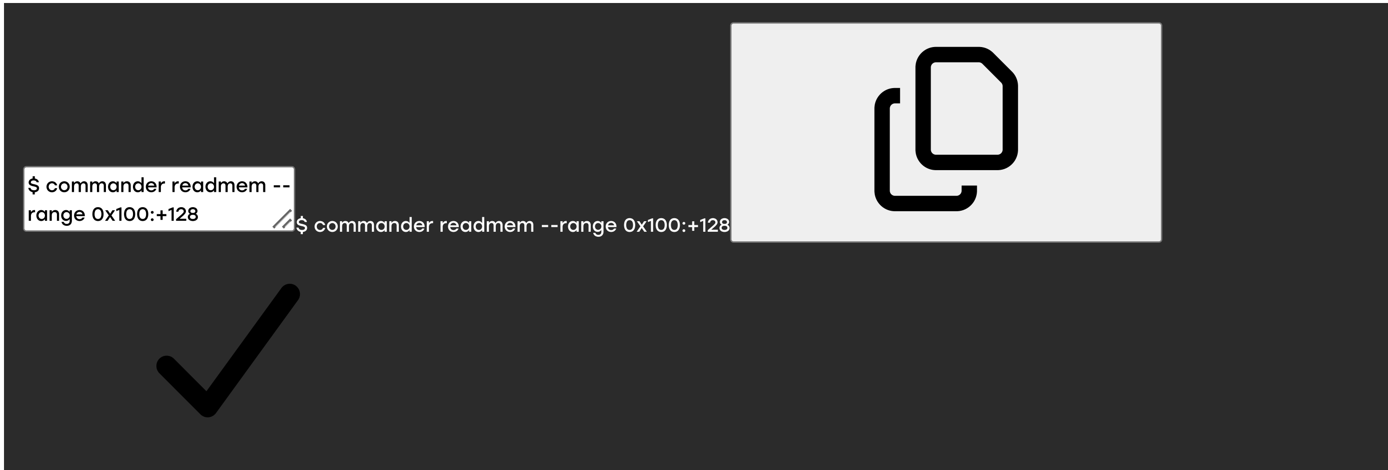
Copyright © 2026 Silicon Laboratories. All rights reserved.

OR

Command Line Syntax

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

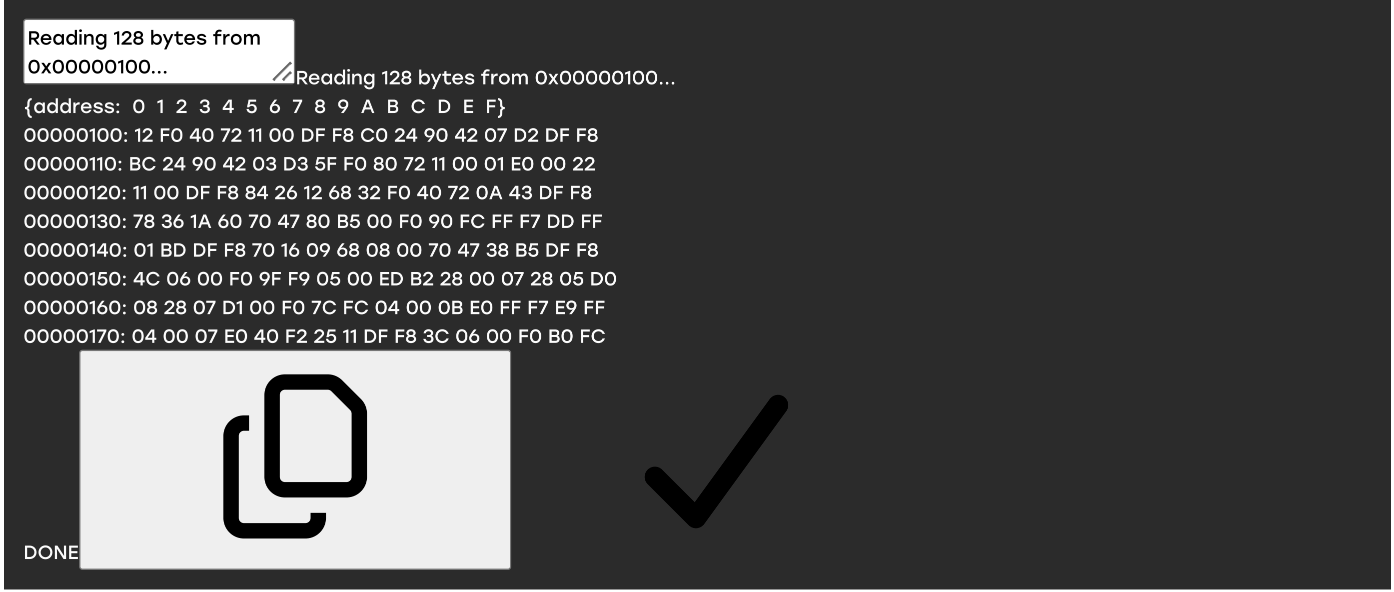
Command Line Input Example



```
$ 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
```

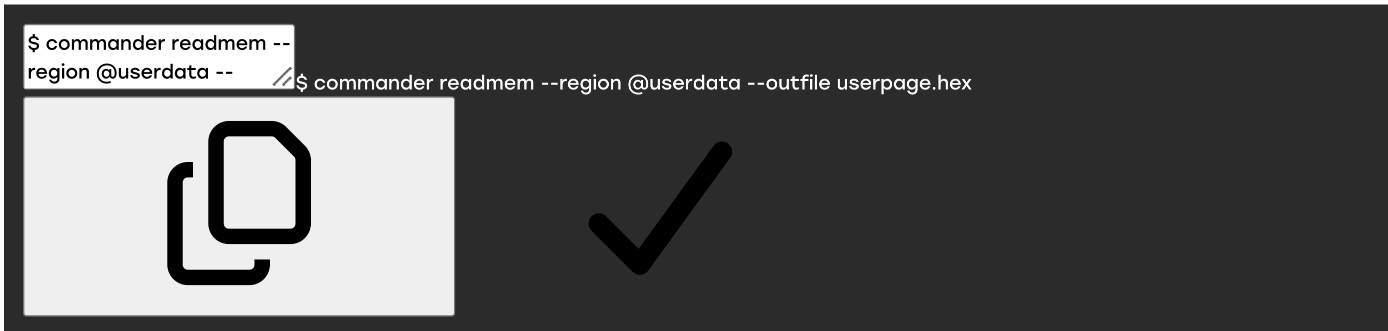
## Dump Flash Contents to File

Copyright © 2026 Silicon Laboratories. All rights reserved.

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). See File Format Overview for more information on file formats.

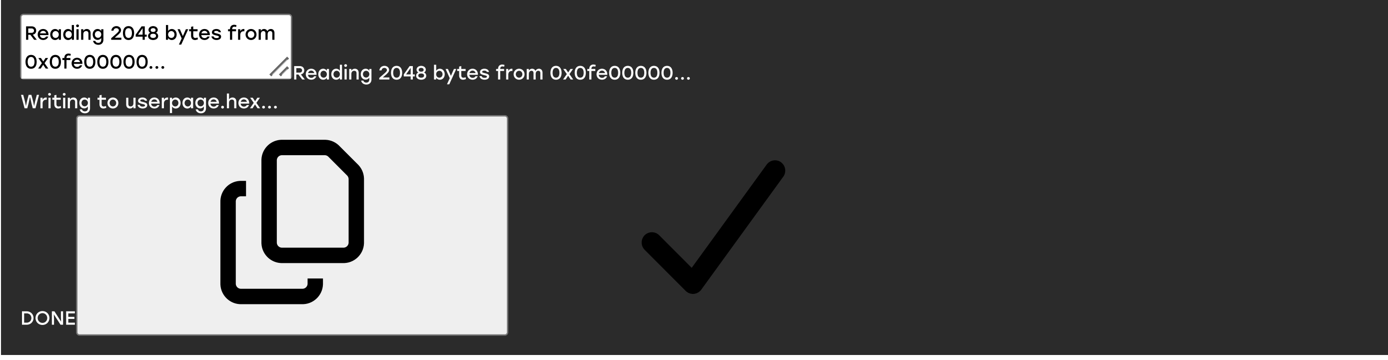
Command Line Syntax

Command Line Input Example




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

Command Line Output Example




Copyright © 2026 Silicon Laboratories. All rights reserved.

# Tokens Command

# Tokens Command

The `tokens` command is the new unified interface for managing manufacturing tokens and static tokens on Silicon Labs devices. It replaces the legacy token handling in `flash`, `tokendump` and `tokenheader` commands, providing a consistent experience across device families.

This command supports both the old manufacturing tokens format and the new static tokens format.

- Manufacturing tokens are used on Series 1 and 2 devices. These tokens are defined with fixed length and fixed locations in memory.
- Static tokens are used on Series 3 devices. These tokens use a Key-Length-Value (KLV) format. There are two types of static tokens
  - Static Secure Tokens are stored in a dedicated flash region. The secure range must be provided when reading/writing these tokens. Static secure tokens are encrypted by default.
  - Static device tokens have limited storage capacity and are mass-erase protected. The number of writes/erases for static device tokens is limited to 100 due to hardware OTP bit constraints.

> Important:
>
> - The KLV chain will be automatically initialized by the `tokens write` command if it is not already present.
> - The `tokens erase` command also leaves the KLV chain initialized after erasing.
> - All token commands require information about the target device. This requires that either the `--device` option is given, or that the device part number can be inferred from a known Silicon Labs board.
> - If neither the `--tokengroup` nor `--tokendefs` option is specified, Simplicity Commander defaults to using all known token groups. For static tokens this means `common` and `zigbee`, and for manufacturing tokens this means `znet`.

## Write Tokens

The `tokens write` command writes tokens to the device. It replaces the legacy `flash --token` and `flash --tokenfile` commands and supports both manufacturing tokens and static tokens. The available options and behavior differ depending on the device type.

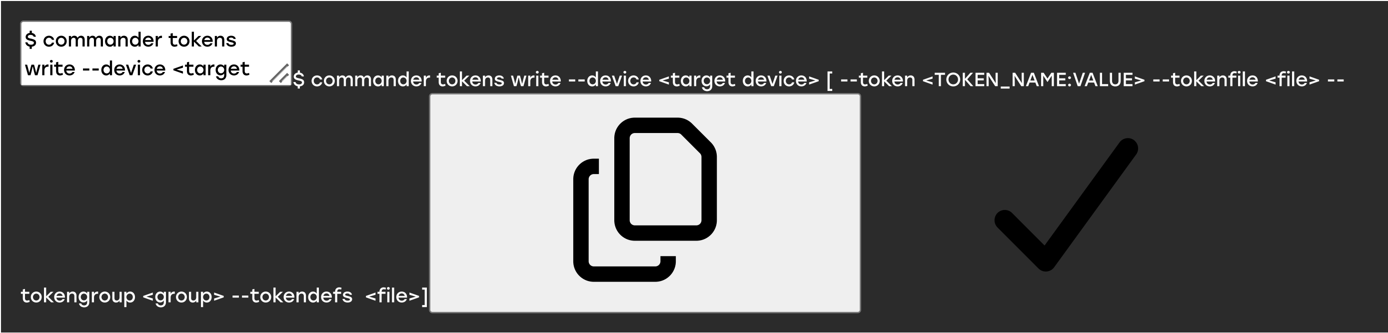
### Command Line Syntax

#### Static Tokens

Copyright © 2026 Silicon Laboratories. All rights reserved.

```
$ commander tokens write  --device <target device> [--token <TOKEN_NAME:VALUE> --tokenfile <file> --securerange <address1:address2> --tokengroup <group> --tokendefs <file>]
```

Manufacturing Tokens



```
$ commander tokens write --device <target device> [ --token <TOKEN_NAME:VALUE> --tokenfile <file> --tokengroup <group> --tokendefs  <file>]
```

> Note:
>
> - At least one of `--token` or `--tokenfile` must be provided.
> - For static tokens `--securerange` is required for secure tokens.

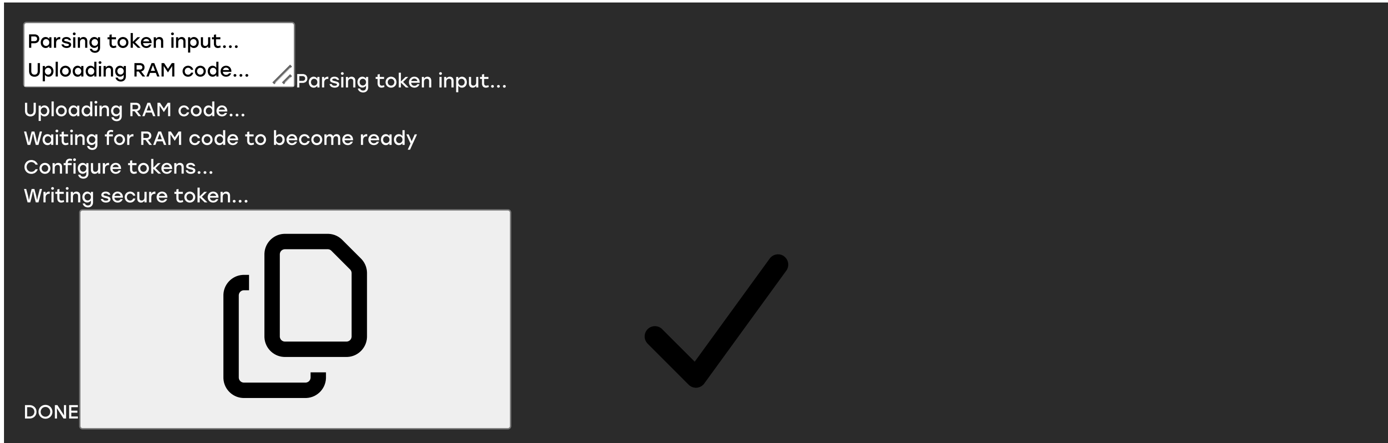
## Command Line Input Example (Static Tokens)

```
$ commander tokens write --securerange 0x0138A000:+0x2000  --token "Install Code:12345123451234512341234512345123" --device SiMG301M104LILB0
```

Writes the `Install Code` token to the device in the secure range.

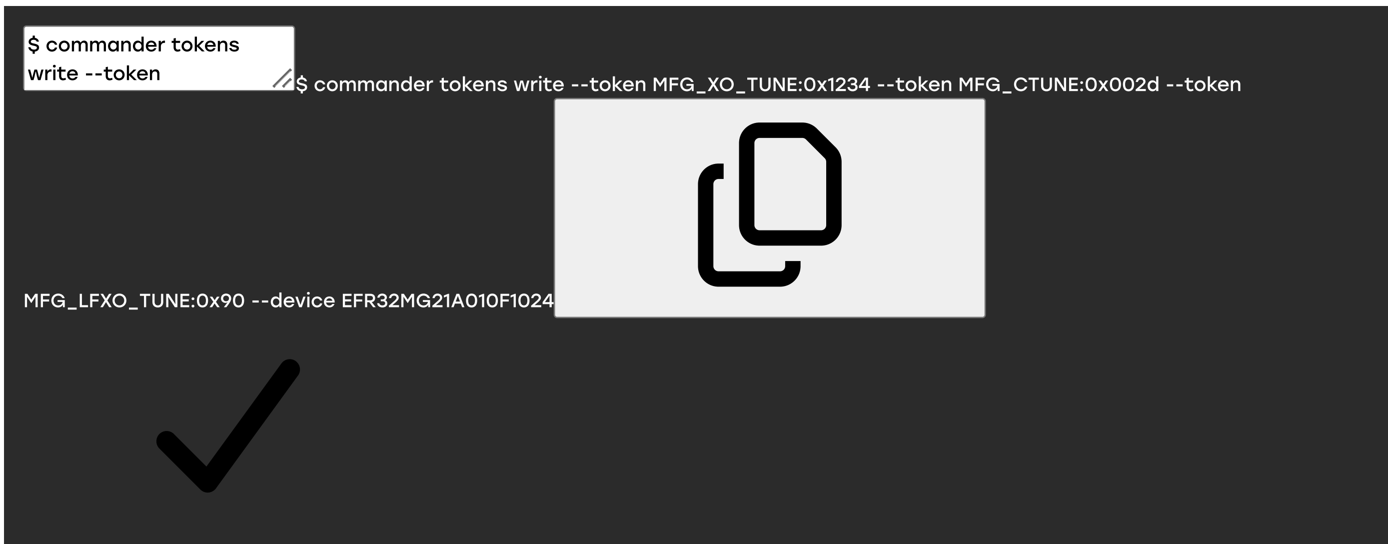
Copyright © 2026 Silicon Laboratories. All rights reserved.

### Command Line Output Example



```
Parsing token input...
Uploading RAM code...
Waiting for RAM code to become ready
Configure tokens...
Writing secure token...
DONE
```

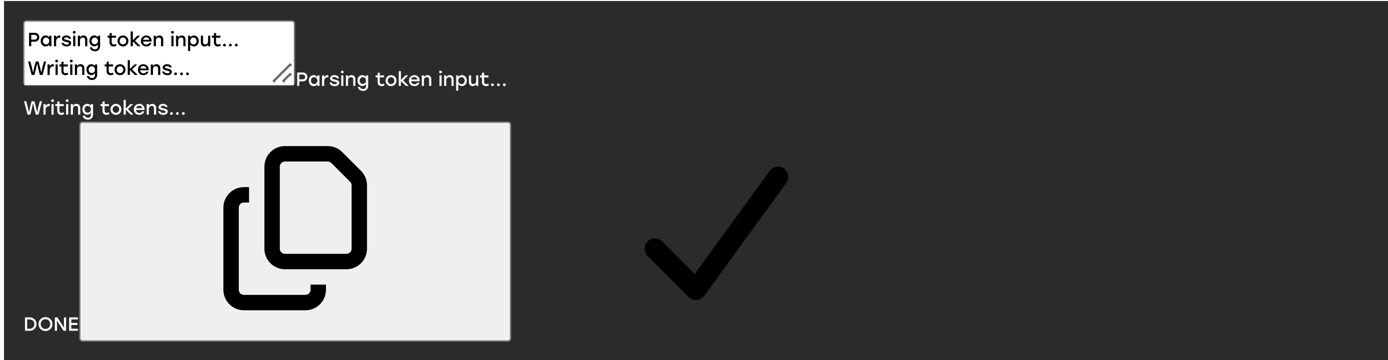
### Command Line Input Example (Manufacturing Tokens)



```
$ commander tokens write --token MFG_XO_TUNE:0x1234 --token MFG_CTUNE:0x002d --token MFG_LFXO_TUNE:0x90 --device EFR32MG21A010F1024
```

Writes the `MFG_XO_TUNE` , `MFG_CTUNE` , and `MFG_LFXO_TUNE` tokens to the device.

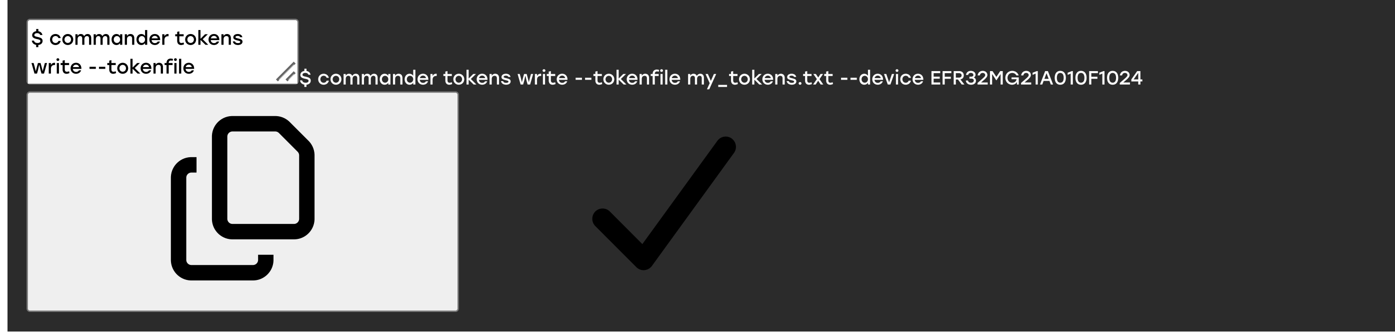
### Command Line Output Example



```
Parsing token input...
Writing tokens...
DONE
```

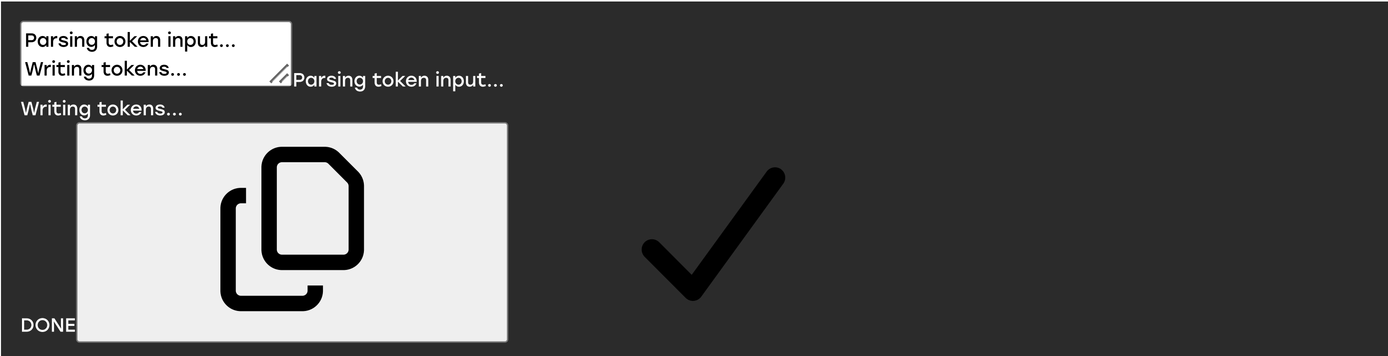
### Command Line Input Example (Using Token File)

Copyright © 2026 Silicon Laboratories. All rights reserved.




Writes all tokens specified in `my_tokens.txt` to the device. A template token file can be generated using the `tokens read` command.

## Command Line Output Example



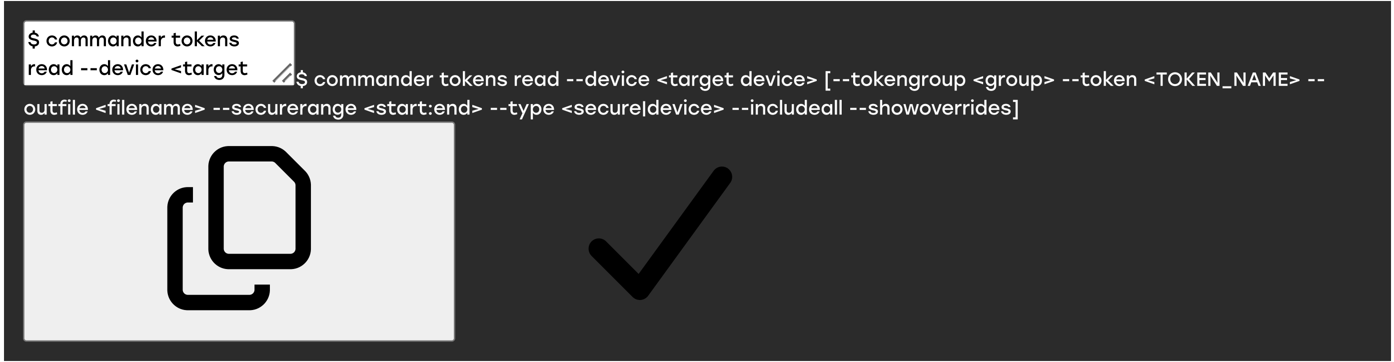

# Read Tokens

The `tokens read` command reads tokens from a device or one or more files (manufacturing tokens only). The output of `read` can either be printed to standard output or written to a file using the `--outfile` option. The output file can be modified and re-used as input to the `tokens write`, `verify`, or `convert` commands using the `--tokenfile` option.

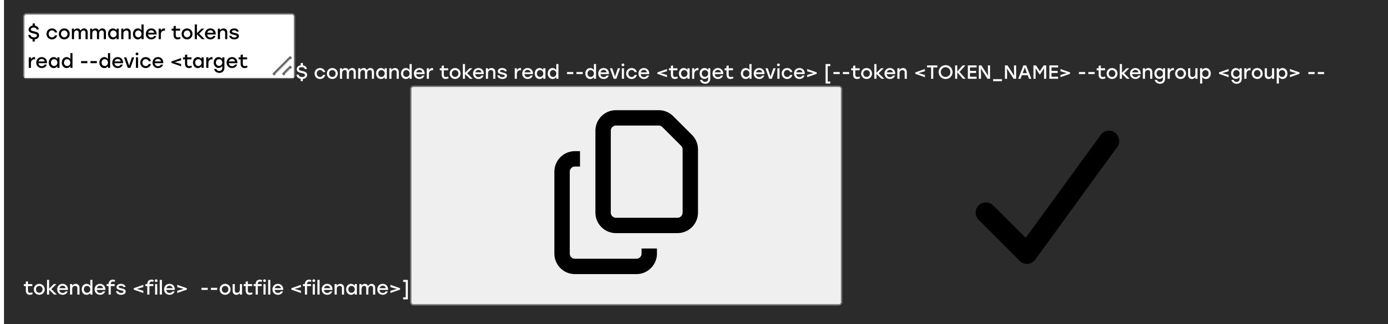
For static tokens, tokens are read directly from the device. All tokens will be included if the `--includeall` option is set; otherwise, only tokens found on the device are shown. Use `--showoverrides` to display any potential NVM3 overrides on the device.

## Command Line Syntax

### Static Tokens



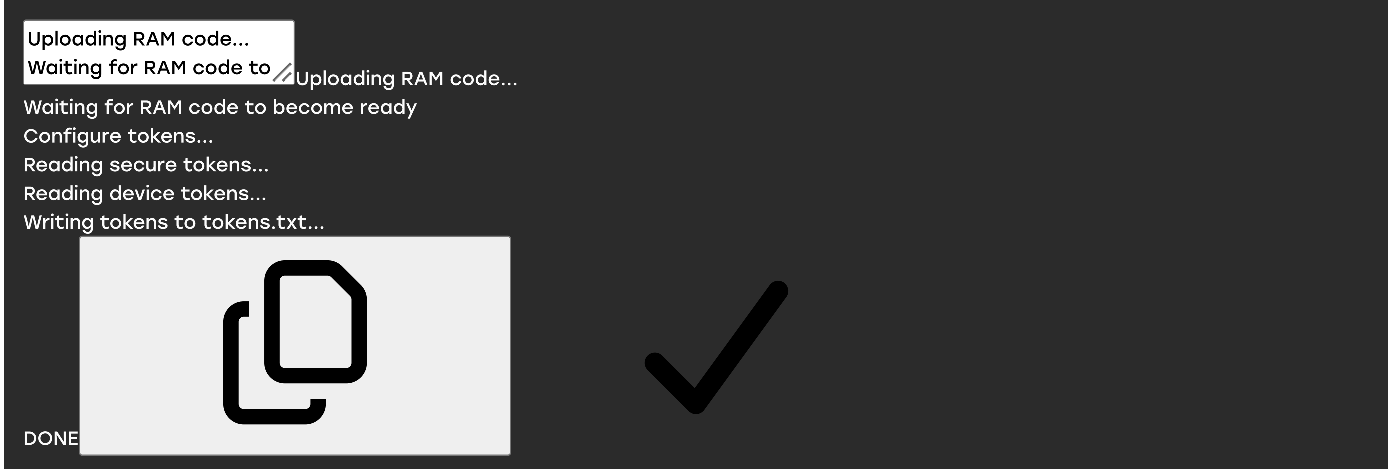

### Manufacturing Tokens



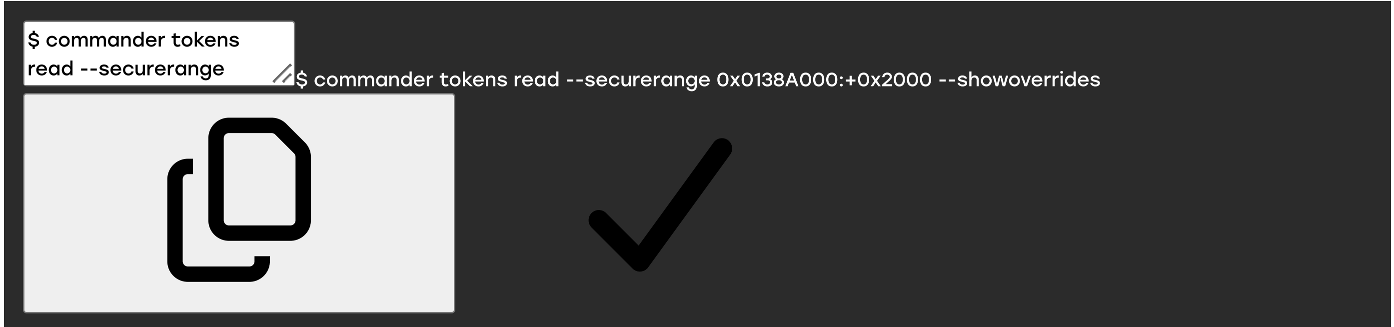

## Command Line Input Example (Static Tokens)

Reads all static tokens from the device in the specified secure range and writes them to `tokens.txt`.

## Command Line Output Example




## Command Line Input Example (Static Tokens, with NVM3 overrides)




Reads all static tokens from the device, including any NVM3 overrides, and prints them to the console.

## Command Line Output Example

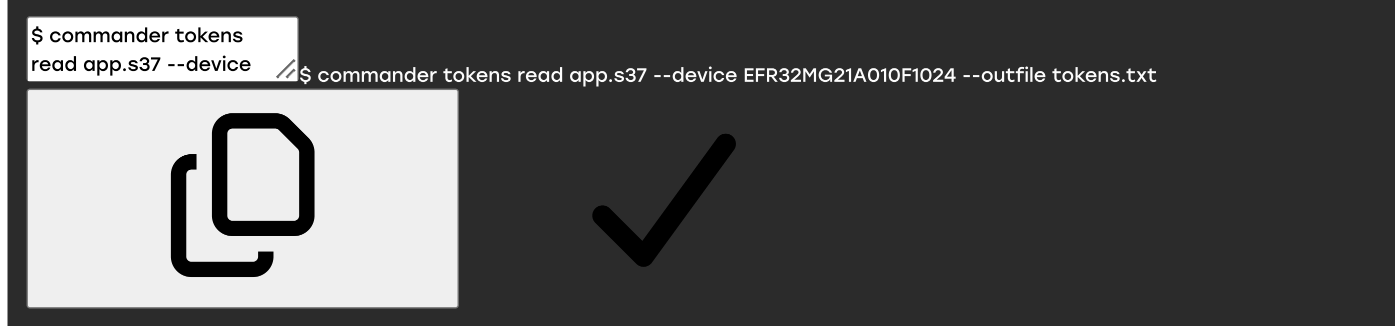
Copyright © 2026 Silicon Laboratories. All rights reserved.

```
Found NVM3 range:
0x01381000 - 0x01385000 Found NVM3 range: 0x01381000 - 0x01385000. Security mode: aead
Uploading RAM code...
Waiting for RAM code to become ready
Configuring secure tokens...
Reading secure tokens...
Reading device tokens...
Opening NVM3...
Getting NVM3 objects...
Closing NVM3...
Resetting device
#
# 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.
#
#
# Lines beginning with '# // ' are static tokens that are overridden in NVM3.
# The NVM3 override token is presented on the following line.
#
MFG_CUSTOM_EUI_64 : ABCDEF0123456789
MFG_CUSTOM_VERSION: 0x1337
MFG_PHY_CONFIG    : 0x1234
MFG_LFXO_TUNE     : 0x69
MFG_CTUNE         : 0x002d
# // MFG_MANUF_ID      : 0xCAFE [overridden]
MFG_MANUF_ID      : 0x3412
MFG_CCA_THRESHOLD : 0x5A5A
MFG_BOARD_NAME    : "My Board"

#'MFG_CBKE_283K1_DATA (Smart Energy 1.2 CBKE)' token group
Device Implicit Cert (283k1) :
12345123451234123451234512345123412345123451234512341234512123451234512345123412345123451231234512345123451234123451234512345 1

CA Public Key (283k1)        : 123451234512341234512345123451234123451234512345123412345121234512345
Device Private Key (283k1)   : 123451234512341234512345123451234123451234512345123412345121234512 3
# CBKE FLAGS (283k1)          : 0x00

#'MFG_CBKE_DATA (Smart Energy CBKE)' token group
Device Implicit Cert : 1234512345123412345123451234512341234512345123451234123451212345123451234512341234512345123
CA Public Key        : 1234512345123412345123451234512341234512345123412345 1
Device Private Key   : 1234512345123412345123451234512345123412 34
# CBKE Flags          : 0x00

#'MFG_INSTALLATION_CODE (Smart Energy Install Code)' token group
# Install Code Flags : 0x0006
Install Code       : 12345123451234123451234512345123
# CRC              : 0xD6F2
DONE
```

Command Line Input Example (Manufacturing Tokens)



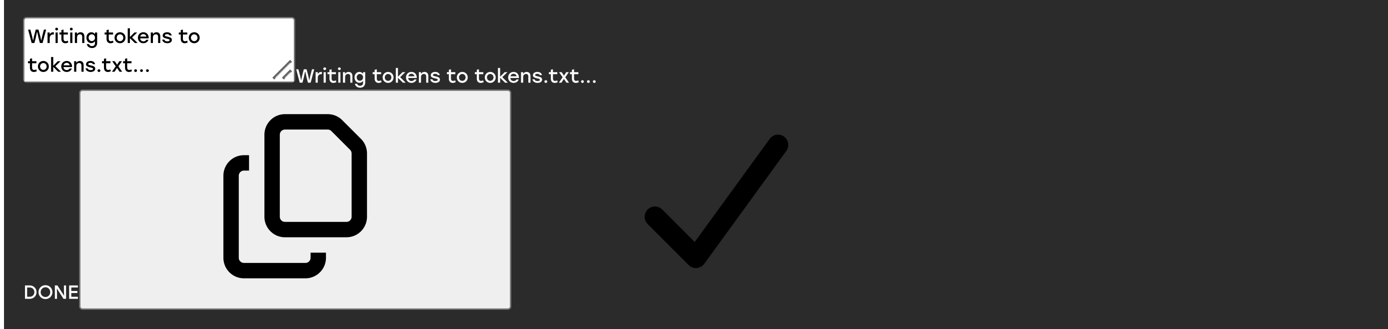
Copyright © 2026 Silicon Laboratories. All rights reserved.



```
$ commander tokens read app.s37 --device EFR32MG21A010F1024 --outfile tokens.txt
```

Reads all tokens from `app.s37` and writes them to `tokens.txt`.

### Command Line Output Example



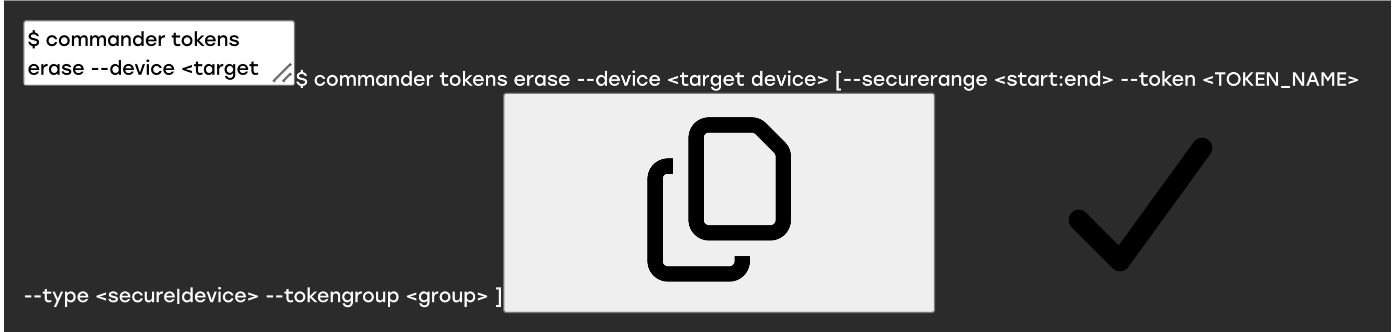
```
Writing tokens to tokens.txt...
DONE
```

## Erase Tokens

The `tokens erase` command erases tokens from a device. Specific tokens can be erased using the `--token` option. For static tokens, you can use this command to erase all tokens, only secure tokens, only device tokens, or specific tokens. If only a subset of the tokens are erased, the remaining tokens are rewritten to maintain a valid KLV chain. If all tokens of a given type are deleted, Commander will leave the KLV chain initialized. Every erase operation involving device tokens consume one OTP bit, so it is recommended to minimize erase/write operations for these tokens. Use the `--type` option to specify whether to erase secure or device tokens. For manufacturing tokens, erasing sets the token memory to 0xFF.

### Command Line Syntax

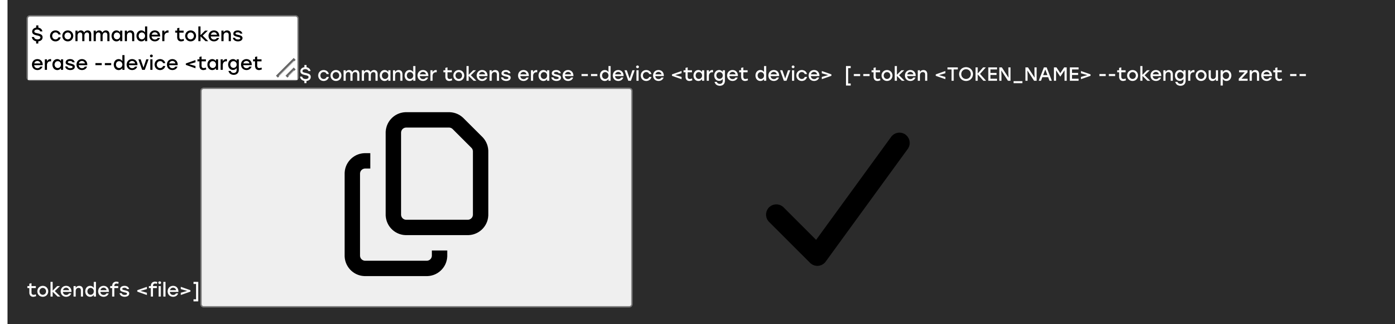
#### Static Tokens



```
$ commander tokens erase --device <target device> [--securerange <start:end> --token <TOKEN_NAME>
--type <secure|device> --tokengroup <group> ]
```

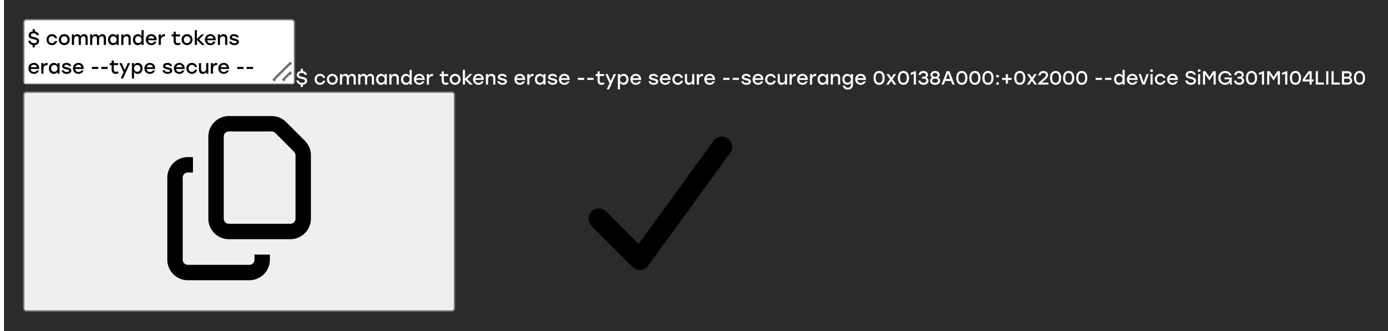
#### Manufacturing Tokens

Copyright © 2026 Silicon Laboratories. All rights reserved.



```
$ commander tokens erase --device <target device>  [--token <TOKEN_NAME> --tokengroup znet --tokendefs <file>]
```

## Command Line Input Example (Erase all secure tokens, Static Tokens)



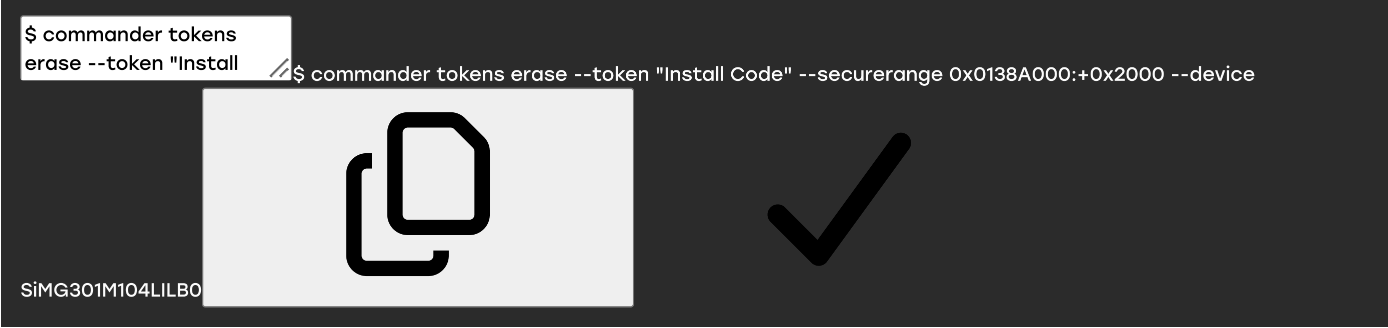
```
$ commander tokens erase --type secure --securerange 0x0138A000:+0x2000 --device SiMG301M104LILB0
```

Erases all secure static tokens from the device in the specified secure range.

## Command Line Output Example

```
Uploading RAM code...
Waiting for RAM code to become ready
Configure tokens...
Erasing secure tokens...
DONE
```

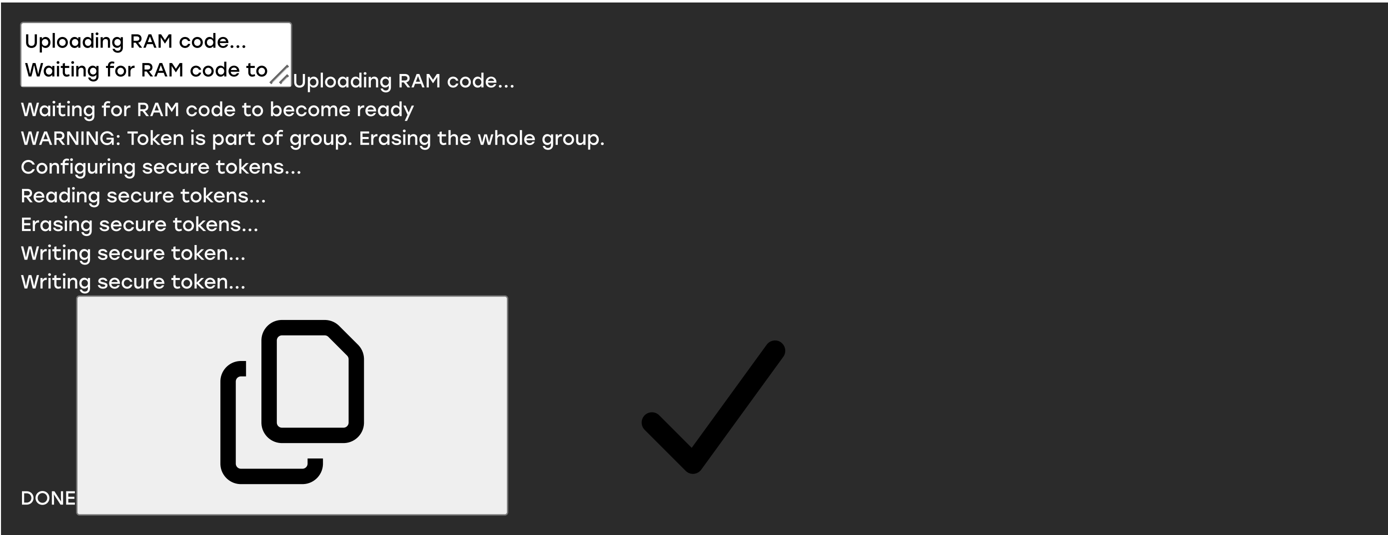
## Command Line Input Example (Erase specific token, Static Tokens)



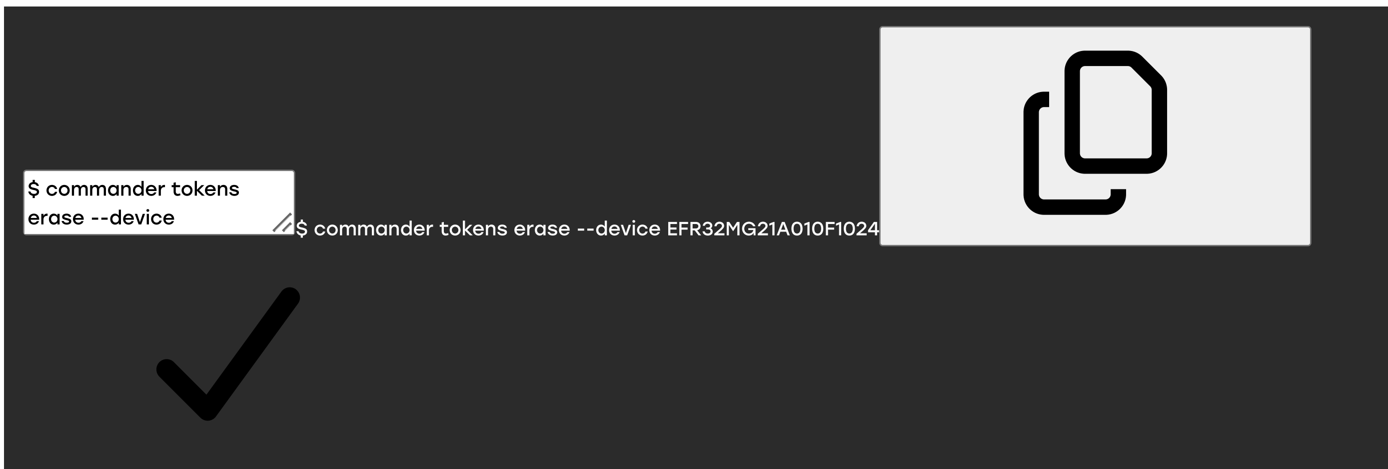
```
$ commander tokens erase --token "Install Code" --securerange 0x0138A000:+0x2000 --device SiMG301M104LILB0
```

Erases the `Install Code` static token from the device. Rewrites two remaining secure tokens to the KLV chain.

## Command Line Output Example

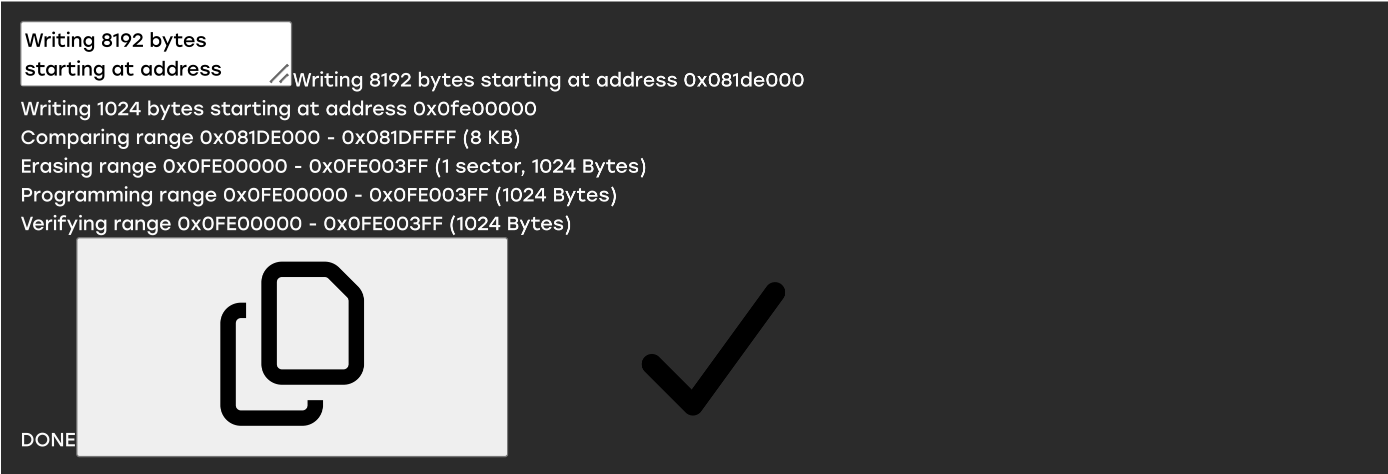



### Command Line Input Example (Erase all tokens, Manufacturing Tokens)




Erases all manufacturing tokens from the device in the `znet` token group (sets memory to 0xFF).

### Command Line Output Example

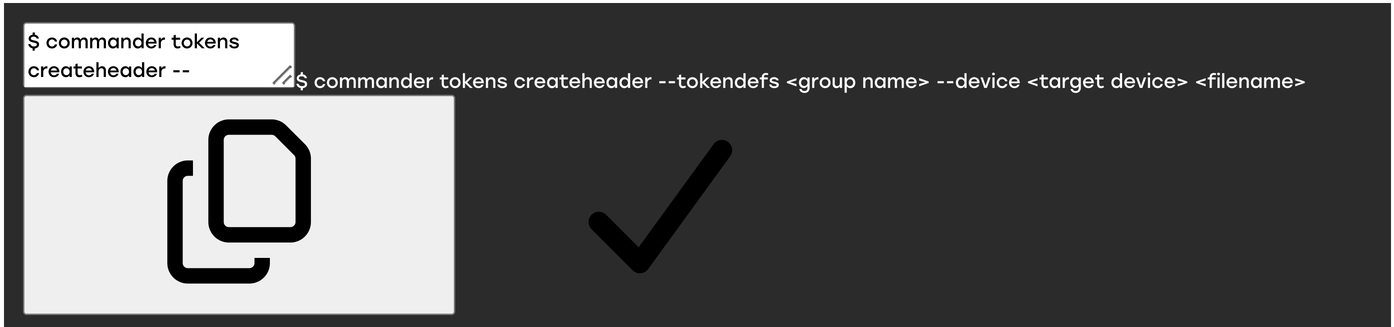



## Generate C Header Files from Token Groups

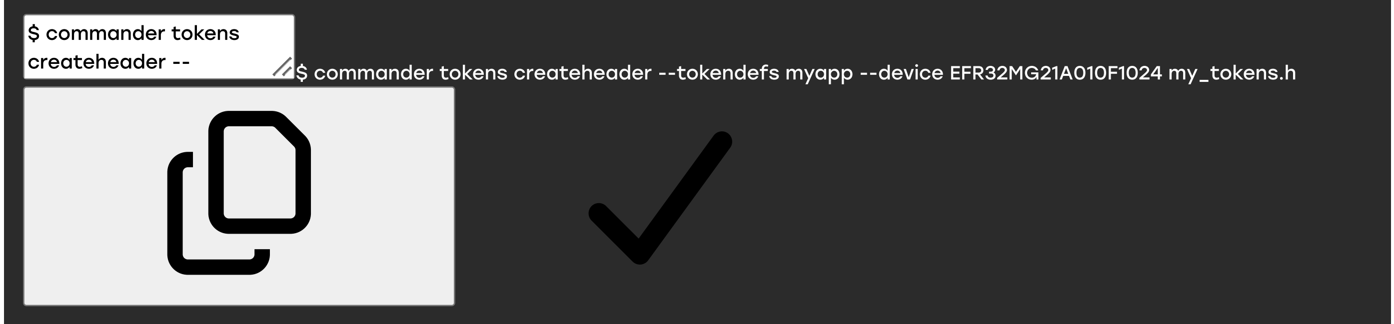
The `createheader` command generates a simple header file based on a custom token group. The generated header file contains pre-processor defines that specify the location and size of each token.

Copyright © 2026 Silicon Laboratories. All rights reserved.

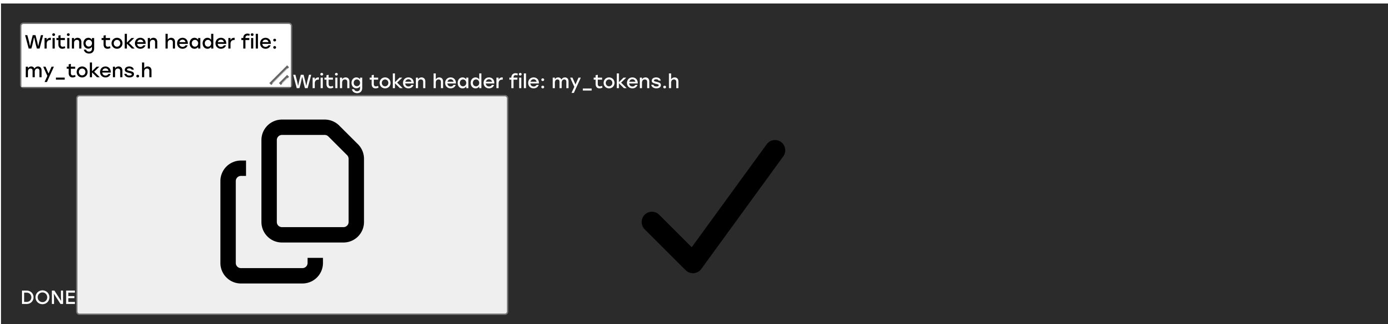
Command Line Syntax




Command Line Input Example




Command Line Output Example




## Custom Tokens

Custom tokens allow the user to define their own token groups and tokens for use with Simplicity Commander. This is useful for applications that require specific token management beyond the predefined groups.

For more information on how to create custom tokens, see the Custom Tokens documentation.

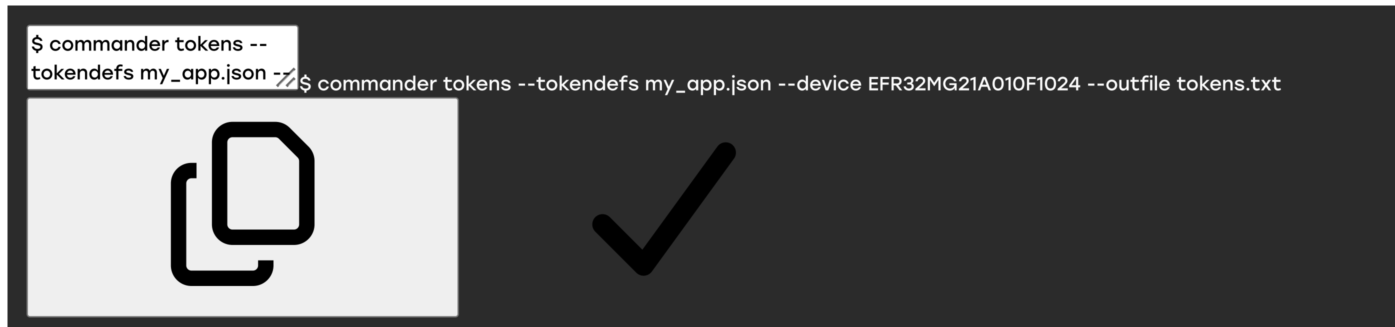
### Command Line Input Example (Static Tokens)

Copyright © 2026 Silicon Laboratories. All rights reserved.

Reads all static tokens defined in `my_app.json` from the device in the specified secure range and writes them to `tokens.txt` .

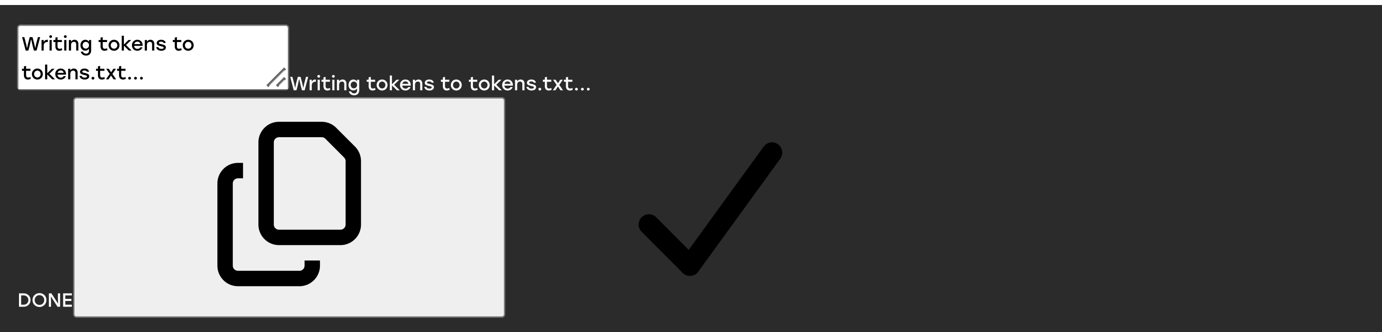
## Command Line Output Example

## Command Line Input Example (Manufacturing Tokens)




Reads all tokens defined in `my_app.json` from the device and writes them to `tokens.txt` .

## Command Line Output Example




Copyright © 2026 Silicon Laboratories. All rights reserved.

# Convert and Modify File Commands

# Convert and Modify File Commands

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

- Converting 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 Ember Bootloader (EBL) files. In this case, you must use the `--device` option.

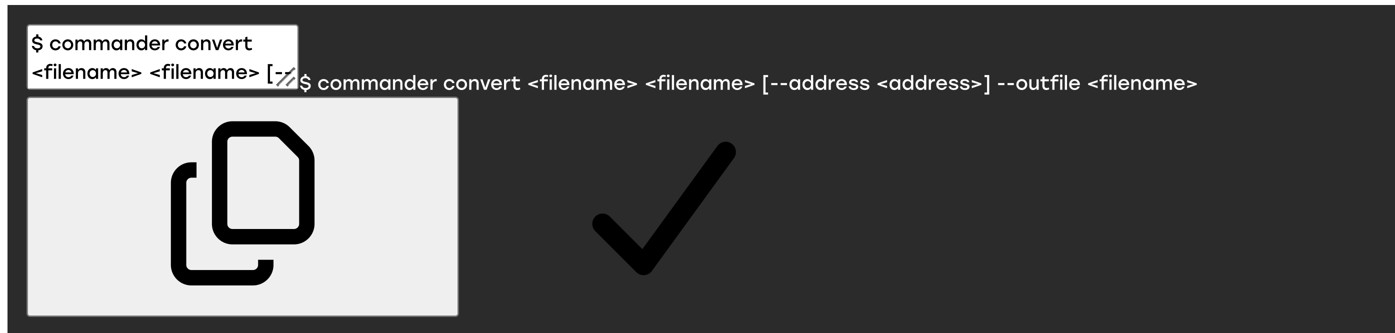
Command Line Syntax

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

## Combine Two Files

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

Command Line Syntax



```
$ commander convert &lt;filename&gt; &lt;filename&gt; [--address &lt;address&gt;] --outfile &lt;filename&gt;
```

Command Line Input Example

Copyright © 2026 Silicon Laboratories. All rights reserved.

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. The address value is interpreted as a hexadecimal number. 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

## Define Specific Bytes

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

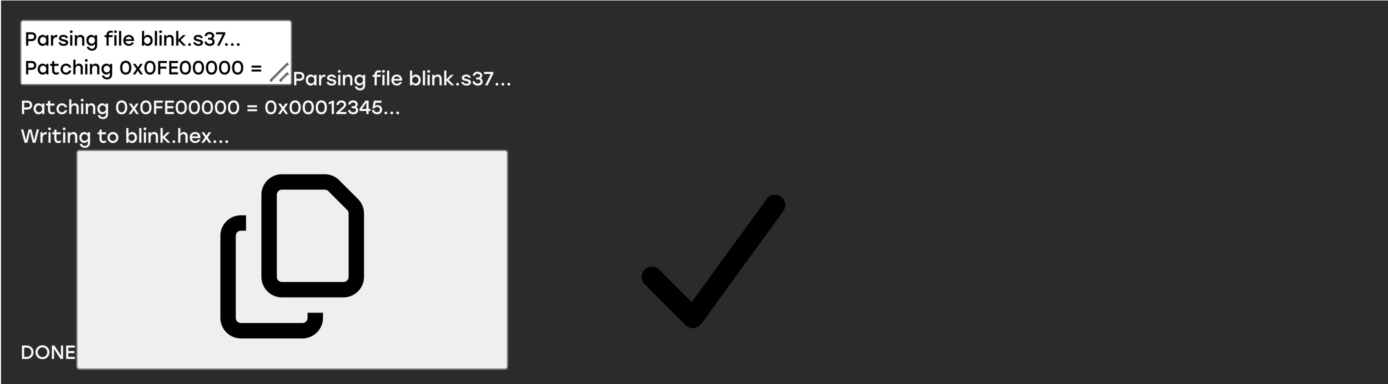
Command Line Syntax

Command Line Input Example

Copyright © 2026 Silicon Laboratories. All rights reserved.

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 0x0FE00000:0x``12345:4`, but works against a file instead of writing to a device flash.

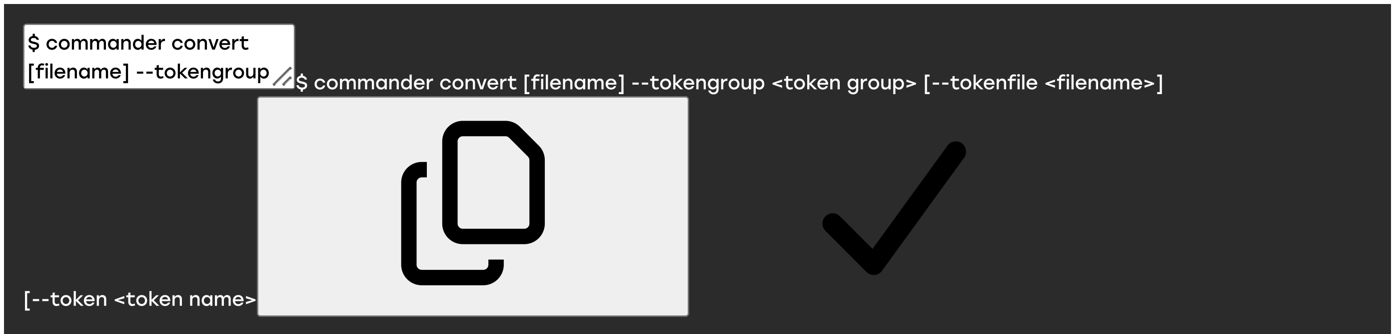
Command Line Output Example




## Define Tokens

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

Command Line Syntax




:<token data>] [--device <device>] [--outfile <filename>]

Command Line Input Example

Copyright © 2026 Silicon Laboratories. All rights reserved.

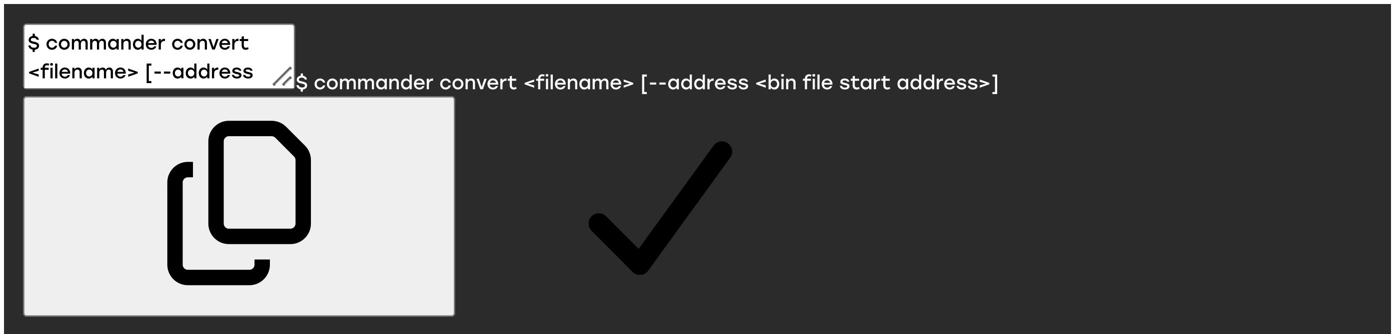
Converts blink.s37 to hex format, while simultaneously defining the tokens defined in tokens.txt and on the command line. Works just like the corresponding options with `flash`, but writes to a file instead of flash.

Command Line Output Example

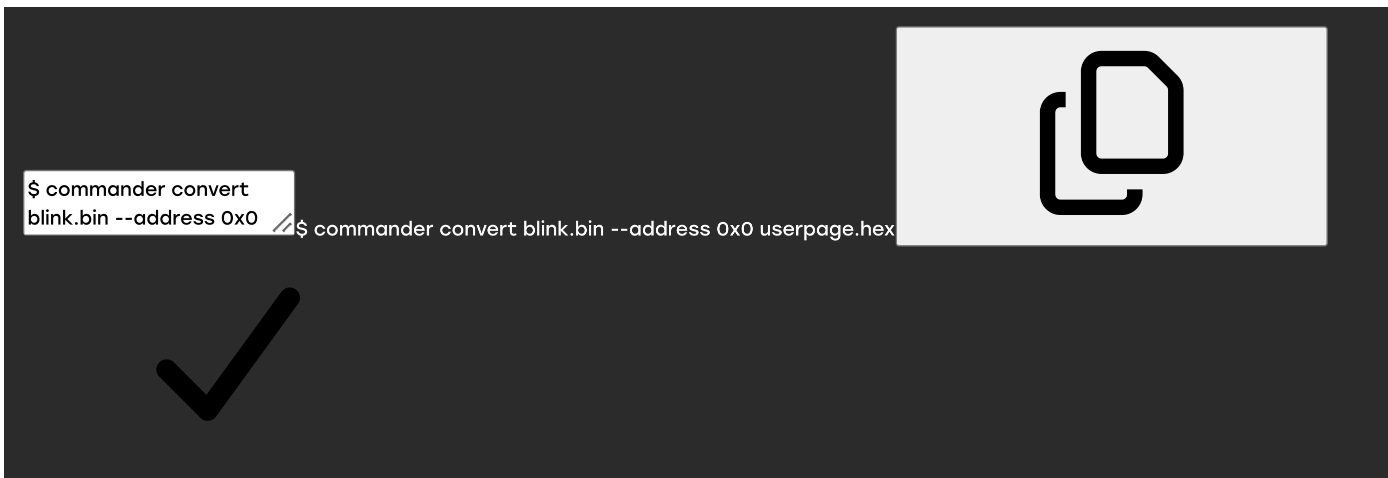
## Dump File Contents

Like the `readmem` command, the `convert` command will print its output in human-readable format to standard out if no output file is given. The value of the address option is interpreted as a hexadecimal number.

Command Line Syntax




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

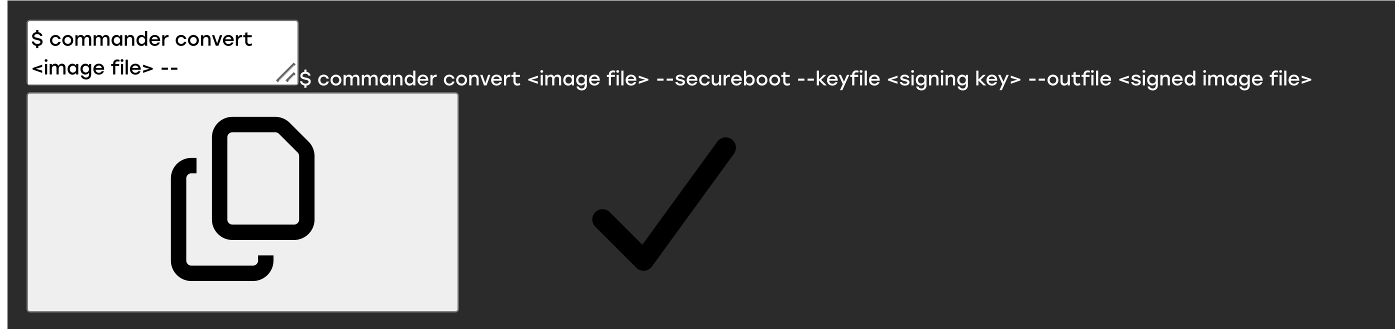
```
Parsing file blink.bin...
Parsing file userpage.hex...
{address:  0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F}
00000000: 10 04 00 20 B5 0A 00 00 57 08 00 00 8B 0A 00 00
00000010: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000020: 00 00 00 00 00 00 00 00 00 00 00 00 97 0A 00 00
00000030: 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
```

## Signing an Application for Secure Boot

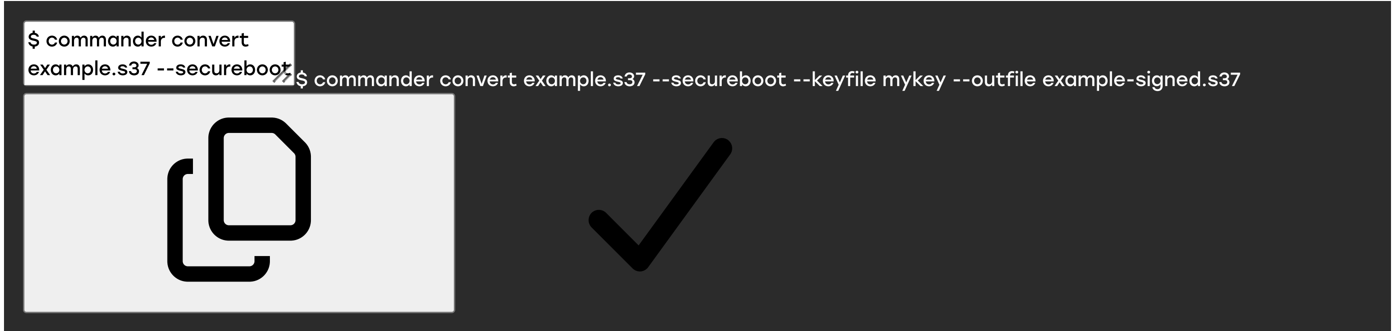
Signs an application for use with a Secure Boot bootloader. For more information, see *UG266: Silicon Labs Gecko Bootloader User's Guide for GSDK 3.x and Lower* or *UG489: Silicon Labs Gecko Bootloader User's Guide for GSDK 4.0 and Higher*.

Command Line Syntax

Copyright © 2026 Silicon Laboratories. All rights reserved.




Command Line Input Example




This example signs the image file named example.s37.

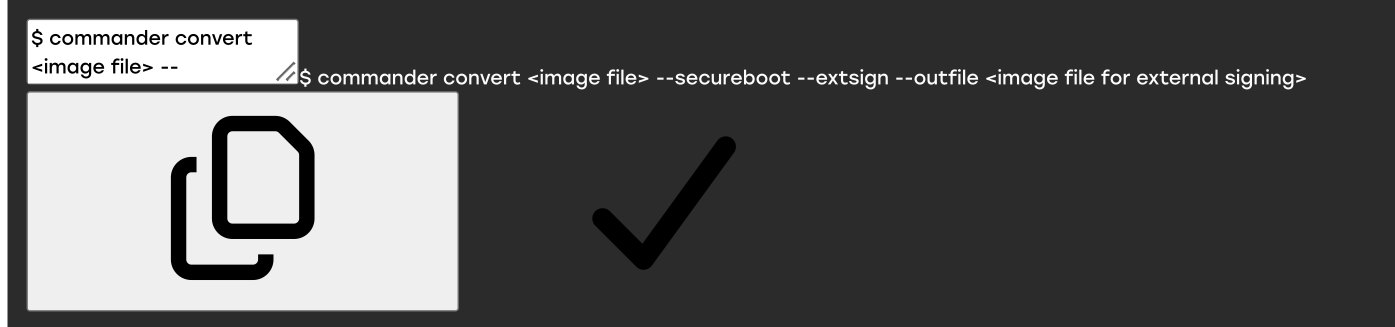
Command Line Output Example

## Signing an Application for Secure Boot using a Hardware Security Module

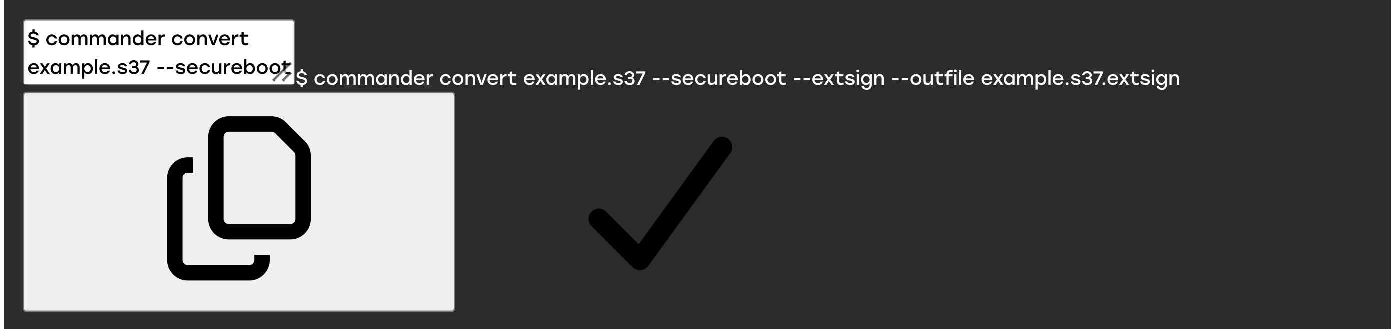
Prepares an application for signing for use with a Secure Boot enabled bootloader using a Hardware Security Module (HSM). For more information, see *UG266: Silicon Labs Gecko Bootloader User's Guide for GSDK 3.x and Lower* or *UG489: Silicon Labs Gecko Bootloader User's Guide for GSDK 4.0 and Higher*.

Command Line Syntax

Copyright © 2026 Silicon Laboratories. All rights reserved.

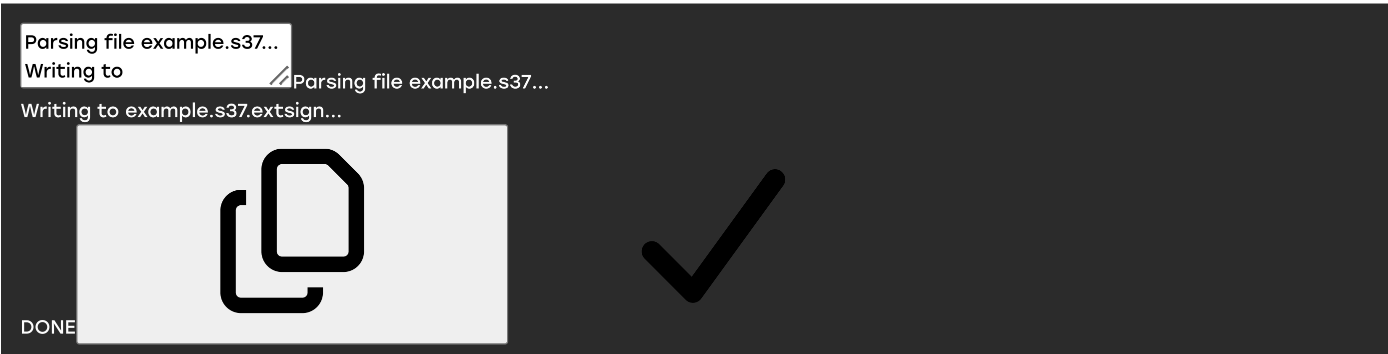



Command Line Input Example




This example creates an output in the form that an HSM can create a signature over of the entire file. This signature can again be written to the file using the command described in the section below.

Command Line Output Example



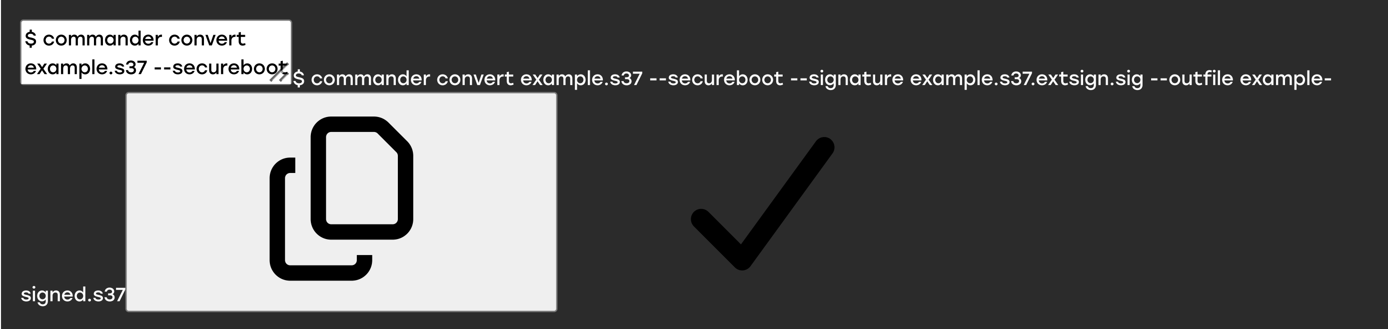

## Signing an Application for Secure Boot Signing using a Signature Created by a Hardware Security Module

Signs an application for use with a Secure Boot bootloader using a signature created by a Hardware Security Module (HSM). For more information, see *UG266: Silicon Labs Gecko Bootloader User's Guide for GSDK 3.x and Lower* or *UG489: Silicon Labs Gecko Bootloader User's Guide for GSDK 4.0 and Higher*.

Command Line Syntax

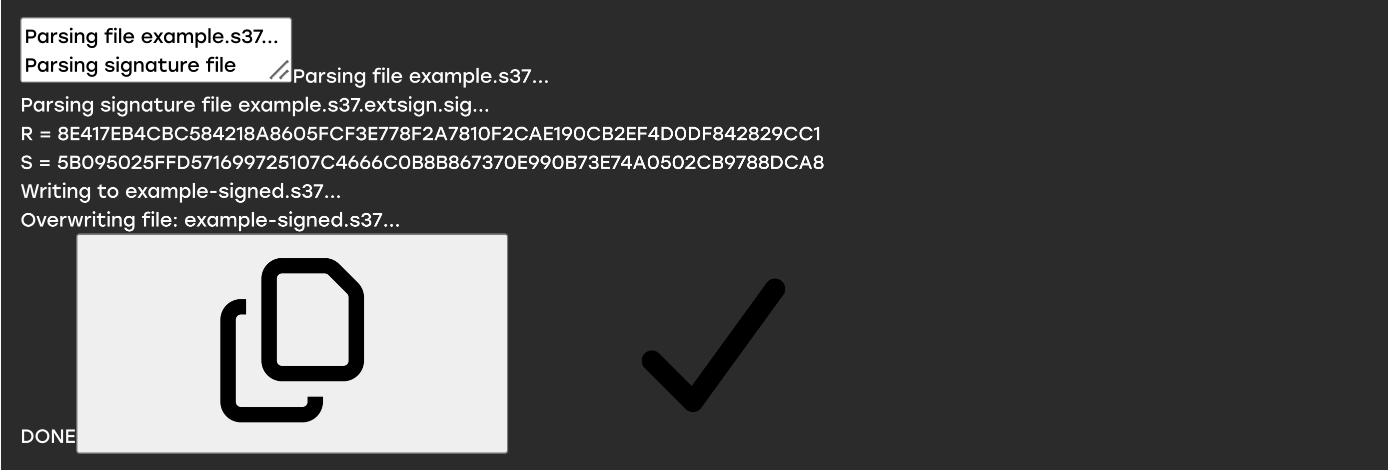
Copyright © 2026 Silicon Laboratories. All rights reserved.

Command Line Input Example




This example signs the image file example.s37 using a signature obtained from an HSM using the .extsign file generated in the section above. The input file (example.s37) used with this function must be the same file as was used when generating the .extsign file in the section above.

Command Line Output Example




## Adding a CRC32 for Gecko Bootloader

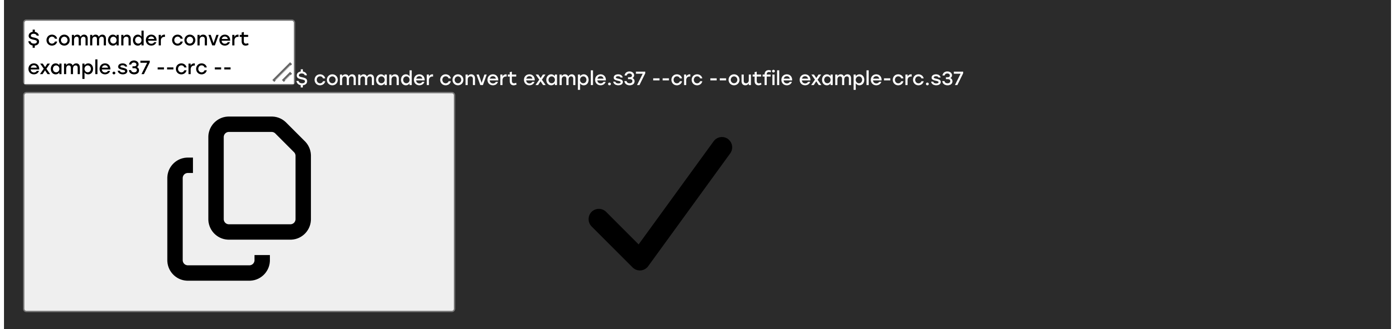
This option adds a CRC32 (32-bit cyclic redundancy check) of the image that the Gecko Bootloader can use to ensure image integrity when Secure Boot is not used. This feature requires that an `ApplicationProperties_t` struct is present in the image. For more details on the `ApplicationProperties_t` struct, see *UG266: Silicon Labs Gecko Bootloader User's Guide for GSDK 3.x and Lower* or *UG489: Silicon Labs Gecko Bootloader User's Guide for GSDK 4.0 and Higher*.

Command Line Syntax

Copyright © 2026 Silicon Laboratories. All rights reserved.

```
$ commander convert <image file> --crc --outfile <image file with CRC>
```

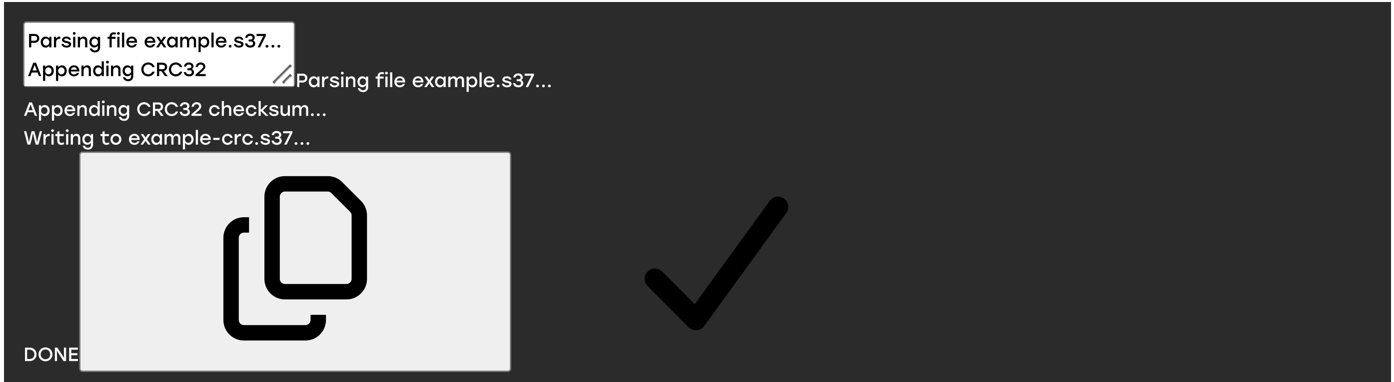
Command Line Input Example



```
$ commander convert example.s37 --crc --outfile example-crc.s37
```

This example adds a checksum to the image file named example.s37.

Command Line Output Example



```
Parsing file example.s37...
Appending CRC32 checksum...
Writing to example-crc.s37...
DONE
```

## Signing an Application for Secure Boot using an Intermediary Certificate

Signs an application for use with a Secure Boot bootloader using an intermediary certificate. When using an intermediary certificate, the `ApplicationProperties_t` struct must be present in the image. For more information on the `ApplicationProperties_t` struct, see *UG266: Silicon Labs Gecko Bootloader User's Guide for GSDK 3.x and Lower* or *UG489: Silicon Labs Gecko Bootloader User's Guide for GSDK 4.0 and Higher*.

Secure Boot verification via an intermediary certificate is only supported on Series 2 EFR32 devices. Secure Boot must be enabled before signing a bootloader with an intermediary certificate. For more information about enabling Secure Boot, see Write User Configuration.

There are two ways of signing the application:

- Providing the private keyfile corresponding to the public key embedded in the certificate directly.
- Preparing an application for signing with a Hardware Security Module (HSM) by generating an output in the form that an HSM can create a signature over the entire file. The signature can then be written to the file by passing it to Simplicity Commander as described below.

Copyright © 2026 Silicon Laboratories. All rights reserved.

A suitable certificate can be generated and signed using the `util gencert` command - see Generate Certificate for details.

Command Line Syntax

```
$ commander convert
<image file> --
```

```
$ commander convert <image file> --secureboot --certificate <certificate file> --keyfile <keyfile> --
outfile <signed image file>
```

```
$ commander convert
<image file> --
```

```
$ commander convert <image file> --secureboot --certificate <certificate file> --extsign --outfile <image
file for external signing>
```

```
$ commander convert
<image file> --
```

```
$ commander convert <image file> --secureboot --certificate <certificate file> --signature <signature> --
outfile <signed image file>
```

Command Line Input Example

Copyright © 2026 Silicon Laboratories. All rights reserved.

This example signs the image file example.s37 using an intermediary certificate. The keyfile used to sign the application corresponds to the public key embedded in the certificate. Simplicity Commander always validates the key before signing the application.

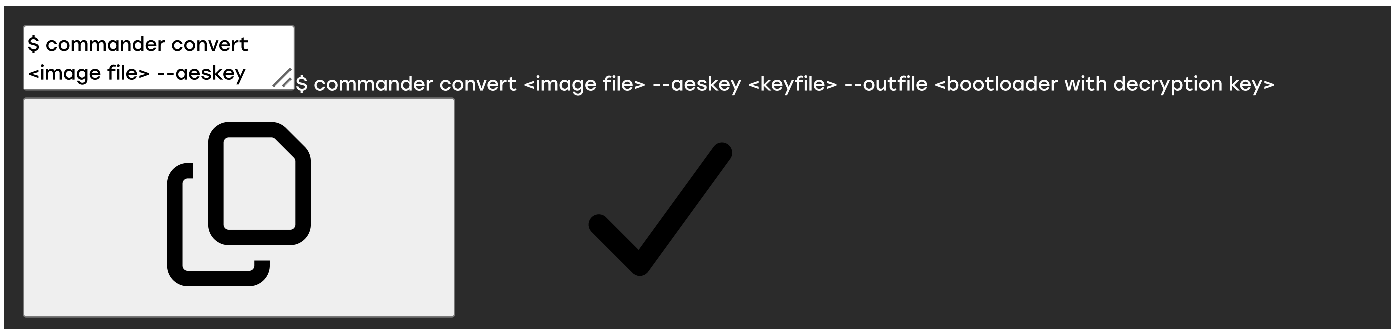
Command Line Output Example

```
Parsing file example.s37...
Private key matches public key in certificate.
R = 137EA7A19F6100E1EFA5C185CA952B67137D0597F4A89C7543BC5A49A7A6681E
S = C537A833018C3A23CF1EBDBAB04559482B0B5333A7C21556E6B42EDA1D1A5102
Writing to example-signed.s37...
DONE
```

## Add a Trust Zone Decryption Key

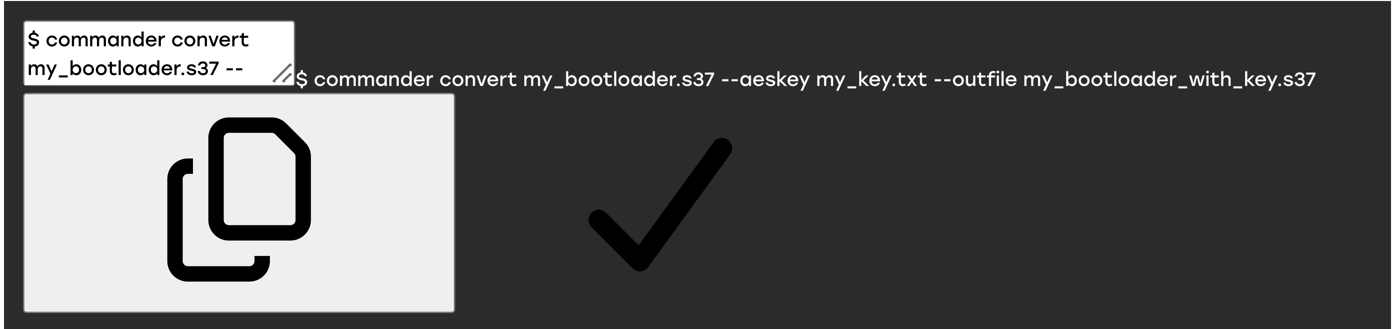
Adds an Advanced Encryption Standard (AES) encryption/decryption key to a bootloader image for TrustZone secure key storage. Requires Application Properties struct version 1.2 or higher.

Command Line Syntax



```
$ commander convert <image file> --aeskey <keyfile> --outfile <bootloader with decryption key>
```

Command Line Input Example

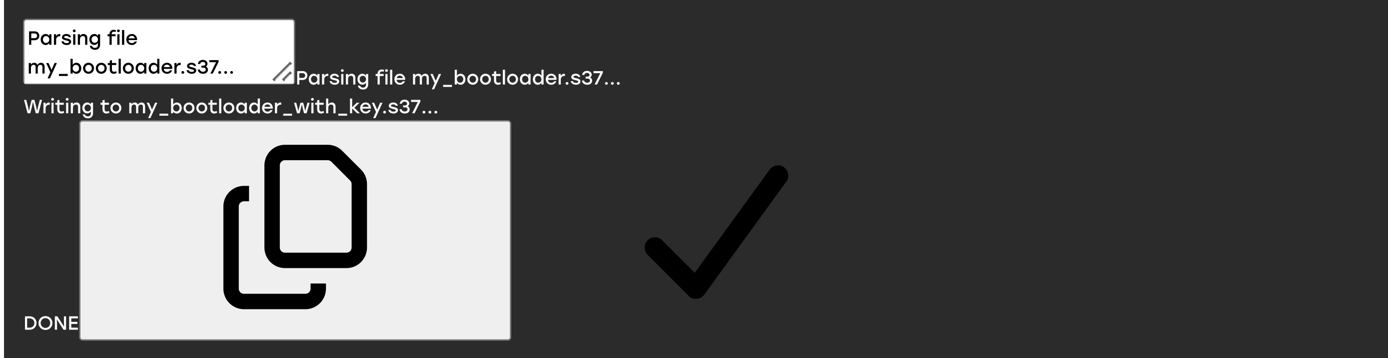


```
$ commander convert my_bootloader.s37 --aeskey my_key.txt --outfile my_bootloader_with_key.s37
```

Adds the decryption key *my_key.txt* to the bootloader image named *my_bootloader_with_key.s37*.

Command Line Output Example

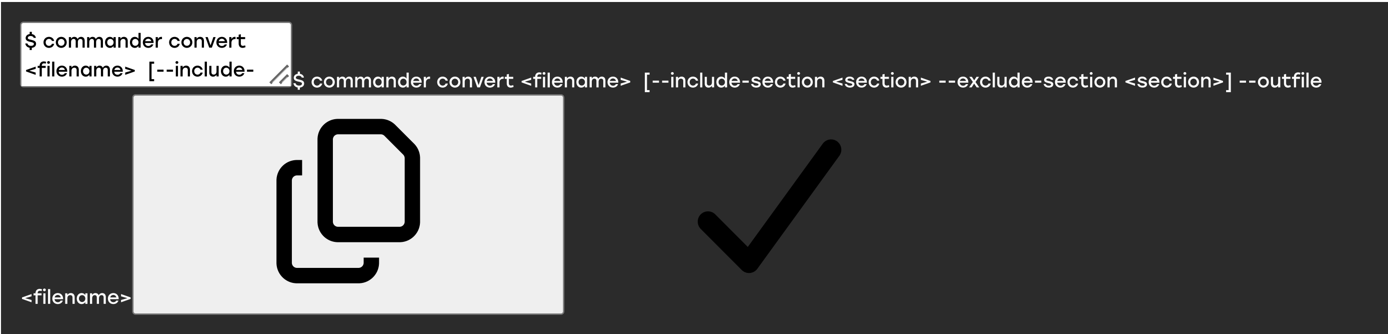
Copyright © 2026 Silicon Laboratories. All rights reserved.



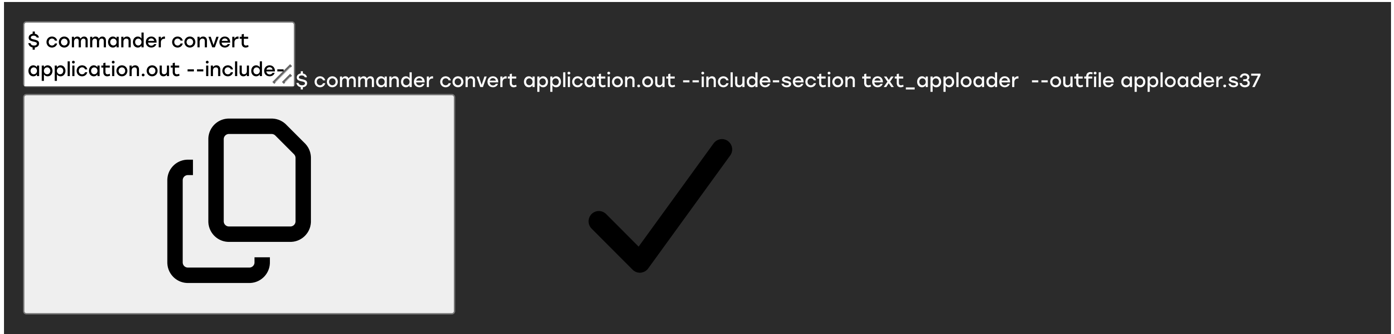

## Extract Sections from ELF Files

Extract sections from an Executable and Linkable Format (ELF) file and convert into the specified output file. If neither the `--include-section` nor the `--exclude-section` option is used, Simplicity Commander will extract all `.text` sections, as well as sections of type `progbits` with address not equal to 0x0.

Command Line Syntax

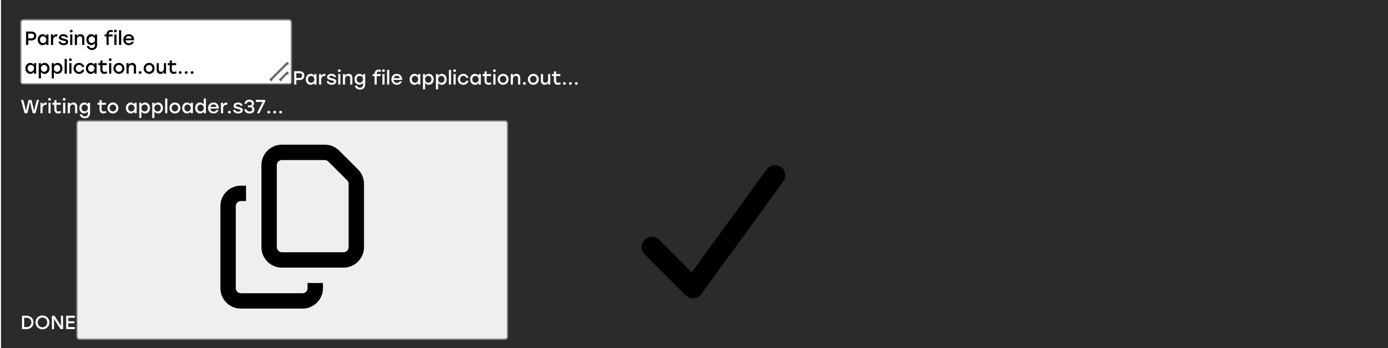



Command Line Input Example




Creates an S-record file from the `text_apploader` section of an ELF file.

Command Line Output Example




Copyright © 2026 Silicon Laboratories. All rights reserved.

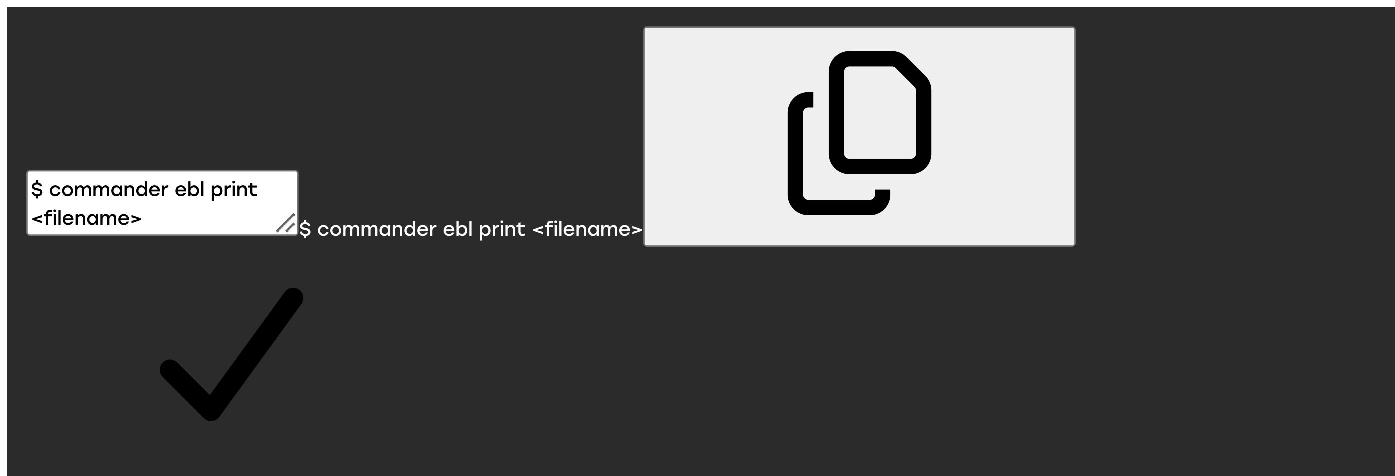
# EBL Commands

# EBL Commands

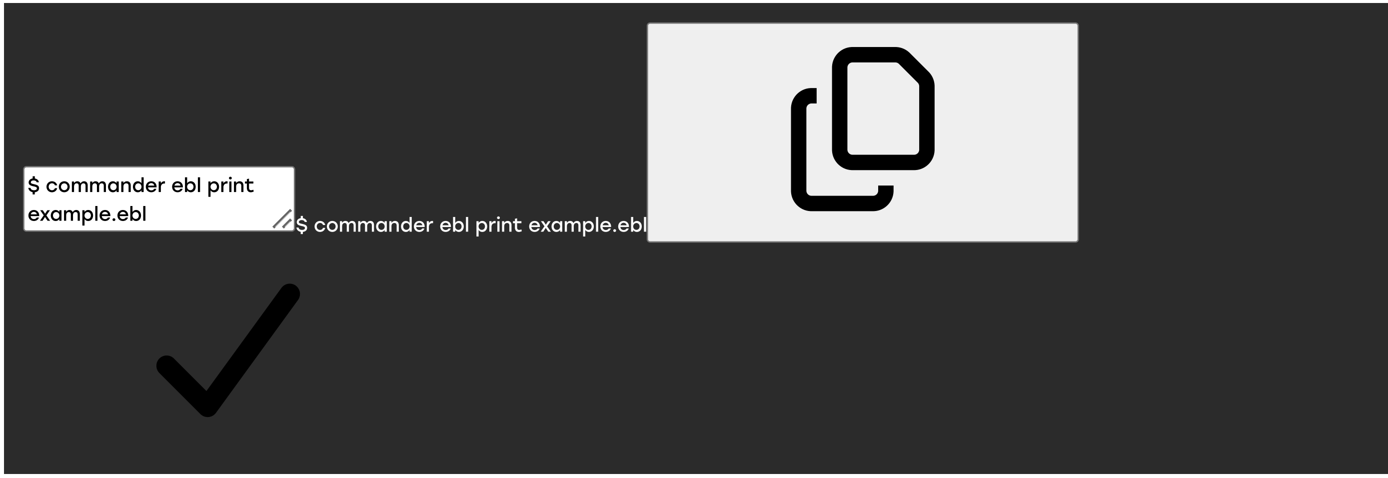
## Print EBL Information

Parses and prints EBL information from the specified .ebl file.

Command Line Syntax




Command Line Input Example




Command Line Output Example

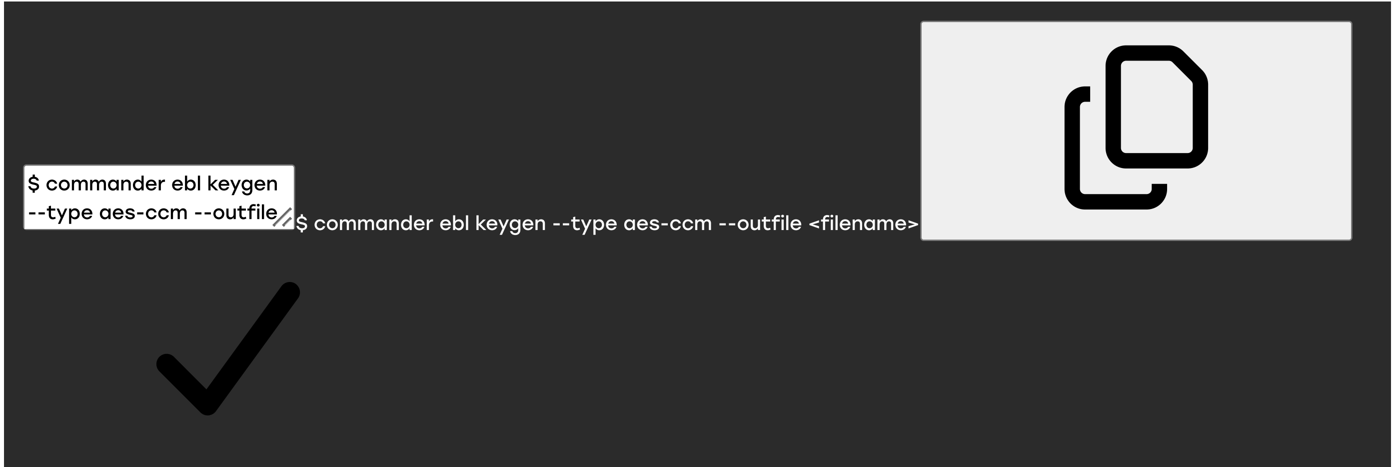
Copyright © 2026 Silicon Laboratories. All rights reserved.

```
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
```

## EBL Key Generation

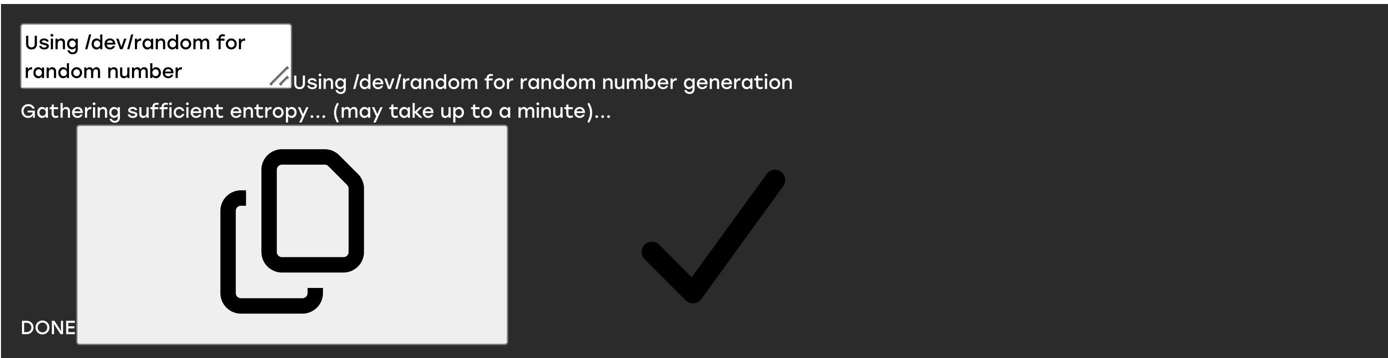
Generates a keyfile to be used for encryption or decryption and outputs the keyfile to the specified filename.

Command Line Syntax




Command Line Input Example

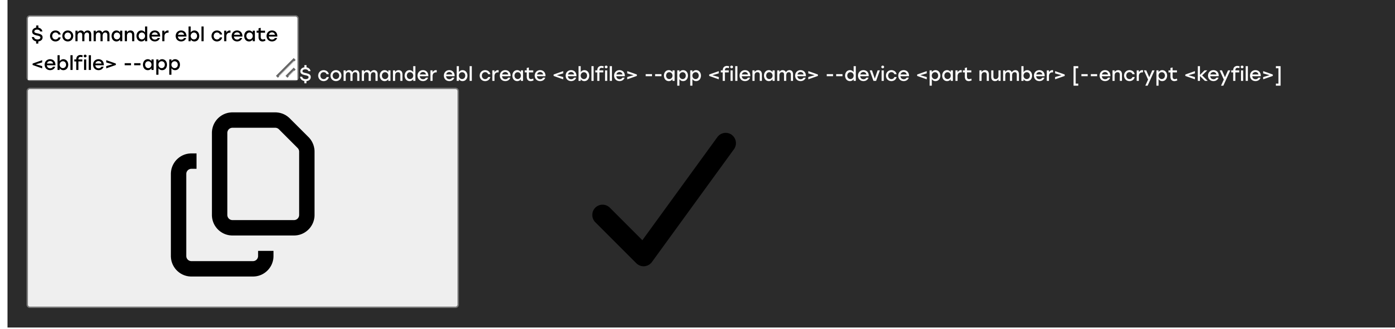
Command Line Output Example




## EBL File Creation

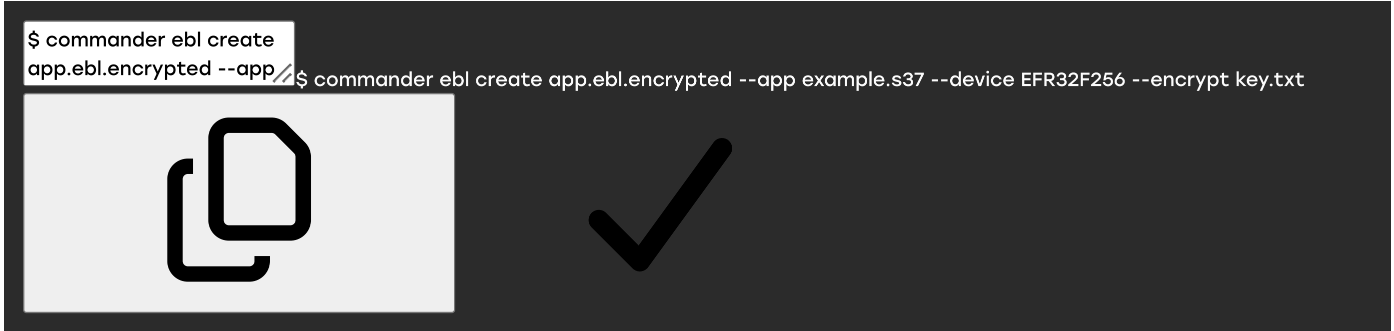
Creates an EBL file from an application image and writes the output to the specified filename. Can optionally encrypt the EBL file using a keyfile generated by the `ebl keygen` command.

Command Line Syntax



```
$ commander ebl create <eblfile> --app <filename> --device <part number> [--encrypt <keyfile>]
```

Command Line Input Example



```
$ commander ebl create app.ebl.encrypted --app example.s37 --device EFR32F256 --encrypt key.txt
```

Command Line Output Example

```
Parsing file example  .s37...
Parse .s37 format for flash
Flash Usage:
  Reserved for Bootloader:          0x00000000-0x00003fff (16384 bytes)
  CODE and Tables:                 0x00004000-0x00014ddb (69084 bytes)
  CONST and INITC:                 0x00014ddc-0x000184ab (14032 bytes)
  Available for future use:          0x000184ac-0x0003dfff (154452 bytes)
  Reserved for SIMEE:               0x0003e000-0x0003ffff (8192 bytes)
Usage Summary:
  262144 total bytes Flash, 107692 used, 154452 available
Setting AAT timestamp to current time: 0x586e1ec9
Create ebl image file
Wrote image stamp into AAT.
Encrypting EBL...
Unencrypted input file: ebl_plaintext_ux8544.ebl
Encrypt output file:     app.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
```

## EBL File Parsing

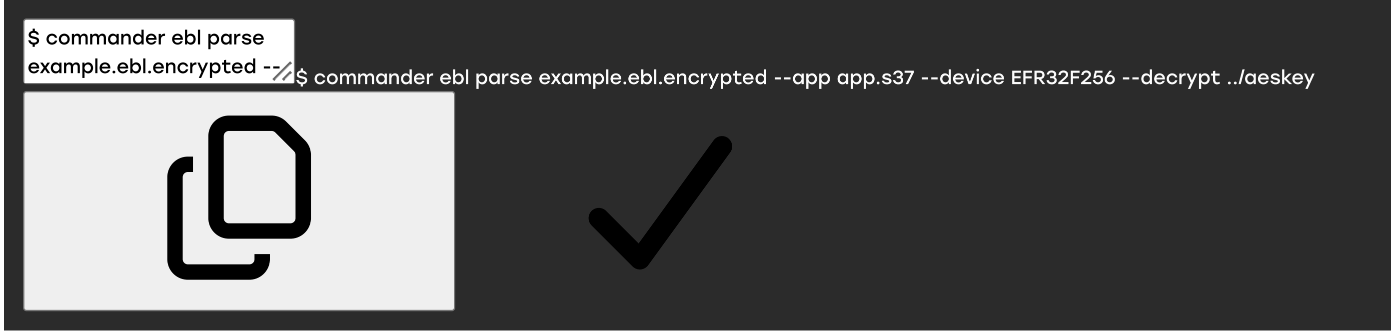
Parses an EBL file and writes the application image to the specified filename. Optionally decrypts an encrypted EBL file. The keyfile must be the same as was used for encrypting the encrypted EBL file.

Copyright © 2026 Silicon Laboratories. All rights reserved.

Command Line Syntax

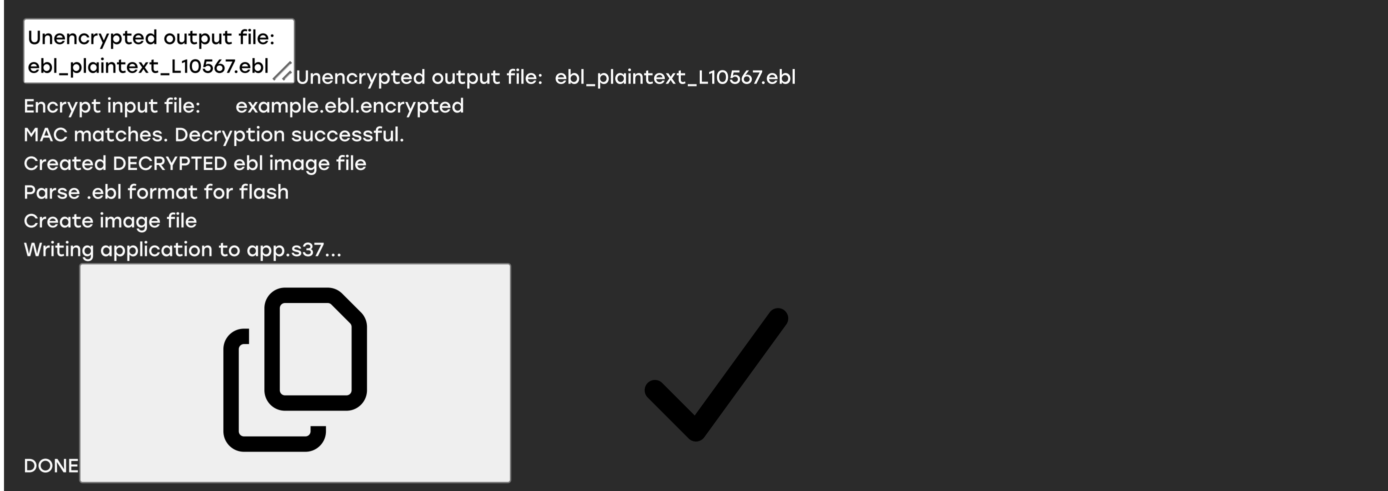
```
$ commander ebl parse <ebl filename> --app < filename> --device <part number> [--decrypt <key filename>]
```

Command Line Input Example



```
$ commander ebl parse example.ebl.encrypted --app app.s37 --device EFR32F256 --decrypt ../aeskey
```

Command Line Output Example



```
Unencrypted output file:  ebl_plaintext_L10567.ebl
Encrypt input file:      example.ebl.encrypted
MAC matches. Decryption successful.
Created DECRYPTED ebl image file
Parse .ebl format for flash
Create image file
Writing application to app.s37...
DONE
```

## Memory Usage Information from AAT

For applications containing an Application Address Table (AAT), Simplicity Commander can analyze the memory usage of the application. The AAT is included in Zigbee applications.

RAM usage is only available for EM3xx applications. Applications built for EFR32 can only be analyzed for flash usage.

Command Line Syntax

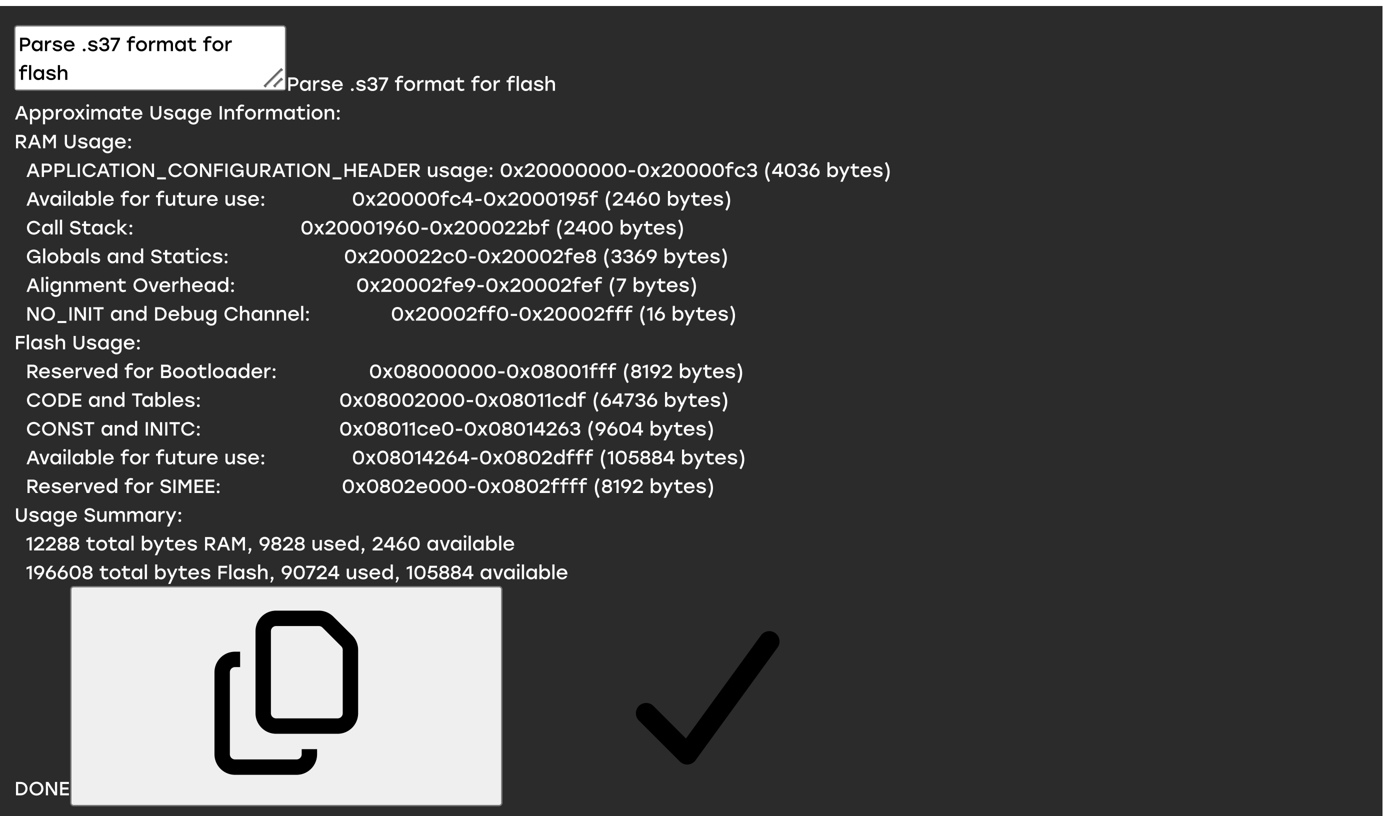
Copyright © 2026 Silicon Laboratories. All rights reserved.

```
$ commander ebl aat-usageinfo <filename> --device <part number>
```

Command Line Input Example

```
$ commander ebl aat-usageinfo example.s37 --device EM357
```

Command Line Output Example



```
Parse .s37 format for flash
Approximate Usage Information:
RAM Usage:
  APPLICATION_CONFIGURATION_HEADER usage: 0x20000000-0x20000fc3 (4036 bytes)
  Available for future use:              0x20000fc4-0x2000195f (2460 bytes)
  Call Stack:                            0x20001960-0x200022bf (2400 bytes)
  Globals and Statics:                   0x200022c0-0x20002fe8 (3369 bytes)
  Alignment Overhead:                    0x20002fe9-0x20002fef (7 bytes)
  NO_INIT and Debug Channel:             0x20002ff0-0x20002fff (16 bytes)
Flash Usage:
  Reserved for Bootloader:               0x08000000-0x08001fff (8192 bytes)
  CODE and Tables:                       0x08002000-0x08011cdf (64736 bytes)
  CONST and INITC:                       0x08011ce0-0x08014263 (9604 bytes)
  Available for future use:              0x08014264-0x0802dfff (105884 bytes)
  Reserved for SIMEE:                    0x0802e000-0x0802ffff (8192 bytes)
Usage Summary:
  12288 total bytes RAM, 9828 used, 2460 available
  196608 total bytes Flash, 90724 used, 105884 available
DONE
```

Copyright © 2026 Silicon Laboratories. All rights reserved.

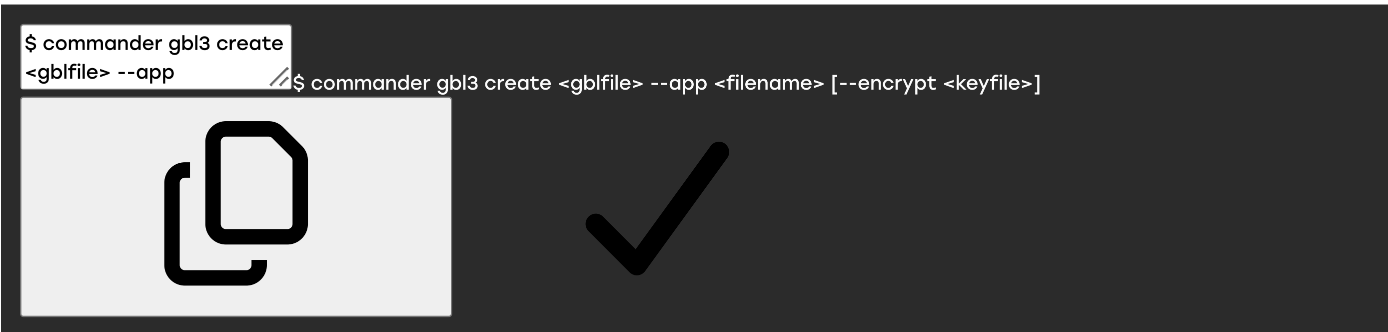
GBL3 Commands

# GBL3 Commands

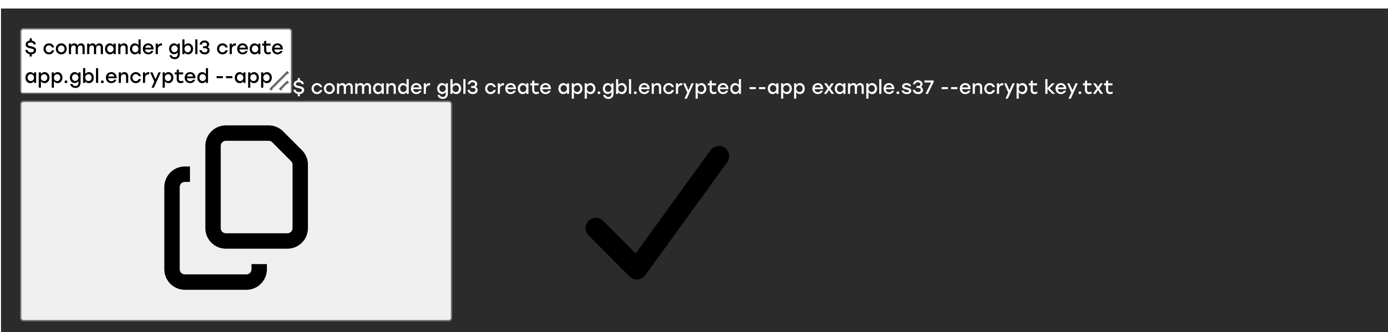
## GBL3 File Creation

Creates a Gecko Bootloader version 3 (GBL3) file from an application image and writes the output to the specified filename. Can optionally encrypt the GBL3 file using a keyfile generated by the `util genkey` command.

Command Line Syntax




Command Line Input Example




Command Line Output Example

## GBL3 File Creation with Compression

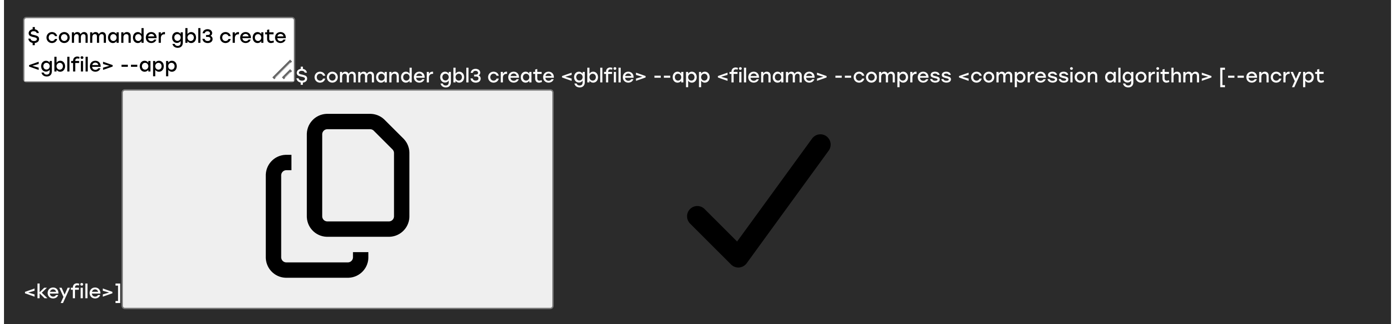
Creates a compressed Gecko Bootloader version 3 (GBL3) file from an application image and writes the output to the specified filename. Can optionally encrypt the GBL3 file using a keyfile generated by the `util`

Copyright © 2026 Silicon Laboratories. All rights reserved.

`genkey` command.

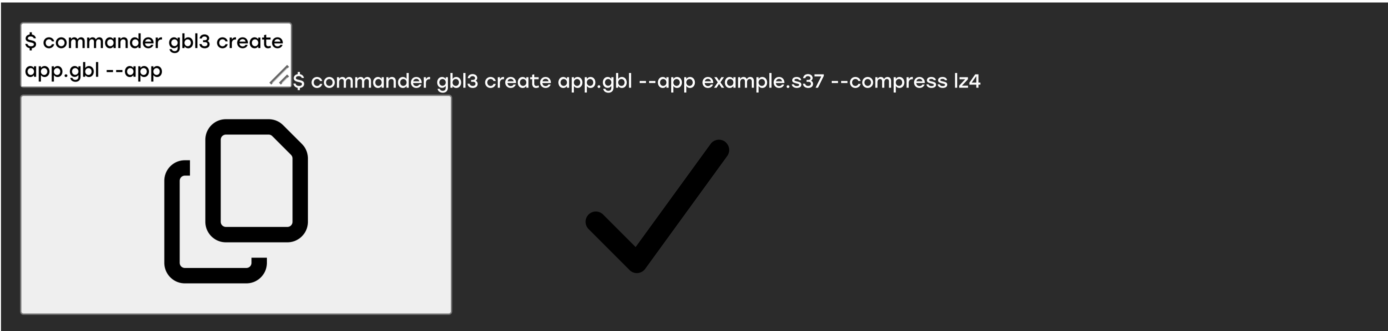
The currently supported compression algorithms are `lz4` and `lzma`. The bootloader on the targeted devices must support decompressing the selected compression type.

Command Line Syntax



```
$ commander gbl3 create <gblfile> --app <filename> --compress <compression algorithm> [--encrypt <keyfile>]
```

Command Line Input Example



```
$ commander gbl3 create app.gbl --app example.s37 --compress lz4
```

Command Line Output Example

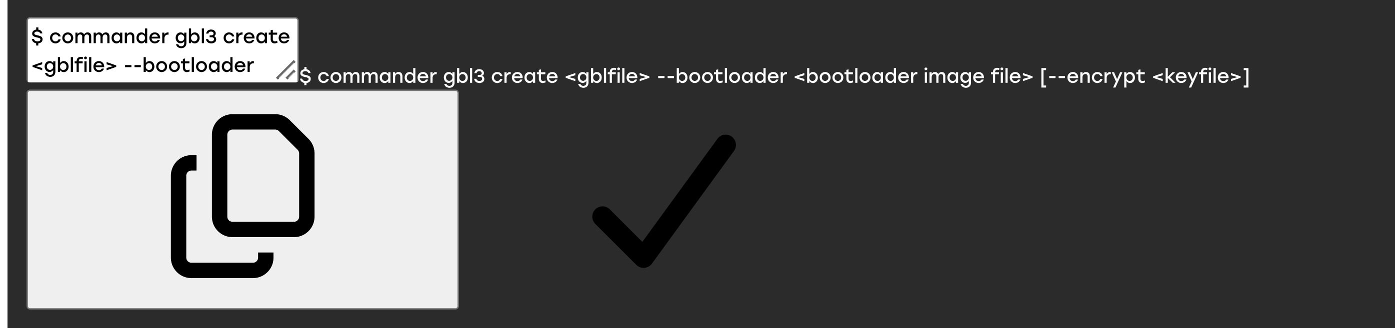
```
Parsing file example.s37...
Initializing GBL file...
Adding application to GBL...
Compressing using lz4...
Writing GBL file app.gbl...
DONE
```

## Create a GBL3 File for Bootloader Upgrade

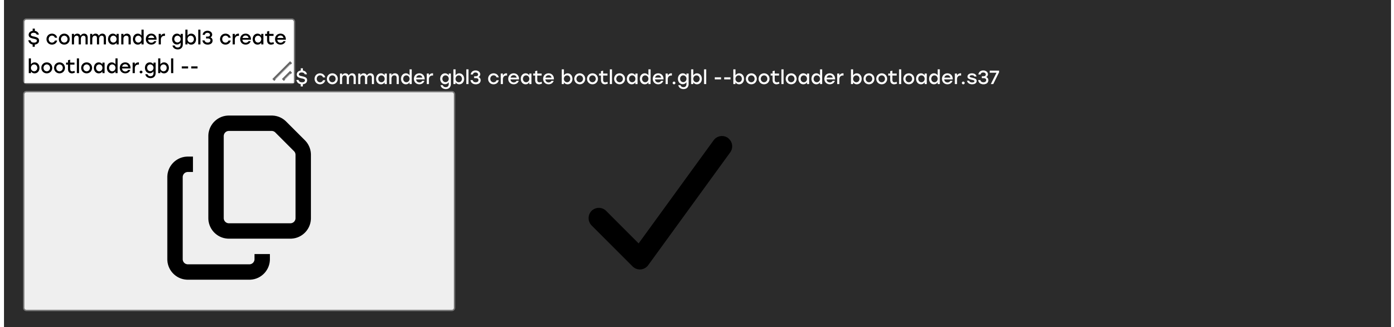
Creates a Gecko Bootloader version 3 (GBL3) file from a bootloader image and writes the output to the specified bootloader image filename. For more information, see *UG266: Silicon Labs Gecko Bootloader User's Guide for GSDK 3.x and Lower* or *UG489: Silicon Labs Gecko Bootloader User's Guide for GSDK 4.0 and Higher*.

Command Line Syntax

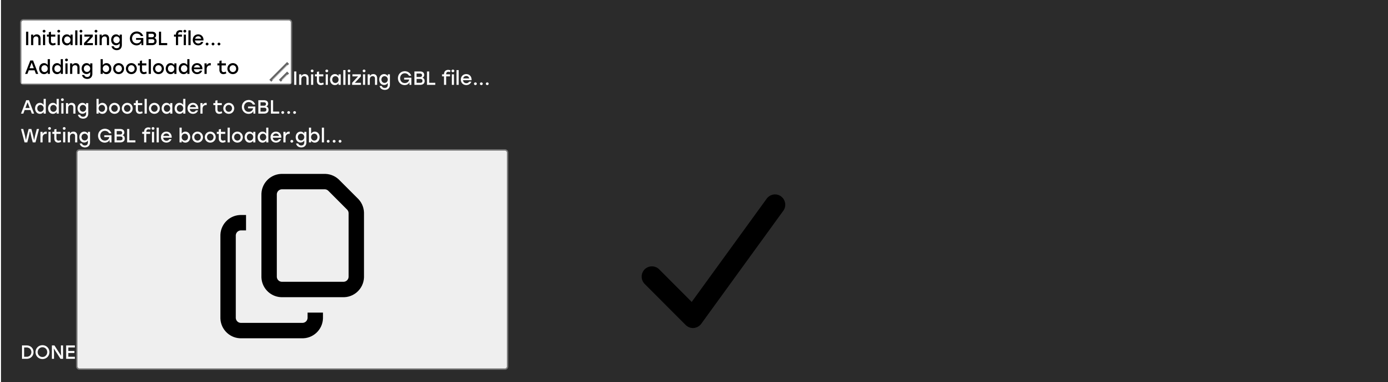
Copyright © 2026 Silicon Laboratories. All rights reserved.




Command Line Input Example



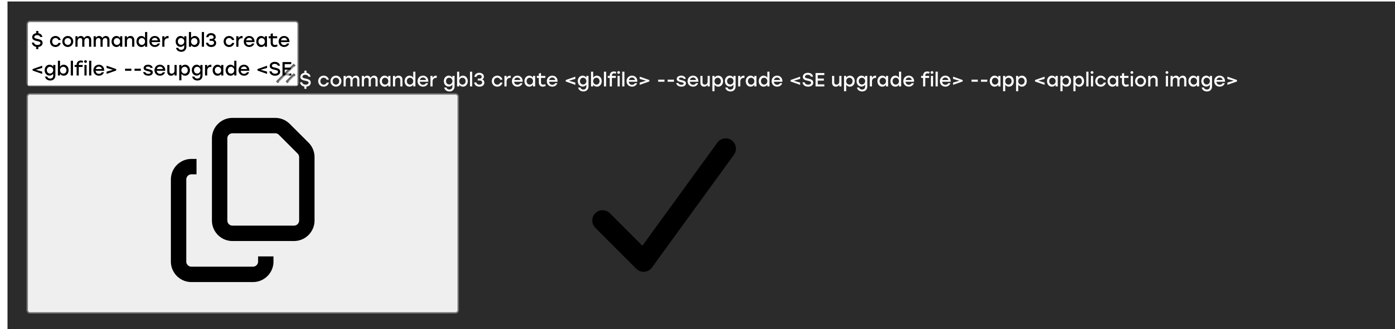

Command Line Output Example




## Creating a GBL3 File for Secure Engine Upgrade

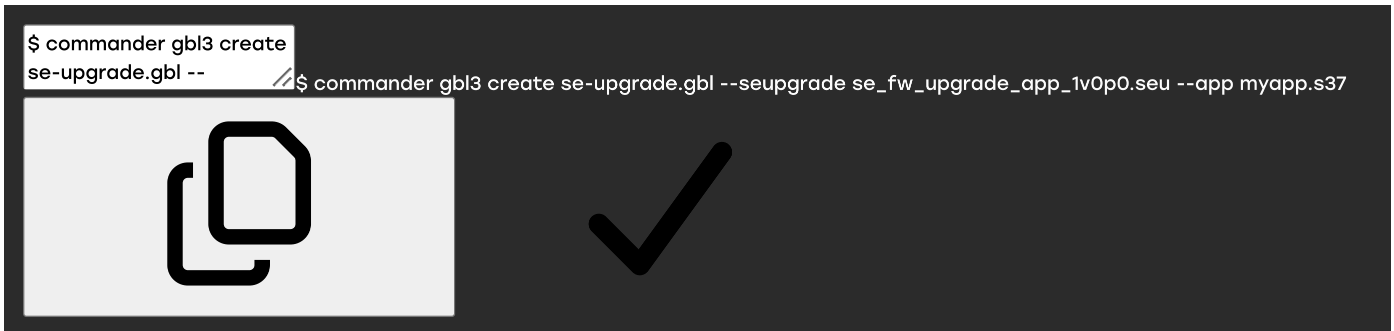
The Secure Engine on EFR32xG21 devices can be upgraded using a Secure Engine upgrade binary provided by Silicon Labs. This command creates a GBL3 file containing a Secure Engine upgrade file and writes the output to the specified GBL3 filename. For more information, see *UG266: Silicon Labs Gecko Bootloader User's Guide for GSDK 3.x and Lower* or *UG489: Silicon Labs Gecko Bootloader User's Guide for GSDK 4.0 and Higher*.

Command Line Syntax

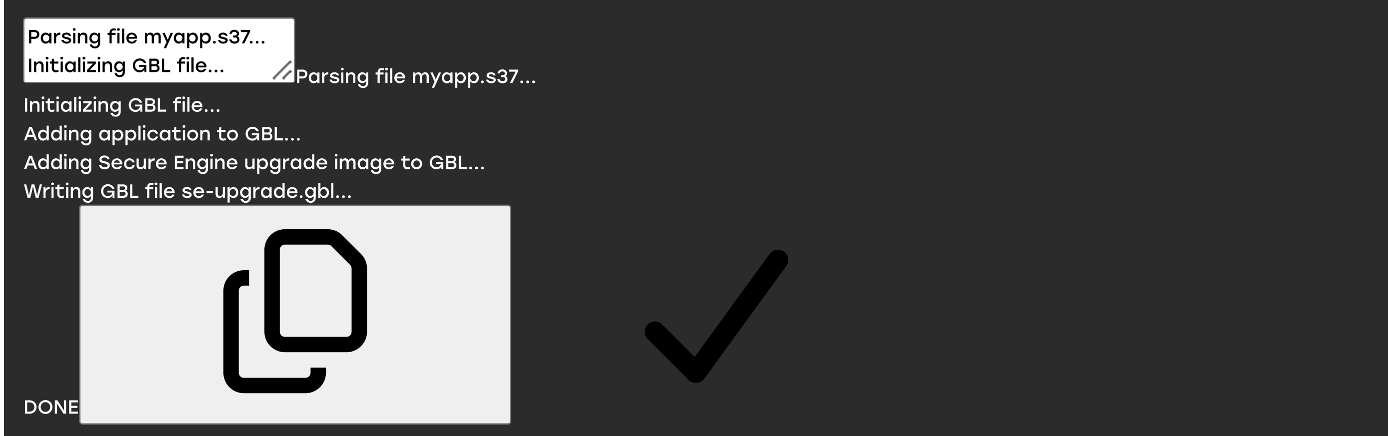



Copyright © 2026 Silicon Laboratories. All rights reserved.

Command Line Input Example



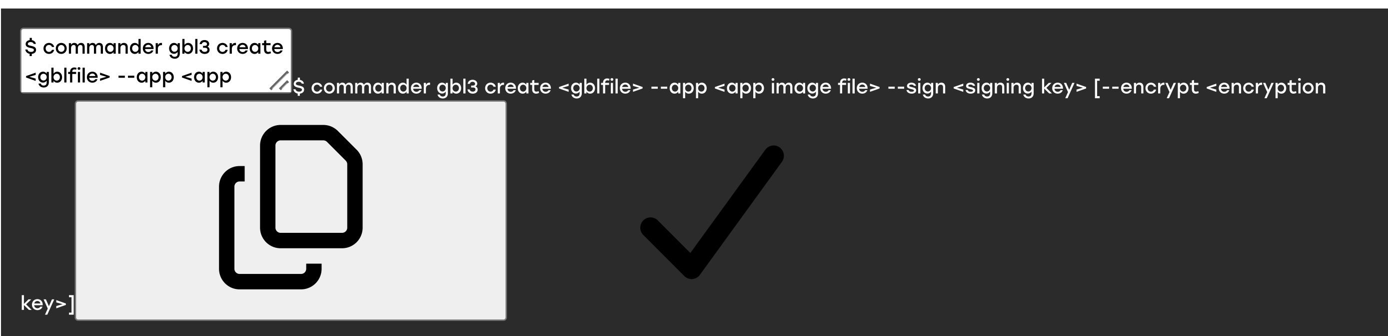

Command Line Output Example




## Creating a Signed and Encrypted GBL3 Upgrade Image File from an Application

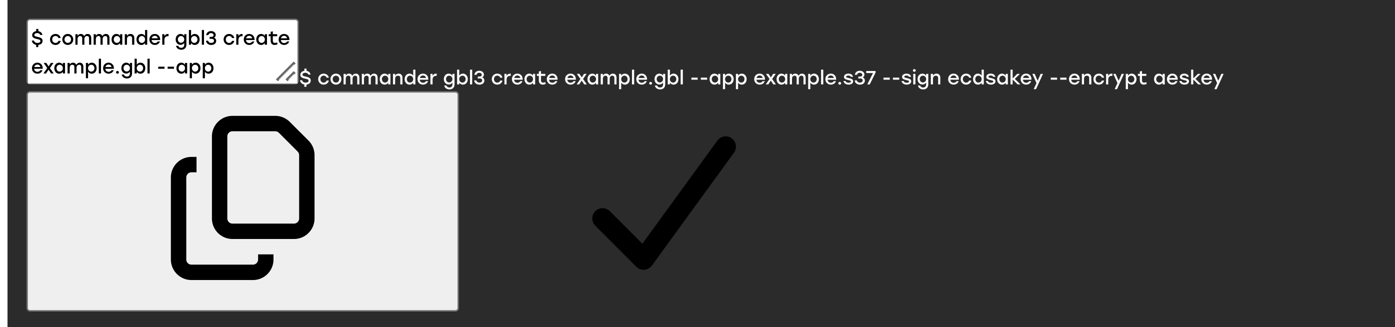
Creates a GBL3 file, signs the GBL3 file, and encrypts the GBL3 file. For more information, see *UG266: Silicon Labs Gecko Bootloader User's Guide for GSDK 3.x and Lower* or *UG489: Silicon Labs Gecko Bootloader User's Guide for GSDK 4.0 and Higher*.

Command Line Syntax

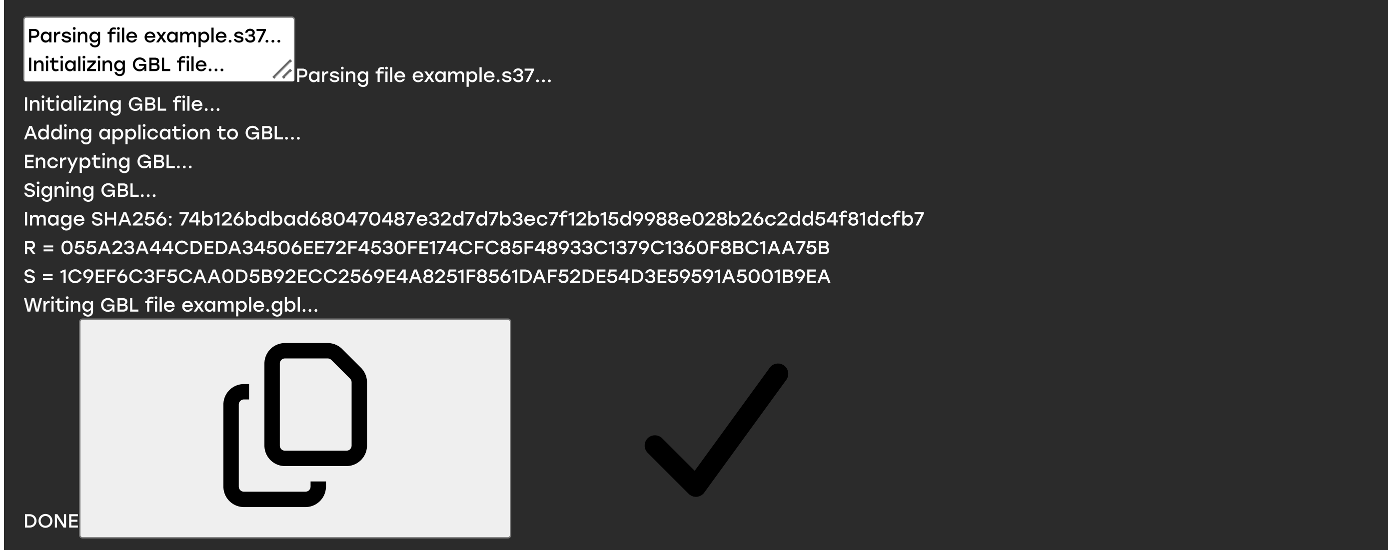



Command Line Input Example

Copyright © 2026 Silicon Laboratories. All rights reserved.




Command Line Output Example




## Preparing a GBL3 Upgrade File for Use with a Hardware Security Module

It is often not desirable to keep the private keys used for signing/encrypting locally on the computer that creates the GBL3 images. A good way to increase security is to use a Hardware Security Module (HSM) to encrypt the GBL3 data and generate the actual signatures.

Simplicity Commander supports both external encryption and external signing of GBL3 files. To enable external encryption, specify the `--extencrypt` option when running the `gbl3 create` command. To enable external signing, use the `--extsign option`. You can use the options together to create a GBL3 file that is encrypted and signed externally. The following sections describe a generic workflow for enabling external encryption, external signing, or both.

1. Prepare the GBL3 file for external encryption and/or signing using Simplicity Commander.
2. If external encryption is used:
   a. Encrypt the relevant data using an HSM.
   b. Use Simplicity Commander to assemble the encrypted data into an encrypted GBL3 file.
3. If external signing is used:
   a. Create an Elliptic Curve Digital Signature Algorithm (ECDSA) signature of the relevant data using an HSM.
   b. Use Simplicity Commander to sign the partial GBL3 file using the signature from the HSM, completing the GBL3 file.

Step 1 is described in Preparing a GBL3 File for External Encryption and Preparing a GBL3 File for External Signing. Steps 2a and 3a are specific to the HSM you are using. Step 2b is described in Completing an Encrypted GBL3 Using a Hardware Security Module, and step 3b is described in Completing a Signed GBL3 File Using a Hardware Security Module. A walkthrough of the process of both externally encrypting and externally signing is provided in Creating a Signed and Encrypted GBL3 File Using a Hardware Security Module. For more

Copyright © 2026 Silicon Laboratories. All rights reserved.

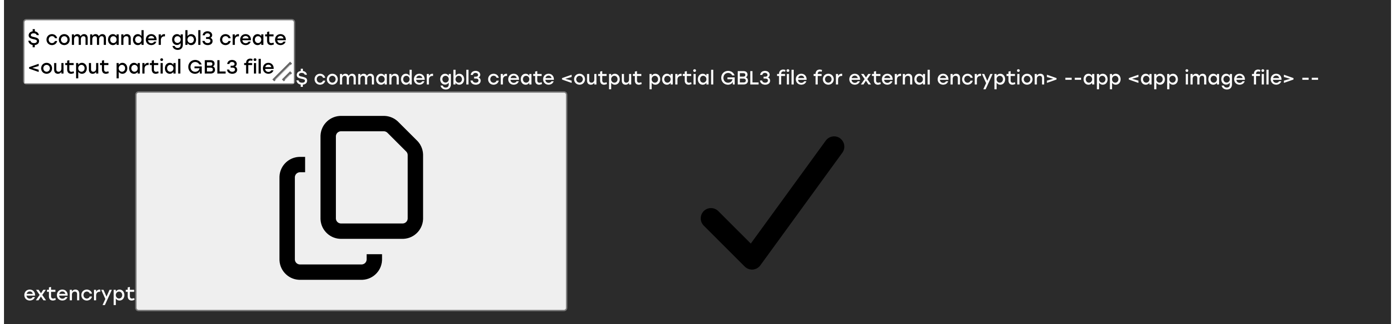
information, see *UG266: Silicon Labs Gecko Bootloader User's Guide for GSDK 3.x and Lower* or *UG489: Silicon Labs Gecko Bootloader User's Guide for GSDK 4.0 and Higher*.

## Preparing a GBL3 File for External Encryption

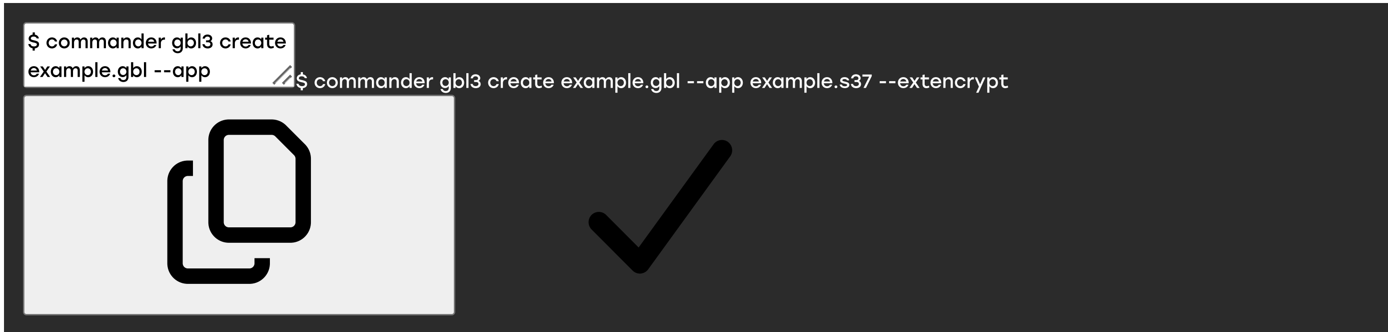
To prepare a GBL3 for external encryption, use the gbl3 create command with the --extencrypt option. This command creates a partial GBL3 file that an external party can encrypt. The data to be encrypted is written to the <app image file>.extencrypt file. It also creates intermediate files named <app image file>.extencrypt.header and <app image file>.extencrypt.footer . Do not modify or delete these files. They are required to construct the final encrypted GBL file when using the gbl3 encrypt command, as is described in Completing an Encrypted GBL3 Using a Hardware Security Module.

> Note: When encrypting the encryption-ready file, the name of the resulting encrypted file must be <app image file>.extencrypt.encrypted .

Command Line Syntax

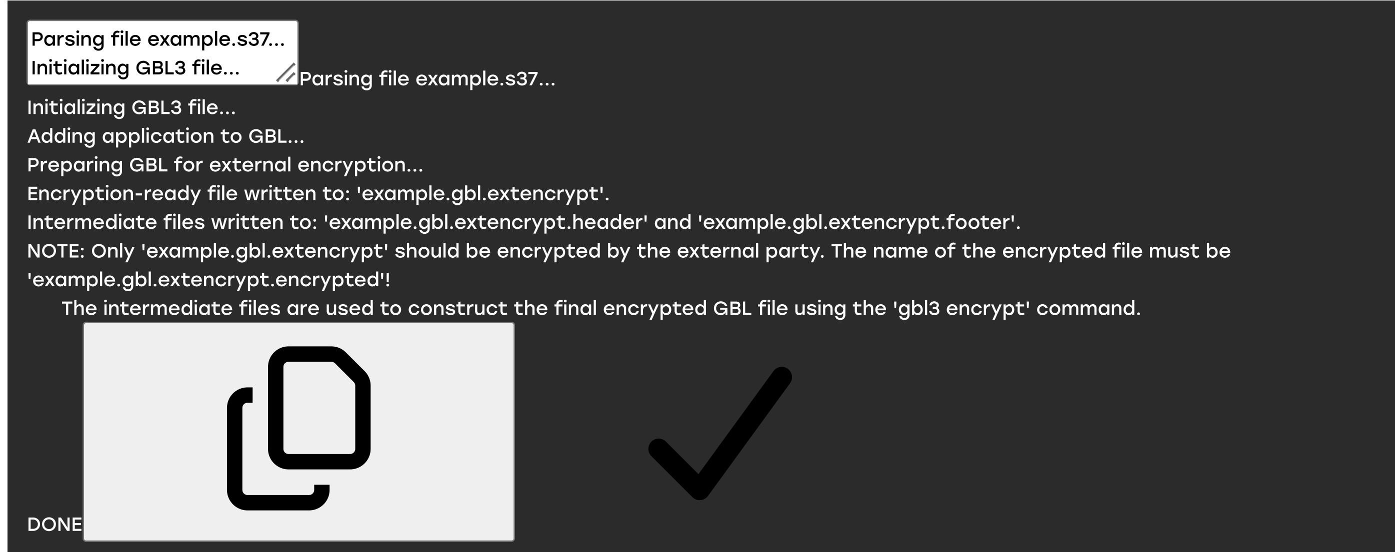



Command Line Input Example




Command Line Output Example

Copyright © 2026 Silicon Laboratories. All rights reserved.



```
Parsing file example.s37...
Initializing GBL3 file...
Adding application to GBL...
Preparing GBL for external encryption...
Encryption-ready file written to: 'example.gbl.extencrypt'.
Intermediate files written to: 'example.gbl.extencrypt.header' and 'example.gbl.extencrypt.footer'.
NOTE: Only 'example.gbl.extencrypt' should be encrypted by the external party. The name of the encrypted file must be 'example.gbl.extencrypt.encrypted'!
      The intermediate files are used to construct the final encrypted GBL file using the 'gbl3 encrypt' command.
DONE
```

## Preparing a GBL3 File for External Signing

To prepare a GBL3 for external signing, use the `gbl3 create` command with the `--extsign` option. This command creates a partial GBL3 file named `<app image file>.extsign` that an external party can sign. To complete the GBL3 file, append the generated signature to the partial GBL3 file by using the `gbl3 sign` command, as described in Completing a Signed GBL3 File Using a Hardware Security Module.

Command Line Syntax

```
$ commander gbl3 create <output partial GBL3 file for external signing> --app <app image file> --extsign [--encrypt <encryption key>]
```

Command Line Input Example

```
$ commander gbl3 create example.gbl.extsign --app example.s37 --extsign
```

Command Line Output Example

Copyright © 2026 Silicon Laboratories. All rights reserved.

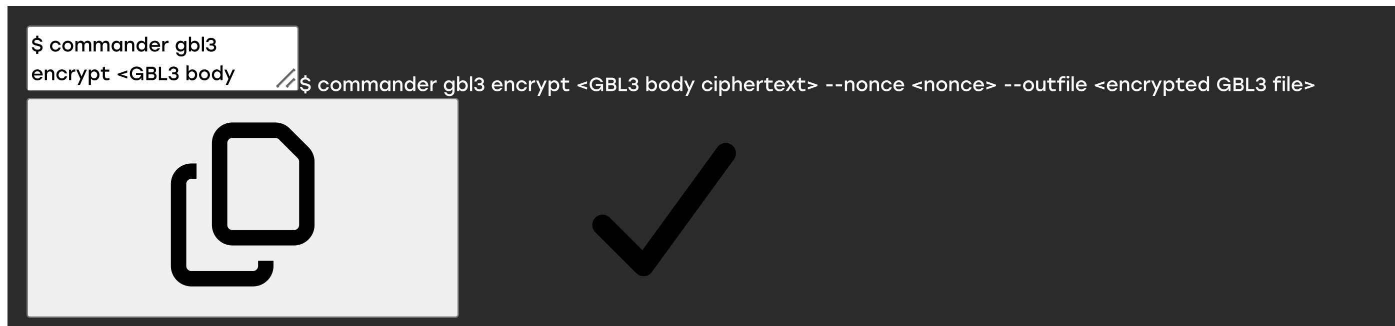
```
Parsing file example.s37...
Initializing GBL file...
Adding application to GBL...
Preparing GBL for external signing...
Writing GBL file example.gbl.extsign...
DONE
```

## Completing an Encrypted GBL3 Using a Hardware Security Module

Completes an encrypted GBL3 file from the intermediate files created by the `gbl3 create` command with the `--extencrypt` option and the encrypted data file from an HSM.

> Note: The nonce must be a 16-byte hexadecimal string in the format `02<12-byte random number>000001`. The HSM generates the random number component. Use the same nonce that was used to encrypt the GBL3 body ciphertext.

Command Line Syntax

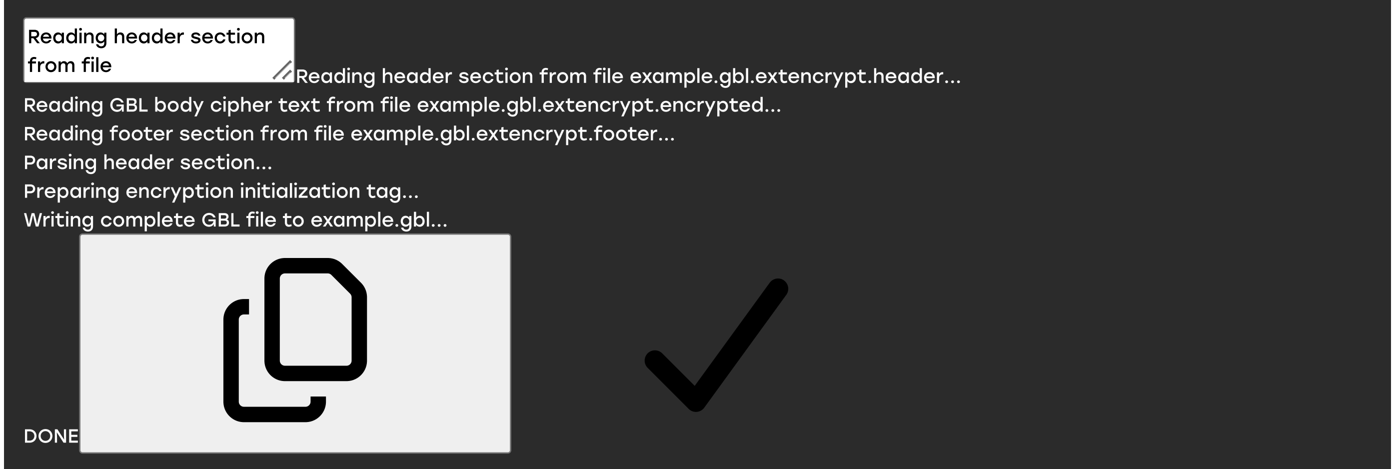


```
$ commander gbl3 encrypt <GBL3 body ciphertext> --nonce <nonce> --outfile <encrypted GBL3 file>
```

Command Line Input Example

```
$ commander gbl3 encrypt example.gbl.extencrypt.encrypted --nonce
020011223344556677889 9aabb000001 --outfile example.gbl
```

Command Line Output Example



```
Reading header section from file example.gbl.extencrypt.header...
Reading GBL body cipher text from file example.gbl.extencrypt.encrypted...
Reading footer section from file example.gbl.extencrypt.footer...
Parsing header section...
Preparing encryption initialization tag...
Writing complete GBL file to example.gbl...
DONE
```

## Completing a Signed GBL3 File Using a Hardware Security Module

Completes a signed GBL3 file from a partial GBL3 file and an ECDSA signature file in Distinguished Encoding Rules (DER) format generated as described in Preparing a GBL3 Upgrade File for Use with a Hardware Security Module. For more information, see *UG266: Silicon Labs Gecko Bootloader User's Guide for GSDK 3.x and Lower* or *UG489: Silicon Labs Gecko Bootloader User's Guide for GSDK 4.0 and Higher*.

Silicon Labs recommends that you use the `--verify` option with the public key corresponding to the private key used by the HSM to ensure the integrity of the generated GBL3 file.

Command Line Syntax

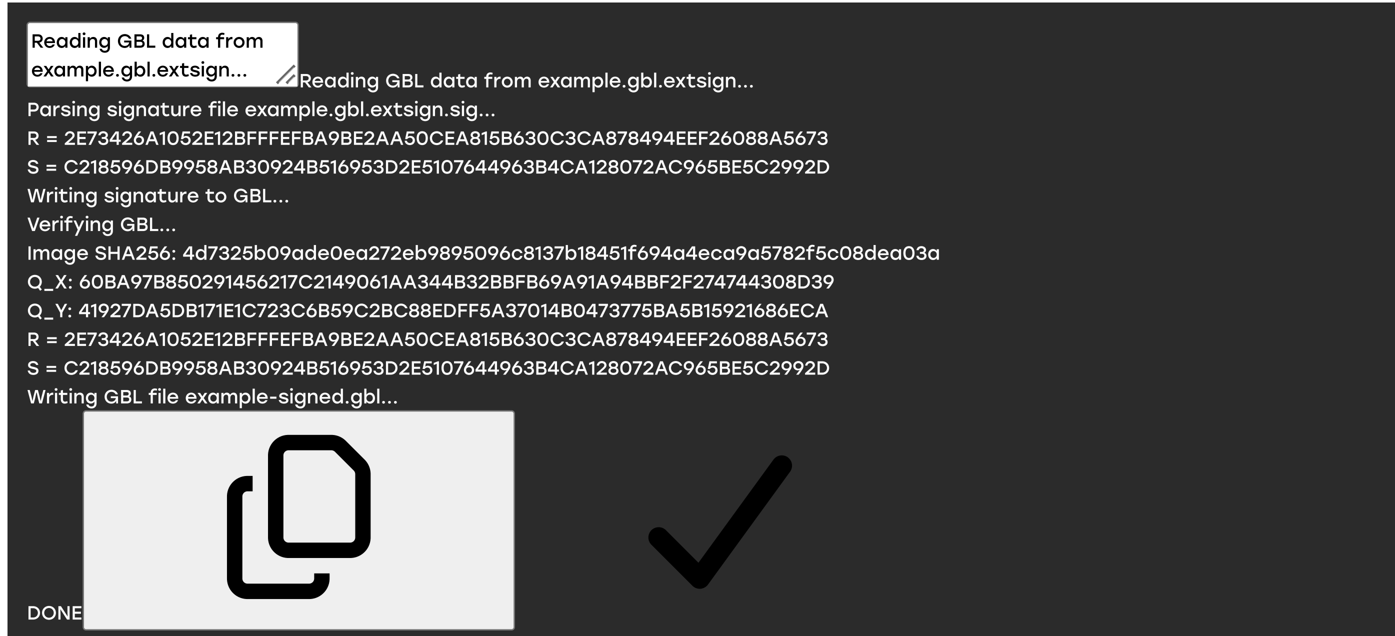
```
$ commander gbl3 sign <partial GBL3 file for external signing> --signature <signature from HSM> [--verify <public key file>] --outfile <signed GBL3 file>
```

Command Line Input Example

```
$ commander gbl3 sign example.gbl.extsign --signature example.gbl.extsign.sig --verify ecdsakey.pub --outfile example-signed.gbl
```

Command Line Output Example

Copyright © 2026 Silicon Laboratories. All rights reserved.



```
Reading GBL data from example.gbl.extsign...
Parsing signature file example.gbl.extsign.sig...
R = 2E73426A1052E12BFFFEFBA9BE2AA50CEA815B630C3CA878494EEF26088A5673
S = C218596DB9958AB30924B516953D2E5107644963B4CA128072AC965BE5C2992D
Writing signature to GBL...
Verifying GBL...
Image SHA256: 4d7325b09ade0ea272eb9895096c8137b18451f694a4eca9a5782f5c08dea03a
Q_X: 60BA97B850291456217C2149061AA344B32BBFB69A91A94BBF2F274744308D39
Q_Y: 41927DA5DB171E1C723C6B59C2BC88EDFF5A37014B0473775BA5B15921686ECA
R = 2E73426A1052E12BFFFEFBA9BE2AA50CEA815B630C3CA878494EEF26088A5673
S = C218596DB9958AB30924B516953D2E5107644963B4CA128072AC965BE5C2992D
Writing GBL file example-signed.gbl...
DONE
```

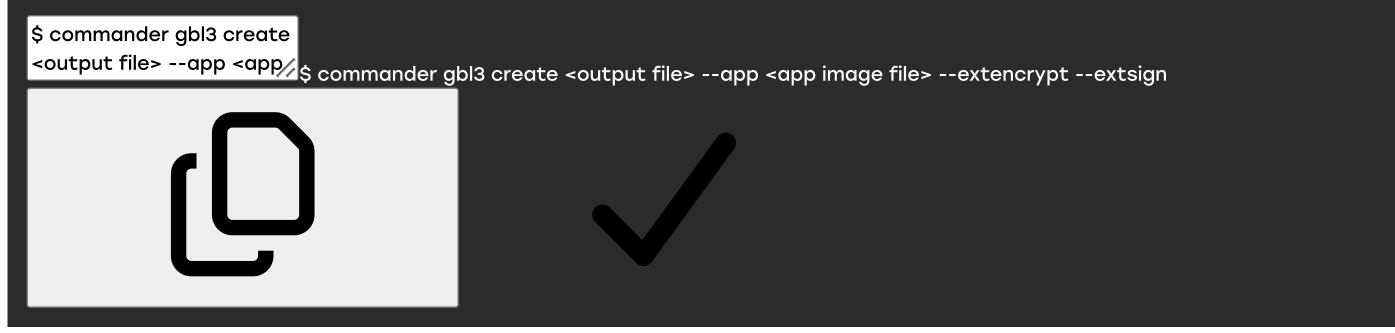
## Creating a Signed and Encrypted GBL3 File Using a Hardware Security Module

To create a GBL3 file that is both signed and encrypted by using a HSM, split the process into multiple steps, as described in the following sections.

### Creating the partial GBL3 file for external encryption and signing

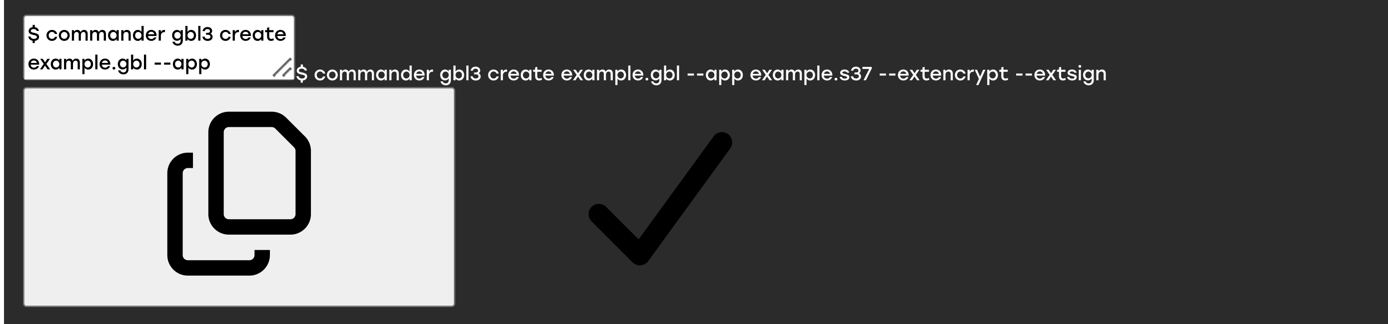
To create a partial GBL3 file for external encryption and signing, use the `--extencrypt` and `--extsign` options with the `gbl3 create` command. Doing so creates a partial GBL3 file that is ready to be encrypted and then signed by an external party.

Command Line Syntax



```
$ commander gbl3 create <output file> --app <app image file> --extencrypt --extsign
```

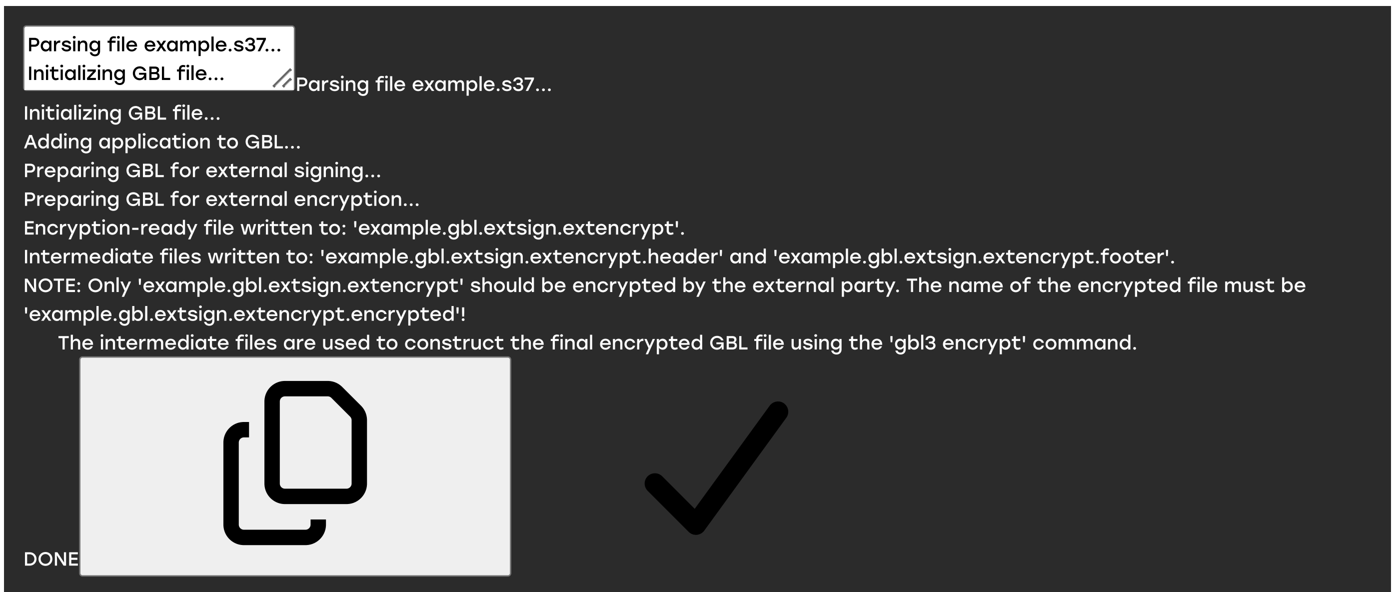
Command Line Input Example



```
$ commander gbl3 create example.gbl --app example.s37 --extencrypt --extsign
```

Copyright © 2026 Silicon Laboratories. All rights reserved.

Command Line Output Example



```
Parsing file example.s37...
Initializing GBL file...
Adding application to GBL...
Preparing GBL for external signing...
Preparing GBL for external encryption...
Encryption-ready file written to: 'example.gbl.extsign.extencrypt'.
Intermediate files written to: 'example.gbl.extsign.extencrypt.header' and 'example.gbl.extsign.extencrypt.footer'.
NOTE: Only 'example.gbl.extsign.extencrypt' should be encrypted by the external party. The name of the encrypted file must be 'example.gbl.extsign.extencrypt.encrypted'!
      The intermediate files are used to construct the final encrypted GBL file using the 'gbl3 encrypt' command.
DONE
```

## Encrypting the GBL3 Data Using a Hardware Security Module

The relevant data is encrypted by using an external HSM tool. In the next step, Commander requires the nonce that was used to encrypt the data. The nonce is a 16-byte hexadecimal value in the format `02<12-byte random number>000001`, where the random number is generated by the HSM.

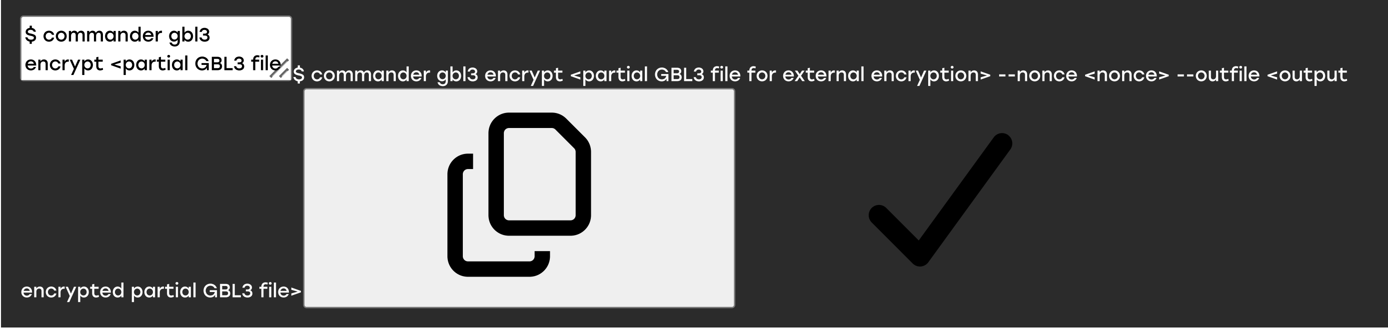
## Applying the external encryption to the partial GBL3 file

To apply external encryption to the partial GBL3 file, run the `gbl3 encrypt` command.

After this step, the GBL3 file is still not a valid GBL file. The file is complete only after you calculate the signature and append it to the GBL file.

See Completing an Encrypted GBL3 Using a Hardware Security Module for more information.

Command Line Syntax



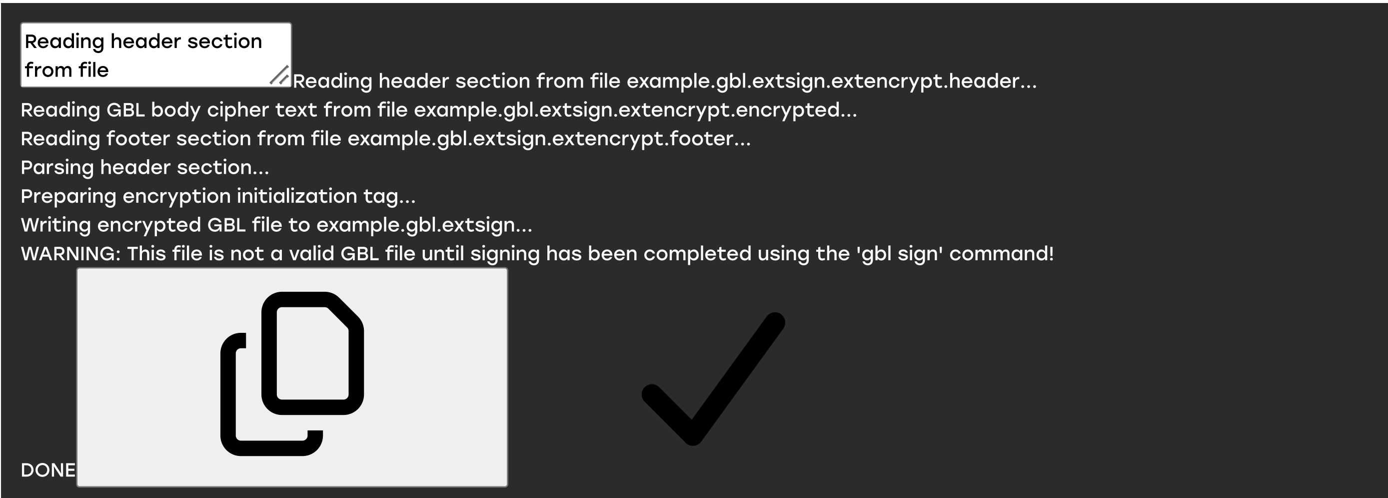
```
$ commander gbl3 encrypt <partial GBL3 file for external encryption> --nonce <nonce> --outfile <output encrypted partial GBL3 file>
```

Command Line Input Example

Copyright © 2026 Silicon Laboratories. All rights reserved.

```
$ commander gbl3 encrypt example.gbl.extsign.extencrypt.encrypted --nonce 0200112233445566778899aabb000001 --outfile example.gbl.extsign
```

Command Line Output Example



```
Reading header section from file example.gbl.extsign.extencrypt.header...
Reading GBL body cipher text from file example.gbl.extsign.extencrypt.encrypted...
Reading footer section from file example.gbl.extsign.extencrypt.footer...
Parsing header section...
Preparing encryption initialization tag...
Writing encrypted GBL file to example.gbl.extsign...
WARNING: This file is not a valid GBL file until signing has been completed using the 'gbl sign' command!
DONE
```

## Signing the GBL3 File Using a Hardware Security Module

Sign the encrypted GBL3 file by using the external HSM tool. The `<filename>.extsign` file contains the data to be signed. Write the resulting signature to a file, which will be used in the next step.

## Applying the external signature to the encrypted partial GBL3 file

To apply the external signature to the encrypted partial GBL3 file, use the `gbl3 sign` command. After this step, the GBL3 file is complete and can be used.

See Completing a Signed GBL3 File Using a Hardware Security Module for more information.

Command Line Syntax

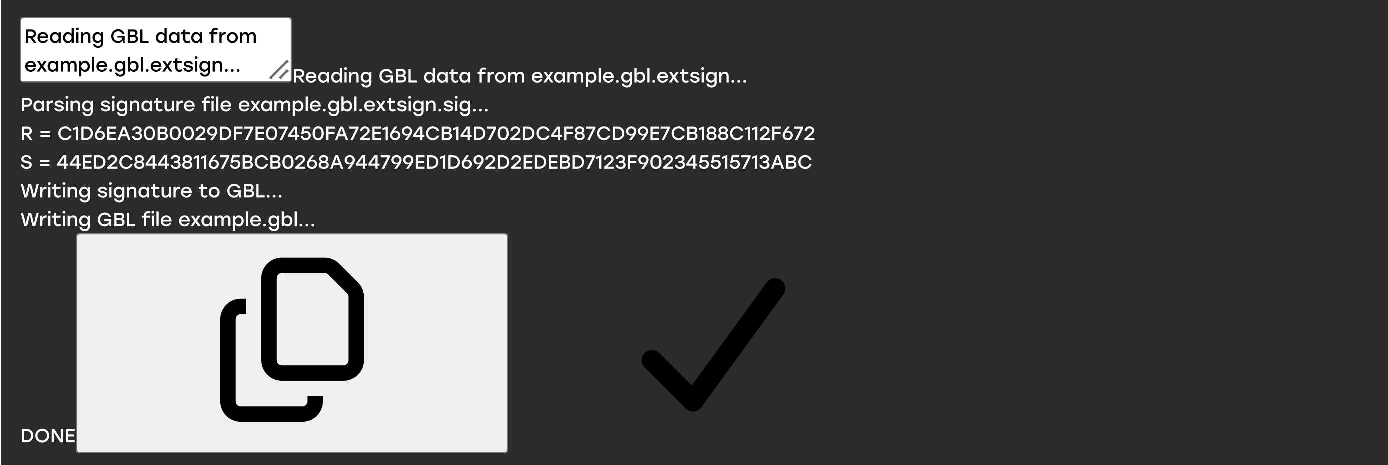
Copyright © 2026 Silicon Laboratories. All rights reserved.

```
$ commander gbl3 sign <encrypted partial GBL3 file> --signature <signature from HSM> [--verify <public key file>] --outfile <output signed GBL3 file>
```

Command Line Input Example

```
$ commander gbl3 sign example.gbl.extsign --signature example.gbl.extsign.sig --outfile example.gbl
```

Command Line Output Example



```
Reading GBL data from example.gbl.extsign...
Parsing signature file example.gbl.extsign.sig...
R = C1D6EA30B0029DF7E07450FA72E1694CB14D702DC4F87CD99E7CB188C112F672
S = 44ED2C8443811675BCB0268A944799ED1D692D2EDEBD7123F902345515713ABC
Writing signature to GBL...
Writing GBL file example.gbl...
DONE
```

## Creating a Signed GBL File Using an Intermediate Certificate

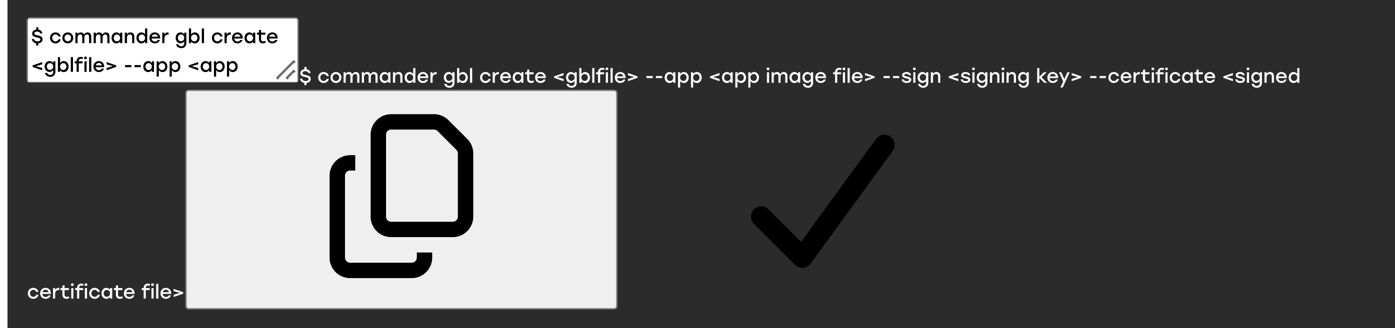
Creates a GBL file with a signed intermediate certificate and signs the GBL file using the private key that corresponds to the public key in the certificate.

A suitable certificate can be generated and signed using the `util gencert` command - see Generate Certificate for details.

> Note: To use GBL files with intermediate certificates the bootloader must be configured to:
>
> - Use certificate-based signing.
> - Include an intermediate certificate to authenticate the certificate embedded in the GBL file.

Command Line Syntax

Copyright © 2026 Silicon Laboratories. All rights reserved.



```
$ commander gbl create <gblfile> --app <app image file> --sign <signing key> --certificate <signed certificate file>
```

Command Line Input Example

```
$ commander gbl create example.gbl --app example.s37 --sign ecdsakey --certificate signed_cert.bin
```

Command Line Output Example

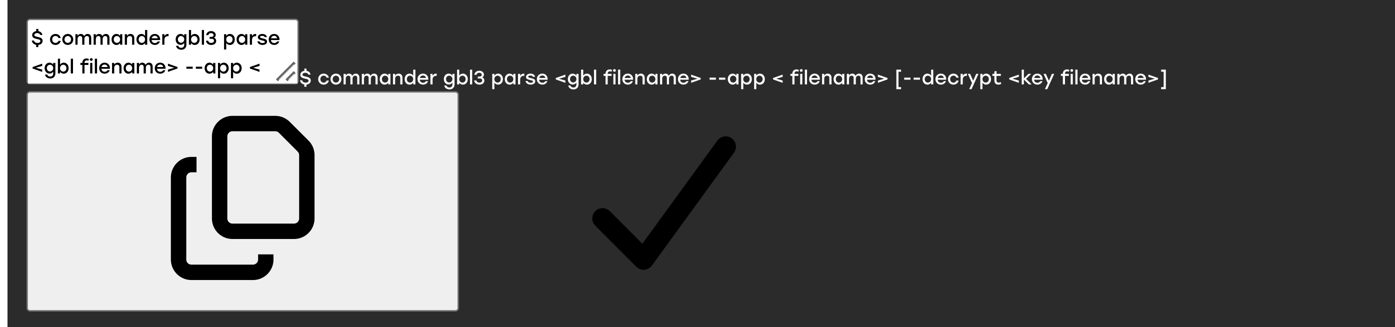
```
Parsing file example.s37...
Initializing GBL file...
Adding application to GBL...
Adding certificate to GBL...
Signing GBL...
Image SHA256: 6b5c6b5b4236c31931ce68d98abc65dfdf9a663ce4bd5a96d747811ea32e003a
R = 04B146AF5757307108CF09D0E2E34E99E1585BD80E6F121B5E643A251BEC4C0C
S = FEC5F26782A05508183486F6EDD2D8AA7F20A527066715243AB4AD9C7DB2BD2E
Writing GBL file example.gbl...
DONE
```

## GBL File Parsing

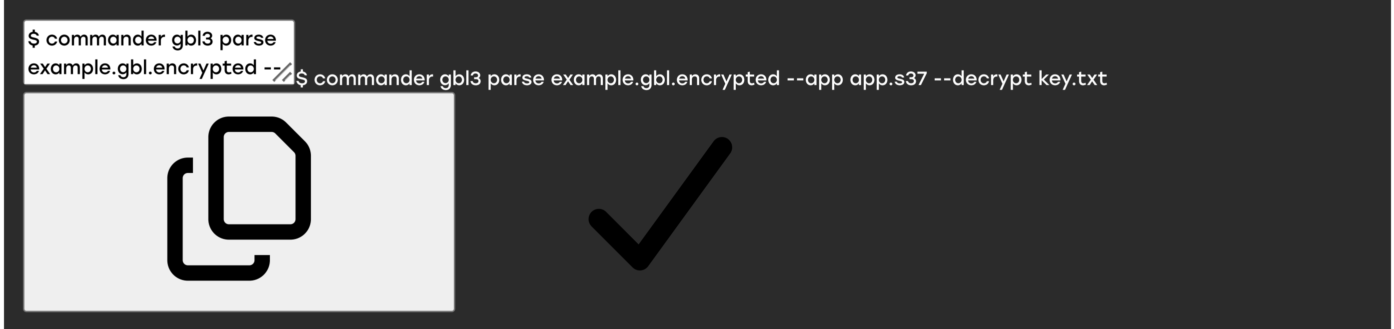
Parses a Gecko Bootloader version 3 (GBL3) file and writes the application image to the specified filename. Optionally decrypts an encrypted GBL3 file. The keyfile must be the same as was used for encrypting the encrypted GBL3 file.

Command Line Syntax

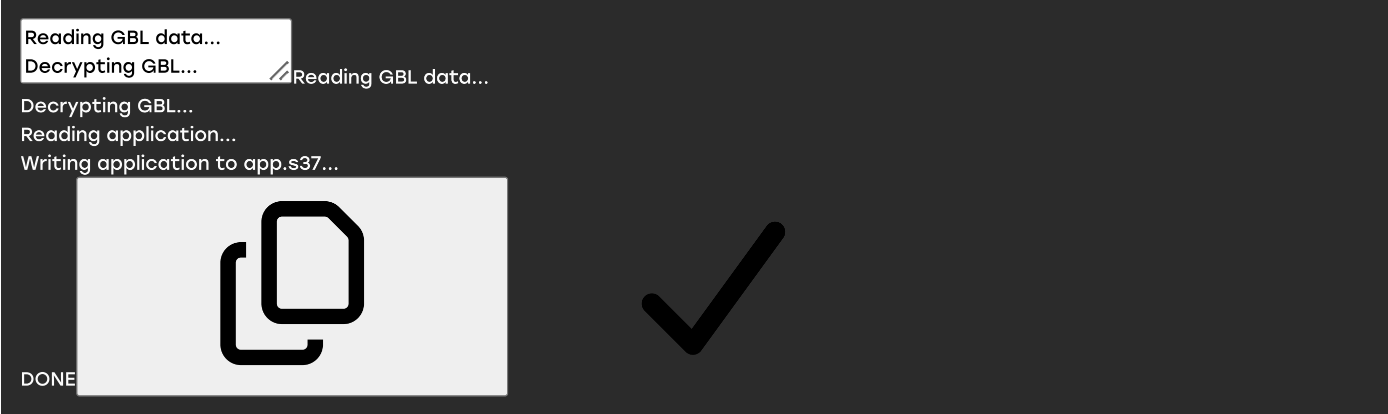
Copyright © 2026 Silicon Laboratories. All rights reserved.




Command Line Input Example




Command Line Output Example




## GBL3 Key Generation

This command is deprecated. See Key Generation for more information about key generation.

## Generating a Signing Key

This command is deprecated. See Generating a Signing Key for more information about generating a signing key.

## Generate a Signing Key Using a Hardware Security Module

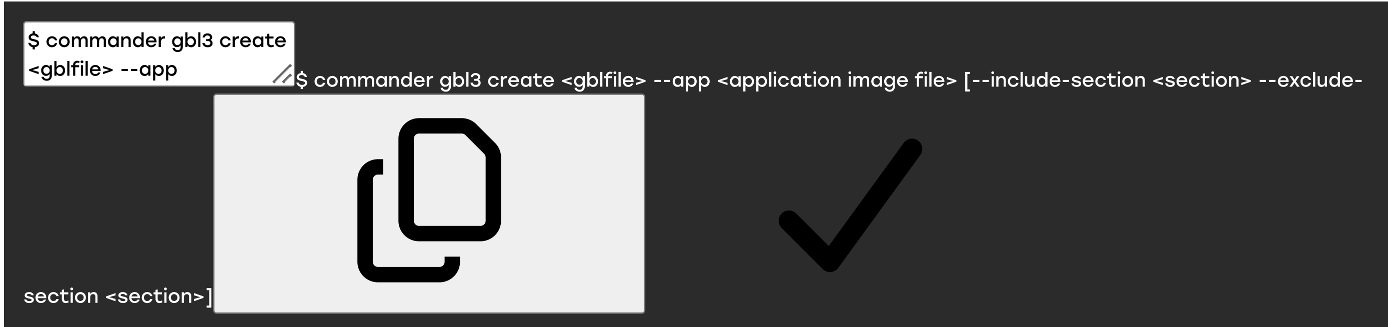
This command is deprecated. See Key to Token for more information about generating a signing key using a hardware security module.

## Create a GBL3 File from an ELF File

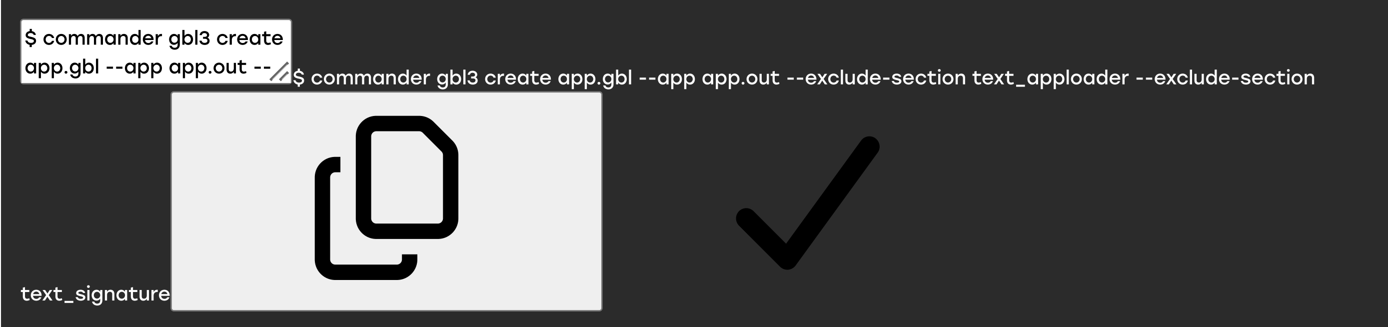
Copyright © 2026 Silicon Laboratories. All rights reserved.

Creates a Gecko Bootloader version 3 (GBL3) file from an Executable and Linkable Format (ELF) file and writes the output to the specified file. If neither the `--include-section` nor the `--exclude-section` option is used, Simplicity Commander will include all sections that appear to be part of the application.

Command Line Syntax

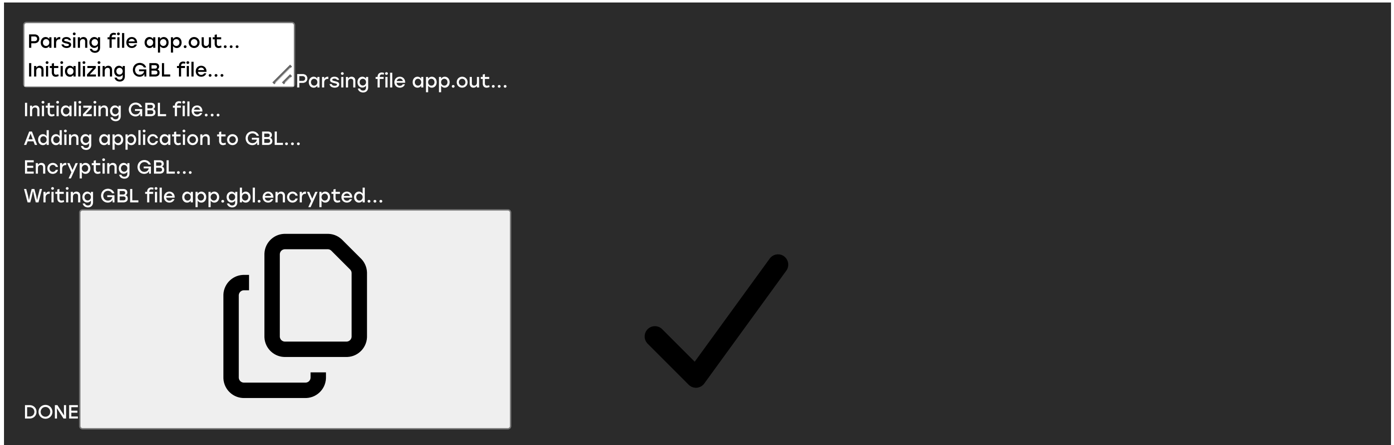



Command Line Input Example




Creates a GBL3 file containing an ELF application, excluding the `text_apploader` and `text_signature` sections from the application.

Command Line Output Example

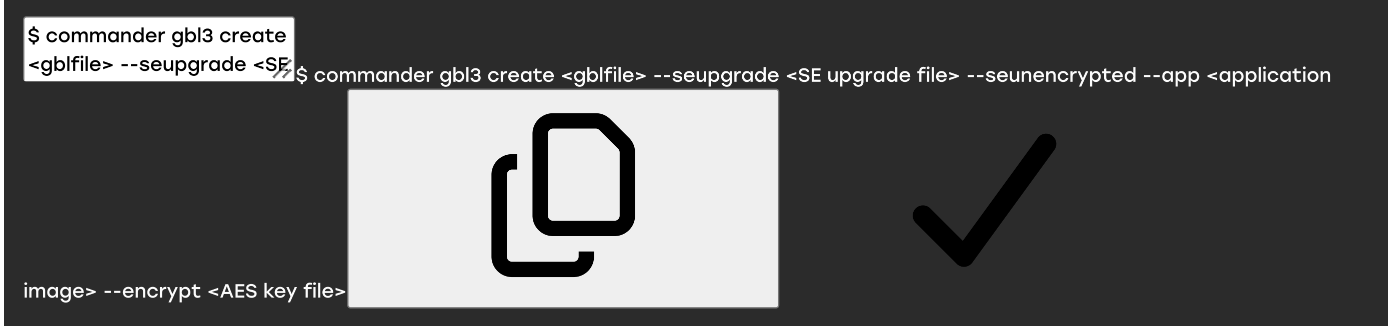



## Create an Encrypted GBL3 File with an Unencrypted Secure Engine Upgrade File

Creates an encrypted Gecko Bootloader version 3 (GBL3) file containing an unencrypted Secure Engine upgrade file and then writes the output to the specified GBL3 file.

Command Line Syntax

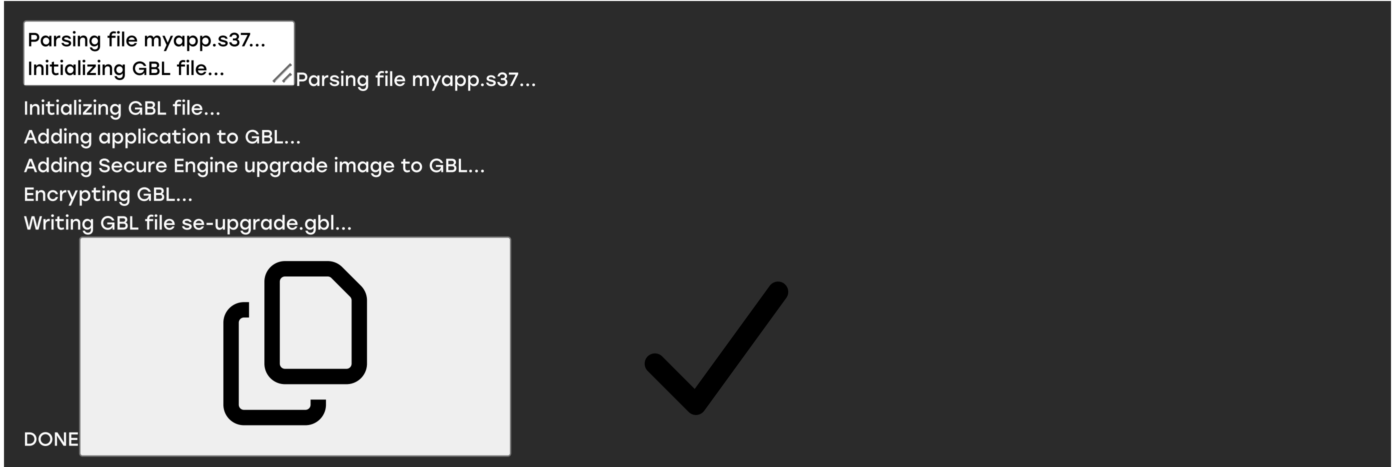
Copyright © 2026 Silicon Laboratories. All rights reserved.




Command Line Input Example

Creates an encrypted GBL3 file with a Secure Engine upgrade file outside the encrypted area of the file.

Command Line Output Example

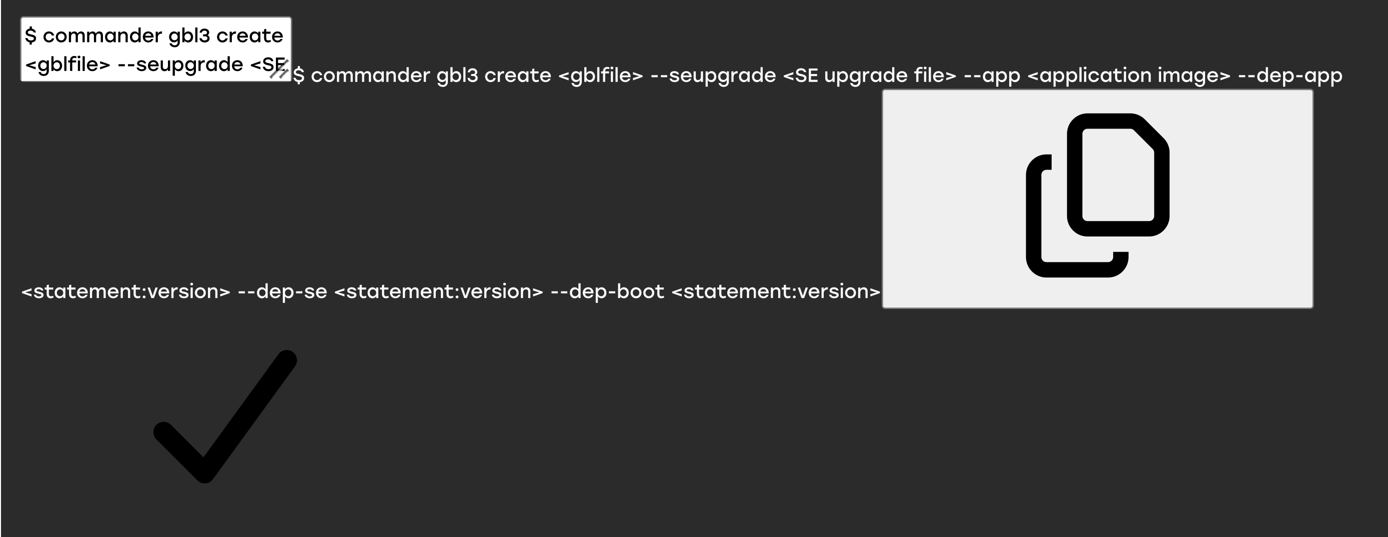



## Create a GBL3 File with Version Dependencies

Any version dependencies between application, bootloader, and Secure Engine upgrade files in a Gecko Bootloader version 3 (GBL3) file may be resolved using the `--dep-app`, `--dep-boot`, and `--dep-se` options.

Command Line Syntax

Copyright © 2026 Silicon Laboratories. All rights reserved.




Dependency Statement

The dependency statement may be one of the following:

| Simplicity Commander Input | Statement |
|---|---|
| g | Greater than |
| geq | Greater than or equal |
| eq | Equal |
| leq | Less than or equal |
| l | Less than |

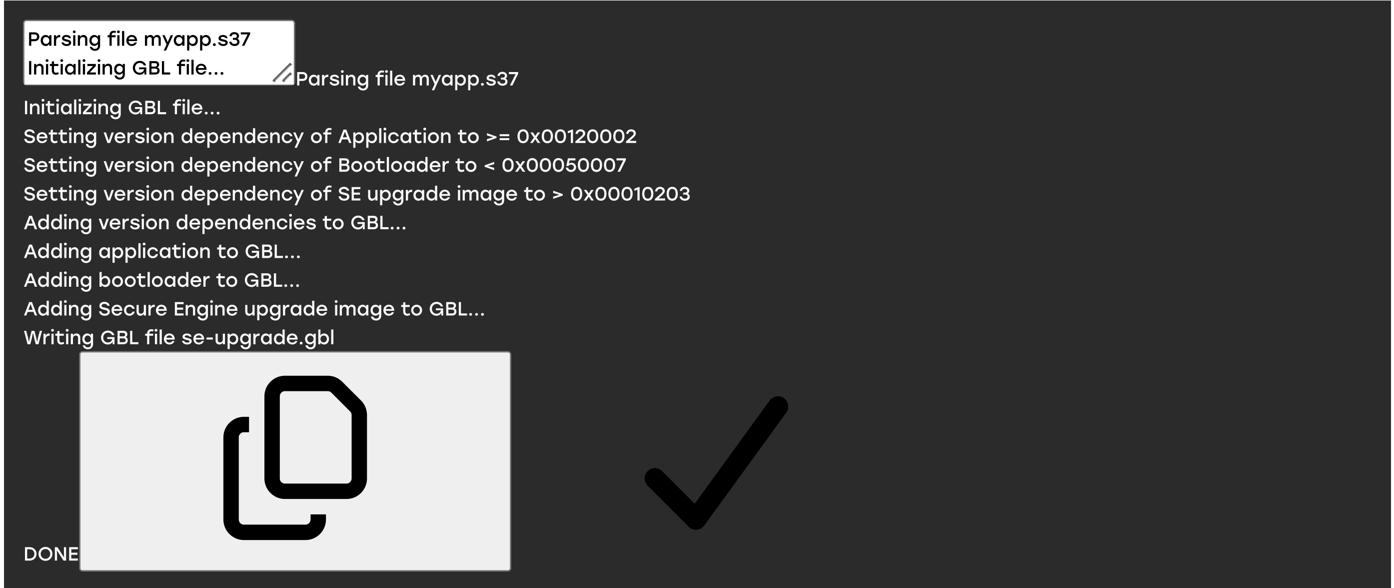
The `--dep-app` option takes an uint32 as version input, while the `--dep-se` and `--dep-boot` options take the version input in the format major.minor.patch.

Command Line Input Example

Creates a GBL3 where the application version must be greater than or equal to version 0x01020002, bootloader version must be less than version 0.5.7, and Secure Engine upgrade version must be greater than version 1.2.3.

Command Line Output Example

Copyright © 2026 Silicon Laboratories. All rights reserved.

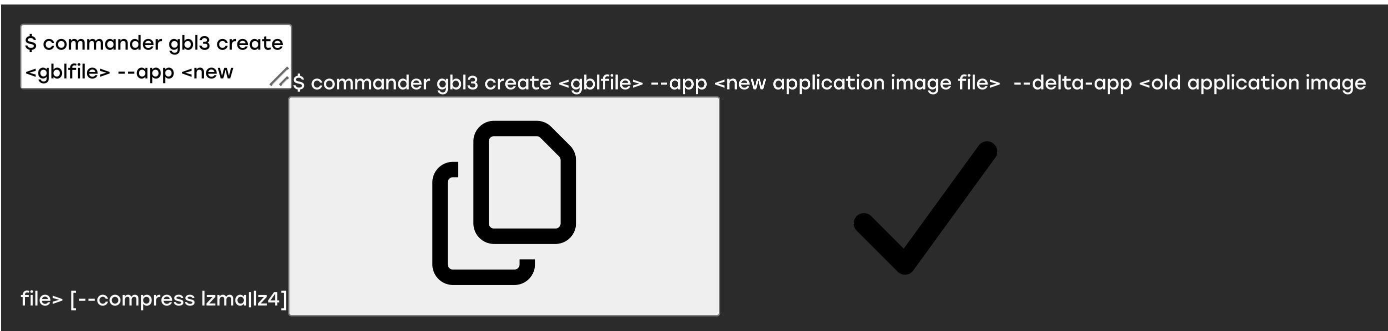


```
Parsing file myapp.s37
Initializing GBL file...
Setting version dependency of Application to >= 0x00120002
Setting version dependency of Bootloader to < 0x00050007
Setting version dependency of SE upgrade image to > 0x00010203
Adding version dependencies to GBL...
Adding application to GBL...
Adding bootloader to GBL...
Adding Secure Engine upgrade image to GBL...
Writing GBL file se-upgrade.gbl
DONE
```

## Create a Delta GBL3 File

Creates a Gecko Bootloader version 3 (GBL3) file which contains the difference between two specific application versions for minimal upgrade file size. For more information on delta updates, see *UG489: Silicon Labs Gecko Bootloader User's Guide for GSDK 4.0 and Higher*.

Command Line Syntax



```
$ commander gbl3 create <gblfile> --app <new application image file>  --delta-app <old application image file> [--compress lzma|lz4]
```

Symbol Information

The calculation of the delta between two application versions is significantly improved by having access to symbol information. When given files without symbol information (typically srec or hex files), Simplicity Commander will try to find ELF files with the same name in the same directory, and extract the symbol information from the ELF file. For example, if Commander is given `--app release/1.3.0/app.s37 --delta-app release/1.2.0/app.s37`, it will try to find `app.axf`, `app.out`, or `app.elf` in both of the `release/1.3.0/` and `release/1.2.0/` folders. Symbol information for both versions is required.

Secure Boot

If using secure boot, please ensure that both the old and the new application file is signed. The signature is then restored as part of applying the delta patch.

Additionally, delta GBL3 files may be signed and/or encrypted just like regular GBL3 files. See Creating a Signed and Encrypted GBL3 Upgrade Image File from an Application for details.

Version Dependency

Simplicity Commander will automatically extract the version of the old application and add a version dependency tag to the GBL3. See Create a GBL3 File with Version Dependencies for details on version

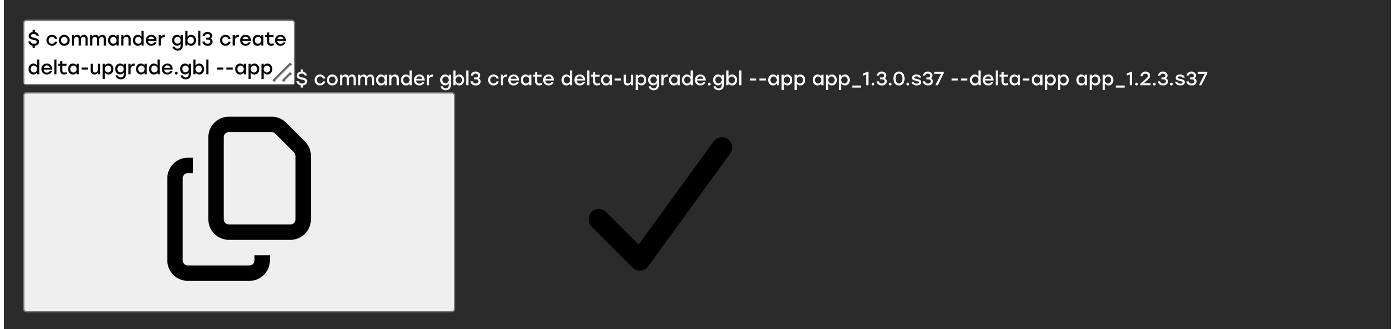
Copyright © 2026 Silicon Laboratories. All rights reserved.

dependencies. If a specific version is provided with the `--dep-app` option, this is used instead of the version found in the application image.

Compression

Delta GBL3 files can be compressed using LZMA or LZ4 compression. See GBL3 File Creation with Compression for details.

Command Line Input Example



```
$ commander gbl3 create delta-upgrade.gbl --app app_1.3.0.s37 --delta-app app_1.2.3.s37
```

Command Line Output Example

```
Parsing file app_1.3.0.s37...
Initializing GBL file...
Parsing file app_1.2.3.s37...
Filling with 0xFF from 0x0800631c to 0x08006320
Automatically adding dependency to the detected version of "--delta-app"...
Setting version dependency of Application to = 0x00010203
Writing GBL file delta-upgrade.gbl...
DONE
```

Copyright © 2026 Silicon Laboratories. All rights reserved.

GBL4 Commands

# GBL4 Commands

Creates and parses Gecko Bootloader (GBL) version 4 files for devices that support GBL4. GBL4 files are used to upgrade the bootloader, application, and/or SE firmware of supported devices.

## GBL4 File Creation With Config File

There are two ways to create a GBL4 file.

- Use a configuration file. The configuration file is a YAML file that describes the GBL file to be created.
- Describe the file through command line options - see GBL4 File Creation with Command Line Options.

> Note: When the `--config` option is used in the `create` command, all other command line options defining the output file are ignored.

A template configuration file can be created using the `gbl4 createconfig` command.

### GBL4 Config File Format

The configuration file is structured into several sections - `manifest`, `se_upgrade`, and `updates`. All sections are optional, but at least either one update file or an SE upgrade must be provided.

#### Manifest

The `manifest` section sets parameters which apply to the GBL file as a whole.

##### Product ID

The product ID is a 16 byte long identifier. It is extracted from the application image, if available. If the product ID is present in both the application and the configuration file, the entry in the configuration file is used. If not provided, the default is zero.

##### Bundle Version

The bundle version is the version of the upgrade bundle. Lower versions will be rejected by the bootloader. If not provided, the default is zero.

##### Minimum Version

The minimum version is the minimum acceptable previous bundle version to which this update can be applied. This may be needed if DFU is only a partial update.

##### Certificate and Signature

The GBL file can be signed with a private key, optionally using a delegate certificate authorizing this key pair. See Generate Certificate and Generating a Signing Key for details.

#### SE Upgrade

Copyright © 2026 Silicon Laboratories. All rights reserved.

The `se_upgrade` section is used to include a Secure Engine (SE) upgrade file in seuv2 format.

### Updates

The `updates` section provides a list of one or more updates to be included. For example, an update can be a bootloader or an application image.

#### Data

Each update must specify an update file in one of the supported formats: s37, hex, or elf.

#### Type, Capabilities, Version

The application type, capabilities, and version are by default extracted from the `ApplicationProperties_t` struct. These values can be overridden by the config file. If not provided at all, they are set to zero by default.

#### Compression

Each update can optionally be compressed. Supported algorithms are `lz4` and `lzma`.

#### Encryption

Each update can be encrypted with a symmetric key using the `aes-ctr-128` algorithm. Keys can be generated using the `util keygen` command.

#### Block Size

A list of hashes of the update data can be added to the GBL file. This can be useful for streaming updates, to allow authenticating the update on the fly. To use this feature, a block size must be set. The block size sets the size of a block of data for which a hash will be generated.

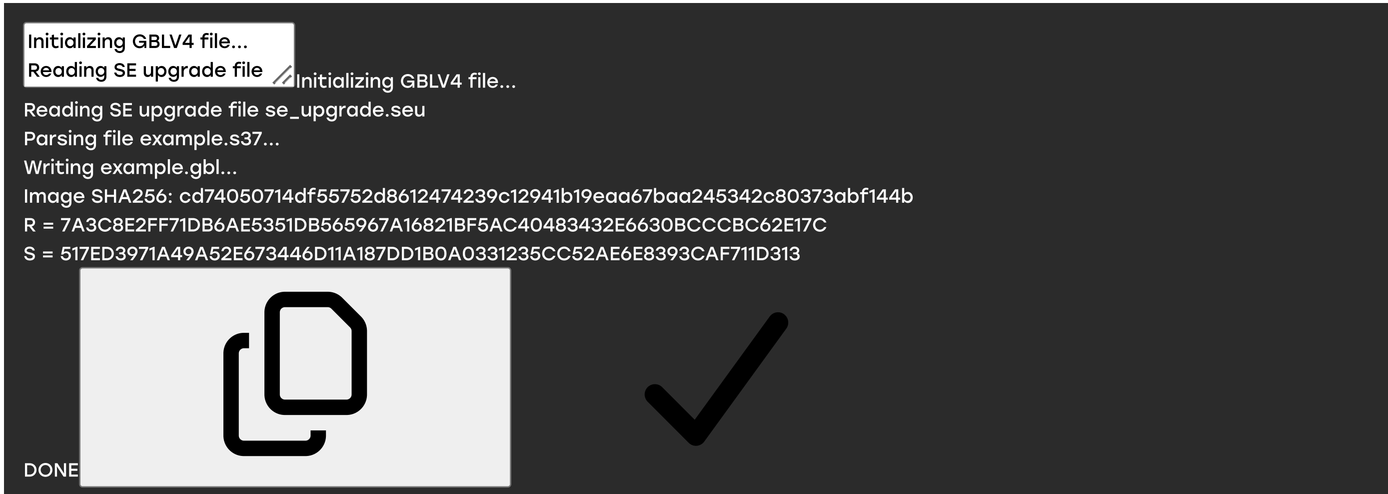
## Command Line Usage

Command Line Syntax

Command Line Input Example

Copyright © 2026 Silicon Laboratories. All rights reserved.

Command Line Output Example




## GBL4 File Creation with Command Line Options

GBL4 files can also be created using command line options. The command line options largely correspond to the options available in GBL4 Config File Format, with the following limitations/simplifications:

- The `--compress` and `--encrypt` options apply to all updates.
- Application type, capabilities, and version cannot be set manually.
- Block hashes are not supported.

Command Line Syntax

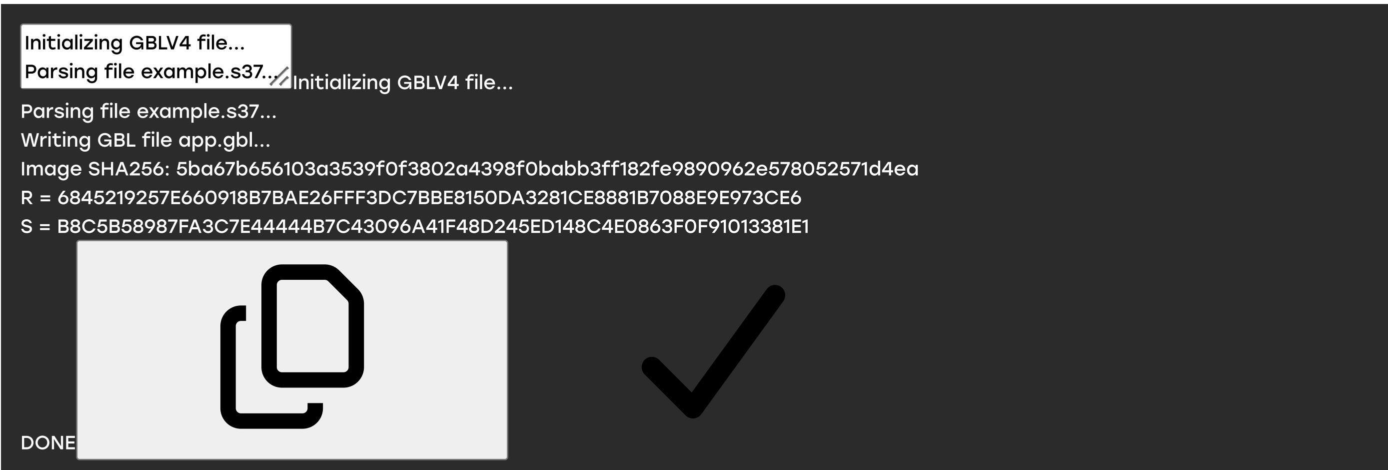
Copyright © 2026 Silicon Laboratories. All rights reserved.

```
$ commander gbl4 create <gblfile> [--data <filename>] [--seupgrade <filename>] [--compress <compression algorithm>] [--encrypt <keyfile>] [--bundleversion <version>] [--productid <product id>] [--minversion <version>] [--sign <keyfile>] [--certificate <certificate file>] [--device <device>]
```

Command Line Input Example

```
$ commander gbl4 create app.gbl --data example.s37 --compress lz4 --encrypt encryption_key.pem --sign signing_key.pem
```

Command Line Output Example



```
Initializing GBLV4 file...
Parsing file example.s37...
Writing GBL file app.gbl...
Image SHA256: 5ba67b656103a3539f0f3802a4398f0babb3ff182fe9890962e578052571d4ea
R = 6845219257E660918B7BAE26FFF3DC7BBE8150DA3281CE8881B7088E9E973CE6
S = B8C5B58987FA3C7E44444B7C43096A41F48D245ED148C4E0863F0F91013381E1
DONE
```

## Preparing a GBL4 File for Use with a Hardware Security Module

Simplicity Commander supports both external encryption and external signing of GBL4 files. To enable external encryption, specify the `--extencrypt` option when running the `gbl4 create` command. To enable external signing, use `--extsign` option. You can use the options together to create a GBL4 file that is both encrypted and signed externally. The following sections describe a generic workflow for enabling external encryption, external signing, or both.

1. Prepare the GBL4 file for external encryption and/or signing using Simplicity Commander.
2. If external encryption is used:
   a. Encrypt the relevant data blobs using the external HSM tool.

Copyright © 2026 Silicon Laboratories. All rights reserved.

   b. Use Simplicity Commander to assemble the encrypted data into an encrypted GBL4 file.
3. If external signing is used:
   a. Create an Elliptic Curve Digital Signature Algorithm (ECDSA) signature of the relevant data using an HSM.
   b. Use Simplicity Commander to sign the GBL4 file using the signature from the HSM, completing the GBL4 file.

Step 1 is described in Preparing a GBL4 File for External Encryption and Preparing a GBL4 File for External Signing. Steps 2a and 3a are specific to the HSM you are using. Step 2b is described in Completing an Externally Encrypted GBL4 File, and step 3b is described in Completing an Externally Signed GBL4 File. A walkthrough of the process of both externally encrypting and externally signing is provided in Creating a Signed and Encrypted GBL4 File Using a Hardware Security Module.

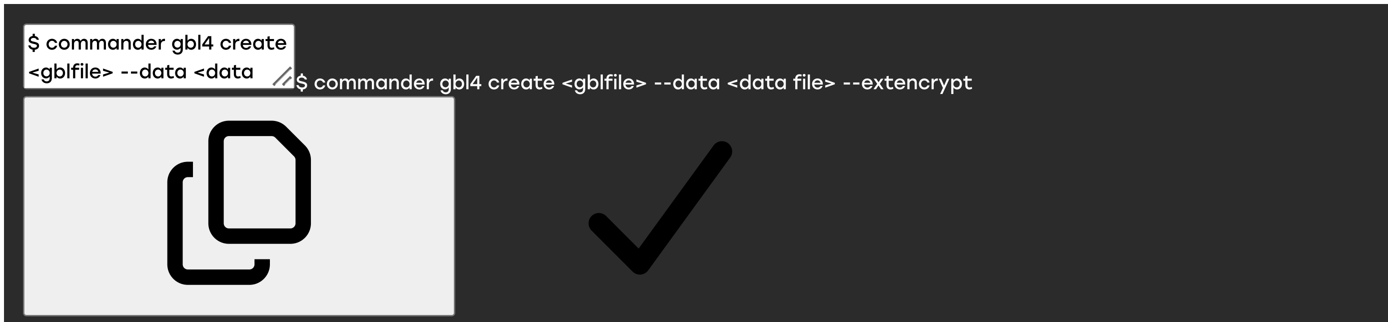
## Preparing a GBL4 File for External Encryption

To prepare a GBL4 file for external encryption, use the `gbl4 create` command with the `--extencrypt` option. This command creates several intermediate files, depending on the number of updates provided. The base GBL4 file is written to `<gblfile>.unencrypted`. The GBL4 data to be encrypted is written to files named `<gblfile>.blob_*.data`, while the intermediate files are written to files named `<gblfile>.blob_*.info`.

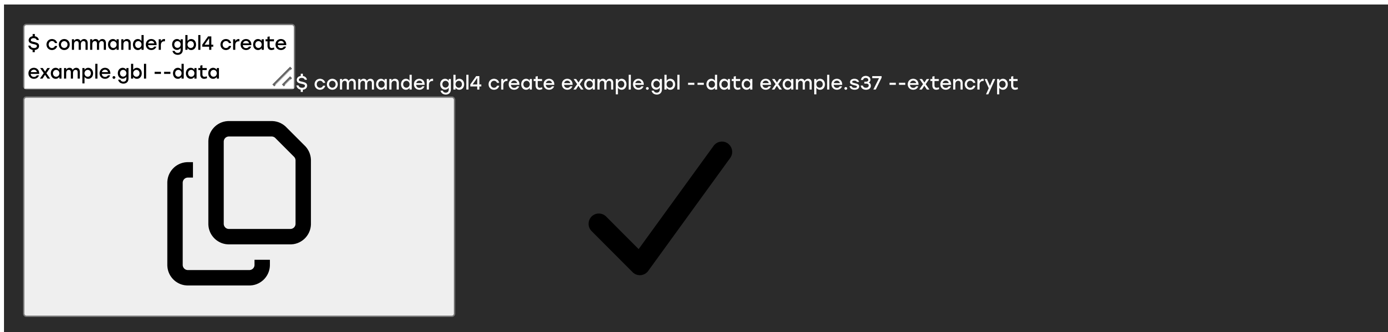
Name the encrypted data blobs `<gblfile>.blob_*.data.encrypted`, preserving the index for each memory section. Encrypt each data blob by using the external HSM tool.

> Note: Do not modify or delete the intermediate files. These files are required to construct the final encrypted GBL file when you use the `gbl4 encrypt` command.

Command Line Syntax

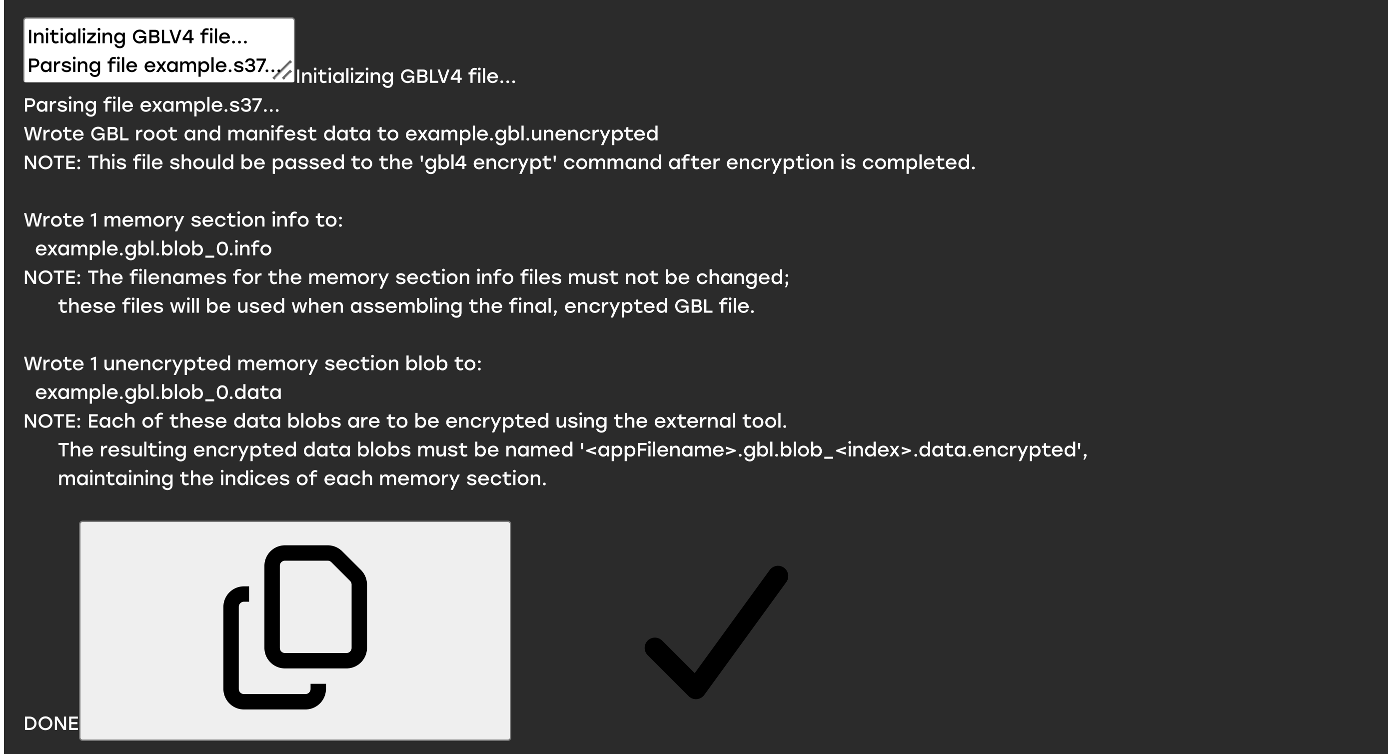



Command Line Input Example




Command Line Output Example

Copyright © 2026 Silicon Laboratories. All rights reserved.

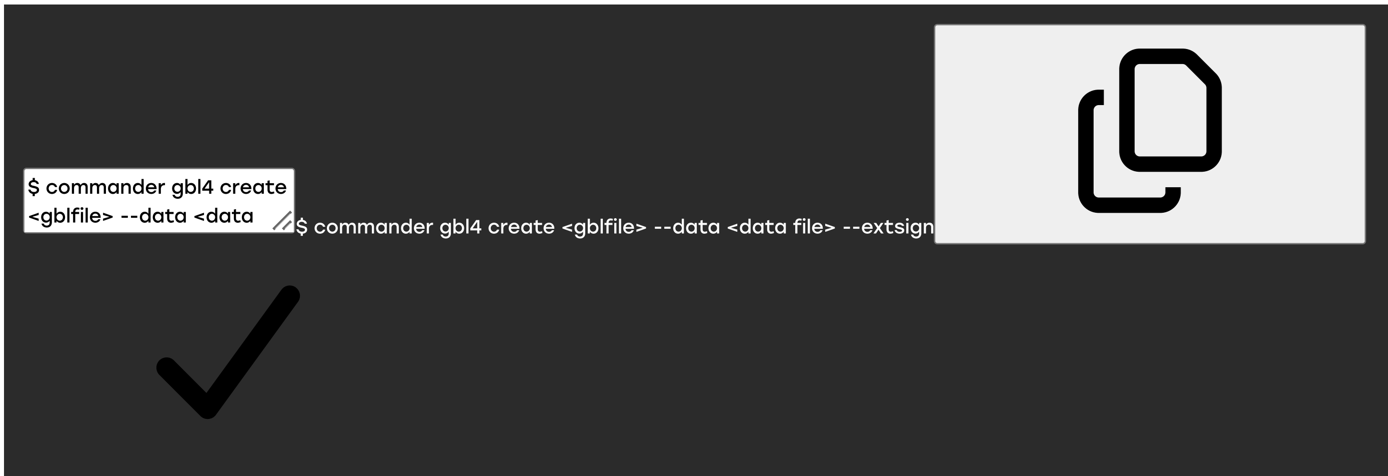


```
Initializing GBLV4 file...
Parsing file example.s37...
Wrote GBL root and manifest data to example.gbl.unencrypted
NOTE: This file should be passed to the 'gbl4 encrypt' command after encryption is completed.

Wrote 1 memory section info to:
  example.gbl.blob_0.info
NOTE: The filenames for the memory section info files must not be changed;
      these files will be used when assembling the final, encrypted GBL file.

Wrote 1 unencrypted memory section blob to:
  example.gbl.blob_0.data
NOTE: Each of these data blobs are to be encrypted using the external tool.
      The resulting encrypted data blobs must be named '<appFilename>.gbl.blob_<index>.data.encrypted',
      maintaining the indices of each memory section.

DONE
```

## Preparing a GBL4 File for External Signing

To prepare a GBL4 file for external signing, use the `gbl4 create` command with the `--extsign` option. This command creates a partial GBL4 file that an external party can sign. The base GBL4 file is written to a file named `<gblfile>.unsigned` , while the data to be signed is written to a file named `<gblfile>.manifest` . Append The generated signature to the partial GBL4 file by using the `gbl4 sign` command.

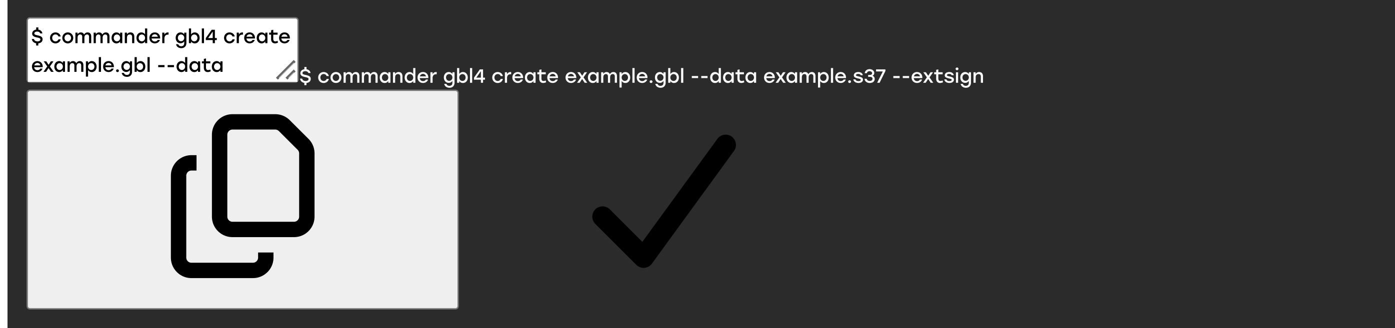
Command Line Syntax



```
$ commander gbl4 create <gblfile> --data <data file> --extsign
```

Command Line Input Example

Copyright © 2026 Silicon Laboratories. All rights reserved.

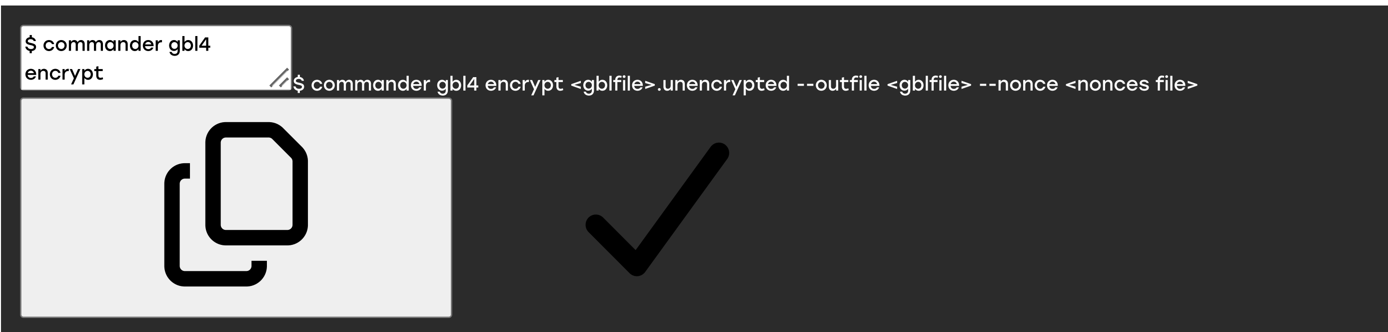



Command Line Output Example

## Completing an Externally Encrypted GBL4 File

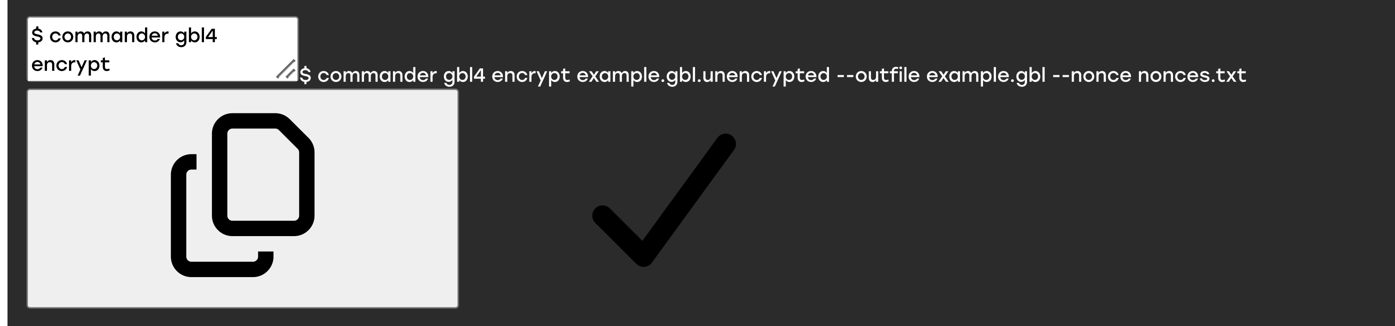
To finalize an externally encrypted GBL4 file, use the `gbl4 encrypt` command. Provide a nonce file that contains one nonce per memory section. The nonce file must be a text file, with each nonce on a separate line, using the format `02<12-byte random number>000001`, where the random number is generated by the HSM. Each line must correspond to the index of the associated memory section, and the number of nonces must match the number of encrypted memory sections.

Command Line Syntax




Command Line Input Example

Copyright © 2026 Silicon Laboratories. All rights reserved.




Command Line Output Example

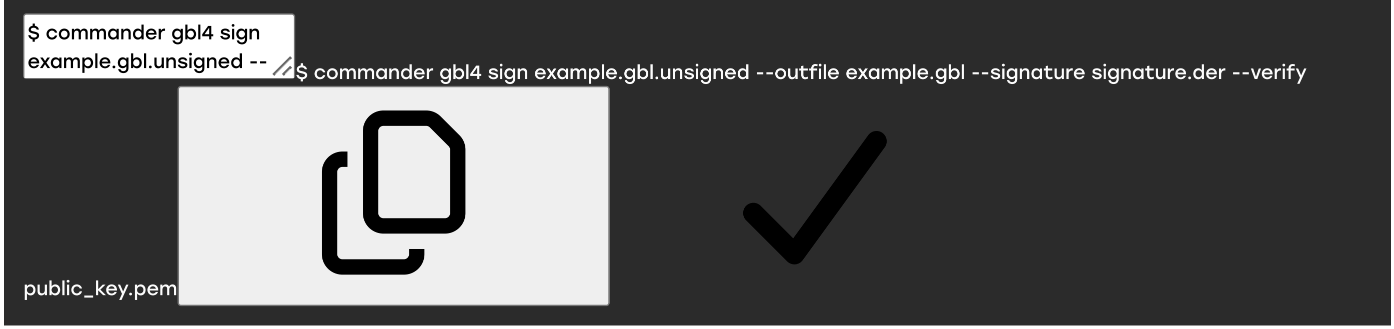
## Completing an Externally Signed GBL4 File

To finalize an externally signed GBL4 file, use the `gbl4 sign` command. The signature file must contain a DER-formatted ECDSA signature generated by the HSM.

Silicon Labs recommends that you use the `--verify` option with the public key corresponding to the private key used by the HSM to ensure the integrity of the generated GBL4 file.

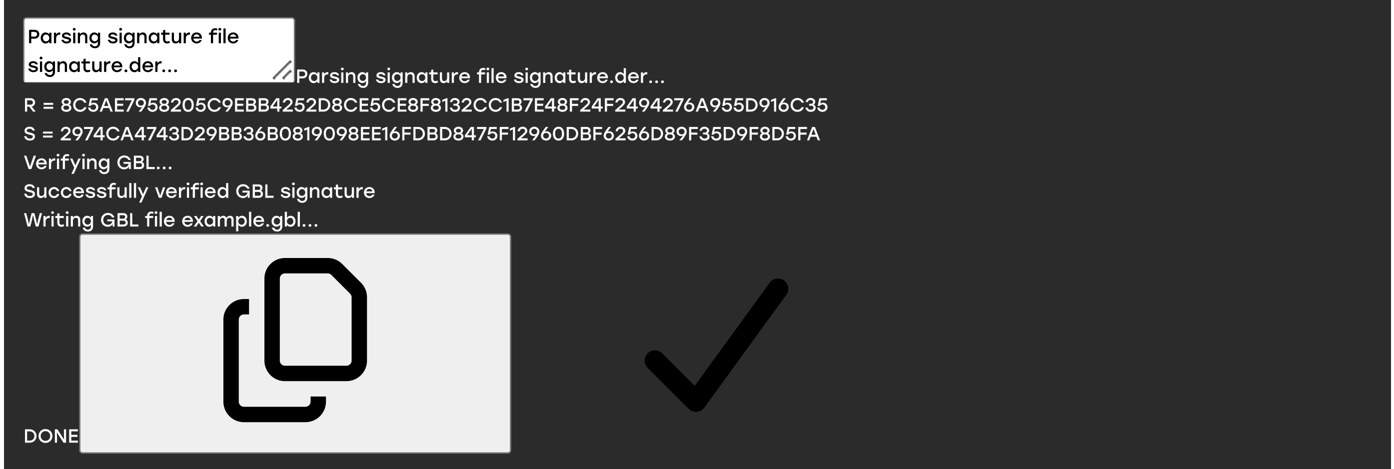
Command Line Syntax

Command Line Input Example




Command Line Output Example

Copyright © 2026 Silicon Laboratories. All rights reserved.



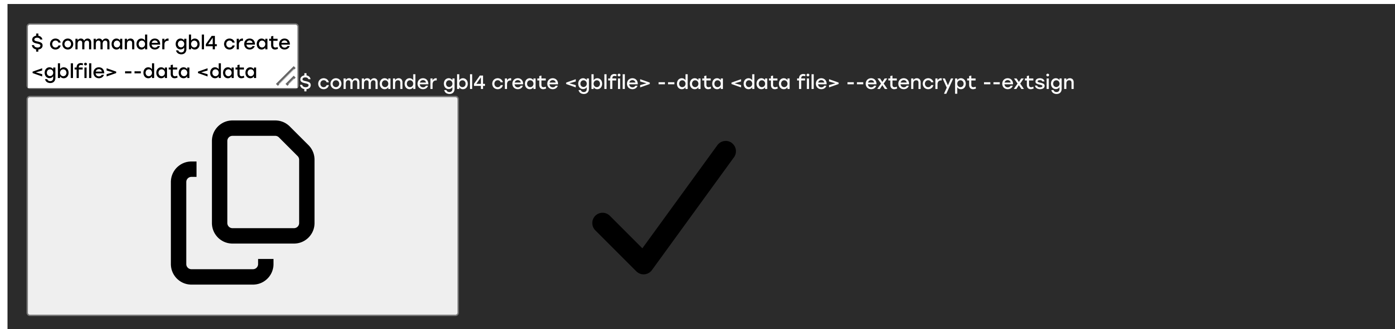

# Creating a Signed and Encrypted GBL4 File Using a Hardware Security Module

To create a GBL4 file that is both signed and encrypted using a Hardware Security Module (HSM), split the process into multiple steps, as described in the following sections.

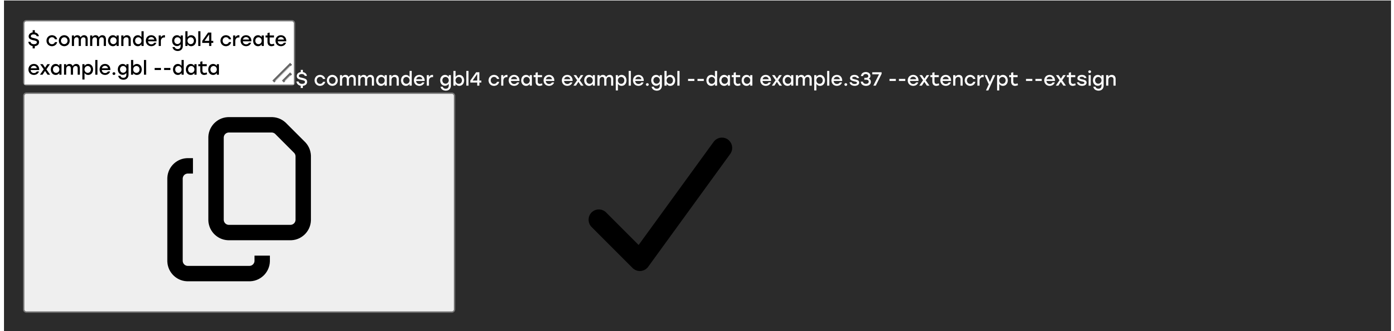
## Preparing the GBL4 File for External Encryption and Signing

To prepare a GBL4 file for both external encryption and external signing, use the `--extencrypt` and `--extsign` options with the 'gbl4 create' command. Doing so creates a partial GBL4 file that is ready to be encrypted and then signed by an external party.

Command Line Syntax




Command Line Input Example




Command Line Output Example

Copyright © 2026 Silicon Laboratories. All rights reserved.

```
Initializing GBLV4 file...
Parsing file example.s37...
Wrote GBL root and manifest data to example.gbl.unencrypted
NOTE: This file should be passed to the 'gbl4 encrypt' command after encryption is completed.

Wrote 1 memory section info to:
  example.gbl.blob_0.info
NOTE: The filenames for the memory section info files must not be changed;
      these files will be used when assembling the final, encrypted GBL file.

Wrote 1 unencrypted memory section blob to:
  example.gbl.blob_0.data
NOTE: Each of these data blobs are to be encrypted using the external tool.
      The resulting encrypted data blobs must be named '<appFilename>.gbl.blob_<index>.data.encrypted',
      maintaining the indices of each memory section.

DONE
```

## Encrypting the GBL4 Data Using a Hardware Security Module

The relevant data is encrypted by using an external HSM tool. In the next step, Commander requires the nonce used to encrypt the data. The nonce is a 16-byte hexadecimal value in the format `02<12-byte random number>000001`, where the random number is generated by the HSM. Each encrypted data blob can use a different nonce. Write the nonces to a file, which is used in the next step.

The encrypted data blobs must be named `<gblfile>.blob_*.data.encrypted`, maintaining the original indices of each data blob.

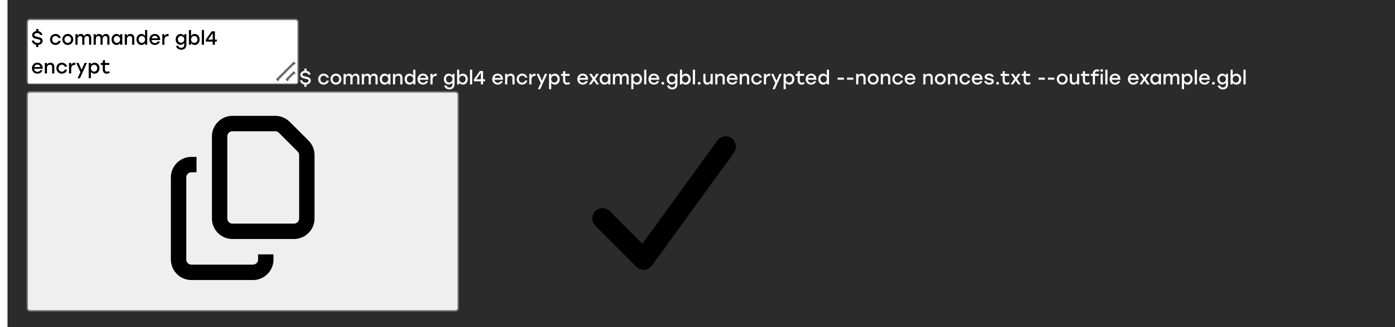
## Applying the external encryption to the partial GBL4 file

To apply the external encryption to the partial GBL4 file, use the `gbl4 encrypt` command. After this step, the GBL4 file is still not a valid GBL file. The file becomes complete only after you calculate the signature and append it to the GBL file. See Completing an Externally Encrypted GBL4 File for more information.

Command Line Syntax

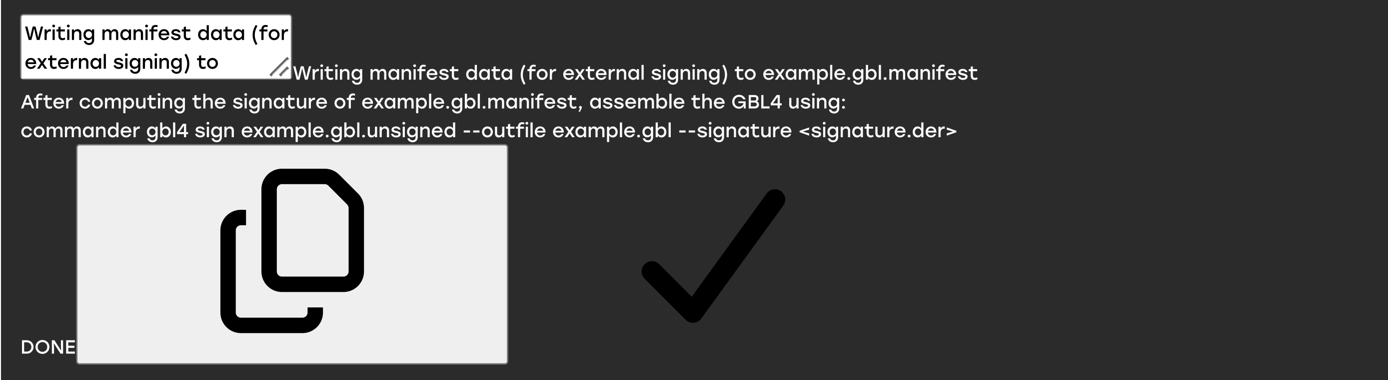
```
$ commander gbl4 encrypt <partial GBL4 file for external encryption> --nonce <nonce file> --outfile <output encrypted partial GBL4 file>
```

Command Line Input Example



```
$ commander gbl4 encrypt example.gbl.unencrypted --nonce nonces.txt --outfile example.gbl
```

Command Line Output Example



```
Writing manifest data (for external signing) to example.gbl.manifest
After computing the signature of example.gbl.manifest, assemble the GBL4 using:
commander gbl4 sign example.gbl.unsigned --outfile example.gbl --signature <signature.der>
DONE
```

## Signing the GBL4 File Using a Hardware Security Module

Sign the encrypted GBL4 file by using the external HSM tool. The `<filename>.manifest` file contains the data that should be signed. Write the resulting signature to a file that will be used in the next step.

## Applying the external signature to the encrypted GBL4 file

To apply the external signature to the encrypted GBL4 file, use the `gbl4 sign` command. After this step, the GBL4 file is complete and can be used.

See Completing an Externally Signed GBL4 File for more information.

Command Line Syntax

```
$ commander gbl4 sign <encrypted GBL4 file> --signature <signature from HSM> --outfile <output signed GBL4 file> [--verify <public key file>]
```

Command Line Input Example

Copyright © 2026 Silicon Laboratories. All rights reserved.

```
$ commander gbl4 sign example.gbl --signature signature.der --outfile example.gbl --verify public_key.pem
```

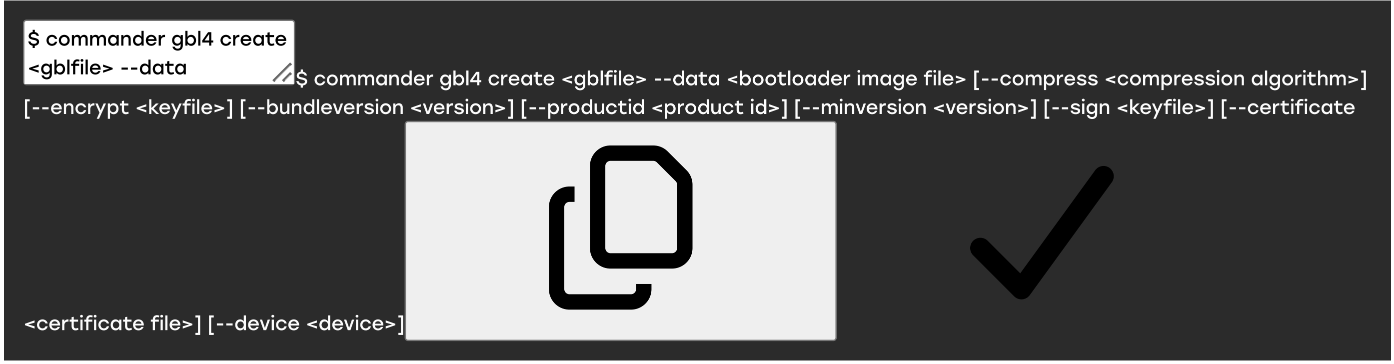
Command Line Output Example

```
Parsing signature file signature.der...
R = 6A0F82592EA11BA96DCFE0890C079924AE5EEAB1FB4B9C2F051AAF7F623CA7B1
S = 70C467A9C9DD232301112EB143541DE5FCA2223844A6325CA33D3194F0601985
Verifying GBL...
Successfully verified GBL signature
Writing GBL file example.gbl...
DONE
```

## Create a GBL4 File for Bootloader Upgrade

Creates a Gecko Bootloader (GBL) file from a bootloader image and writes the output to the specified bootloader image filename.

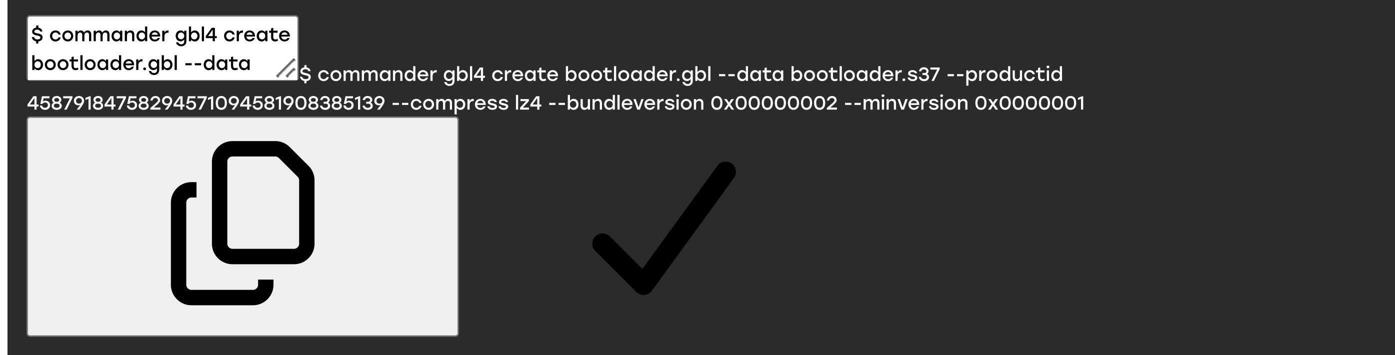
Command Line Syntax



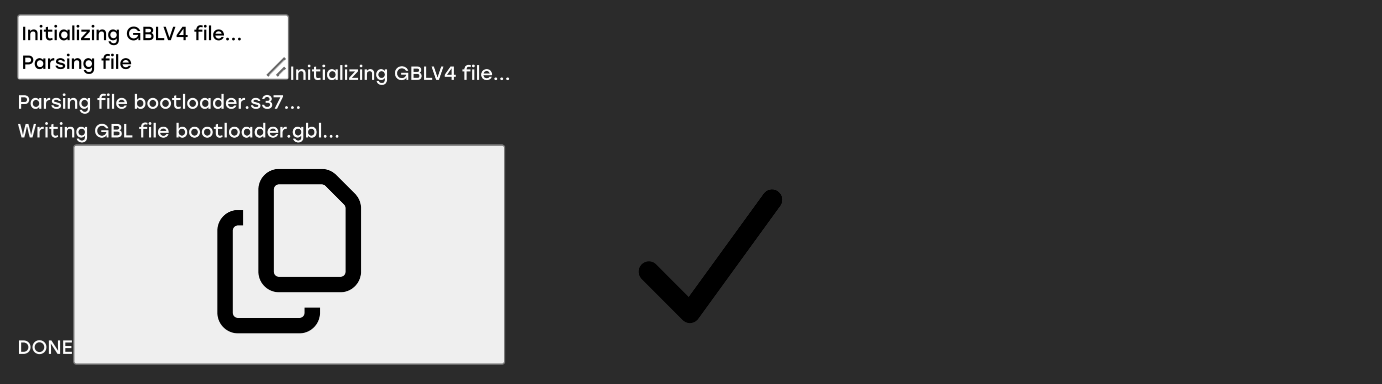
```
$ commander gbl4 create <gblfile> --data <bootloader image file> [--compress <compression algorithm>] [--encrypt <keyfile>] [--bundleversion <version>] [--productid <product id>] [--minversion <version>] [--sign <keyfile>] [--certificate <certificate file>] [--device <device>]
```

Command Line Input Example

Copyright © 2026 Silicon Laboratories. All rights reserved.




Command Line Output Example

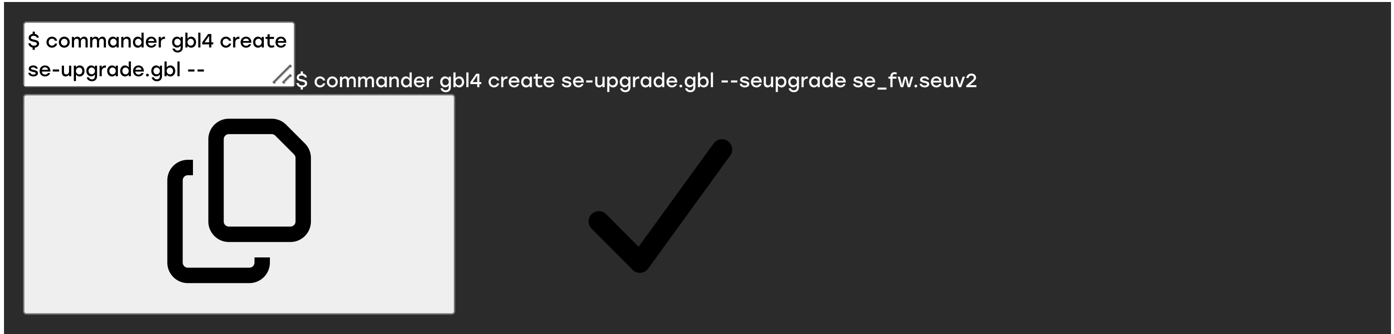



## Creating a GBL File for Secure Engine Upgrade

The Secure Engine can be upgraded using a Secure Engine upgrade binary provided by Silicon Labs. This command creates a GBL file containing a Secure Engine upgrade file and writes the output to the specified GBL filename.

Command Line Syntax

Command Line Input Example




Command Line Output Example

Copyright © 2026 Silicon Laboratories. All rights reserved.

## GBL4 Create Configuration File

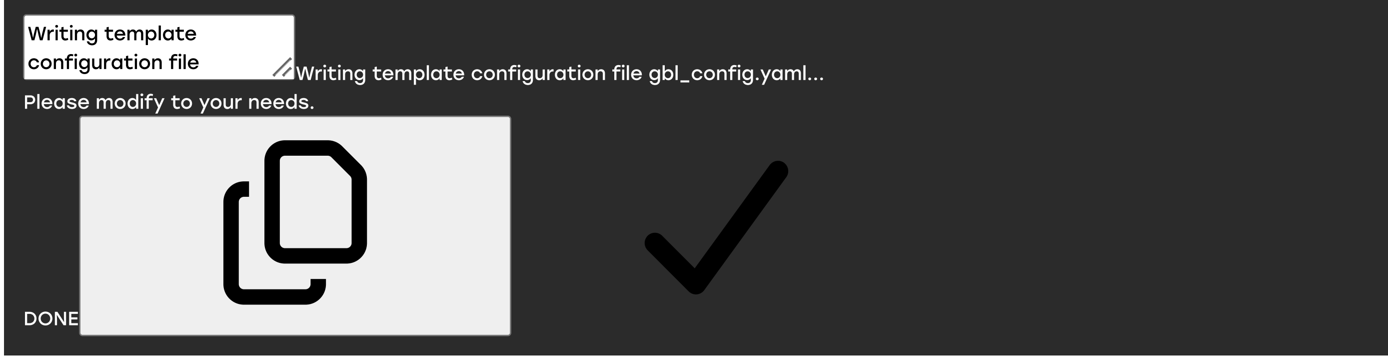
This command creates a default template configuration file for creating a GBL file. The configuration file is a YAML file that describes the GBL file to be created. See GBL4 Config File Format for details on the fields of the config file.

Command Line Syntax

Command Line Input Example

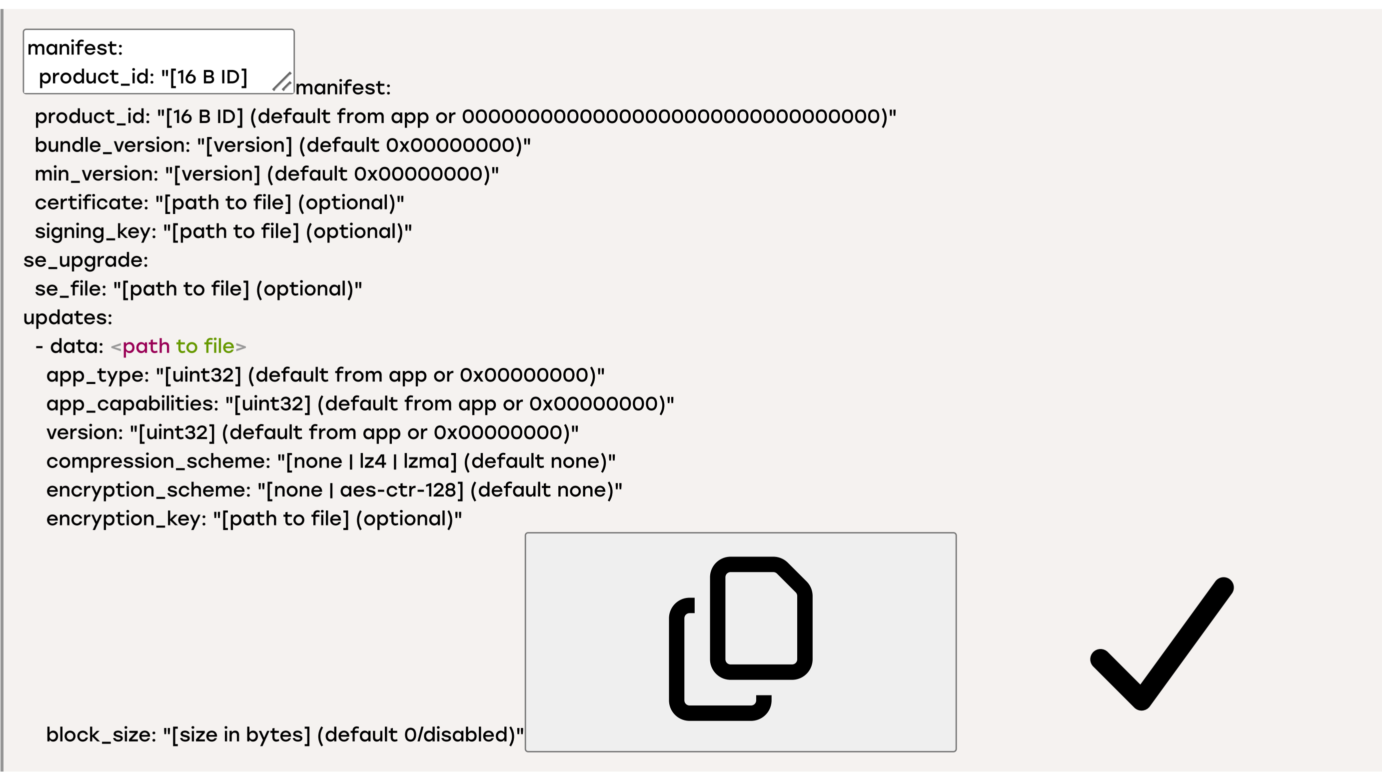
Command Line Output Example

Copyright © 2026 Silicon Laboratories. All rights reserved.



```
Writing template configuration file gbl_config.yaml...
Please modify to your needs.
DONE
```

Below is an example of the default configuration file:



```
manifest:
  product_id: "[16 B ID] (default from app or 00000000000000000000000000000000)"
  bundle_version: "[version] (default 0x00000000)"
  min_version: "[version] (default 0x00000000)"
  certificate: "[path to file] (optional)"
  signing_key: "[path to file] (optional)"
se_upgrade:
  se_file: "[path to file] (optional)"
updates:
  - data: <path to file>
    app_type: "[uint32] (default from app or 0x00000000)"
    app_capabilities: "[uint32] (default from app or 0x00000000)"
    version: "[uint32] (default from app or 0x00000000)"
    compression_scheme: "[none | lz4 | lzma] (default none)"
    encryption_scheme: "[none | aes-ctr-128] (default none)"
    encryption_key: "[path to file] (optional)"
    block_size: "[size in bytes] (default 0/disabled)"
```

## GBL4 File Information

Parse a Gecko Bootloader (GBL) file and shows information about the file.

Command Line Syntax

```
$ commander gbl4 info <gbl filename>
```

Copyright © 2026 Silicon Laboratories. All rights reserved.

Command Line Input Example

```
$ commander gbl4 info example.gbl
```

Command Line Output Example

```
GBL manifest info:
  GBL version:        4.0.0
  Encryption:         Enabled
  Compression:        Disabled
  Certificate:        Not used
  Signature type:     ECDSA-P256
  Product version:    00000000000000000000000000000000
  Bundle version:     0x00000000
  Min previous version: 0x00000000

Update info:
  Target address:           0x01008000
  Version:                  0x00000001
  Binary size:              43916 B
  Compression scheme:       None
  Encryption scheme:        AES-CTR-128
  Secure boot scheme:       Appinfo
  Signature block size:     0
  Number of blocks:         0

DONE
```

Copyright © 2026 Silicon Laboratories. All rights reserved.

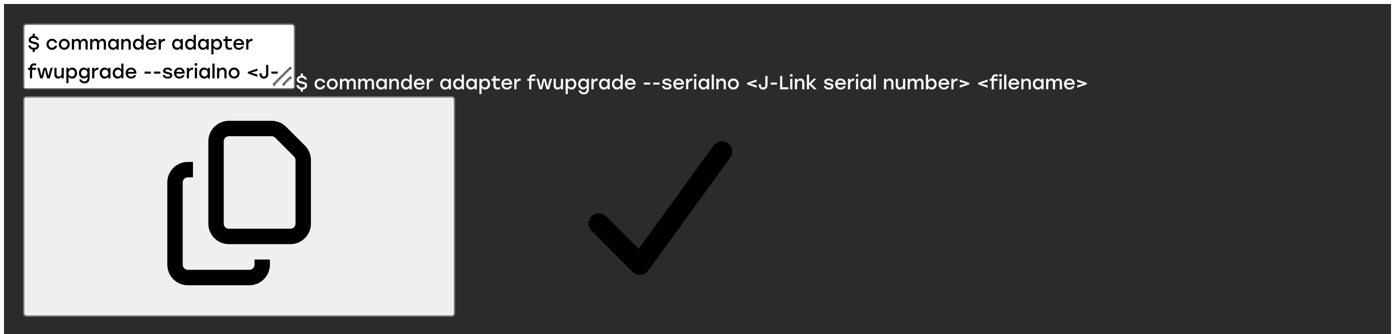
# Kit Utility Commands

# Kit Utility Commands

## Firmware Upgrade

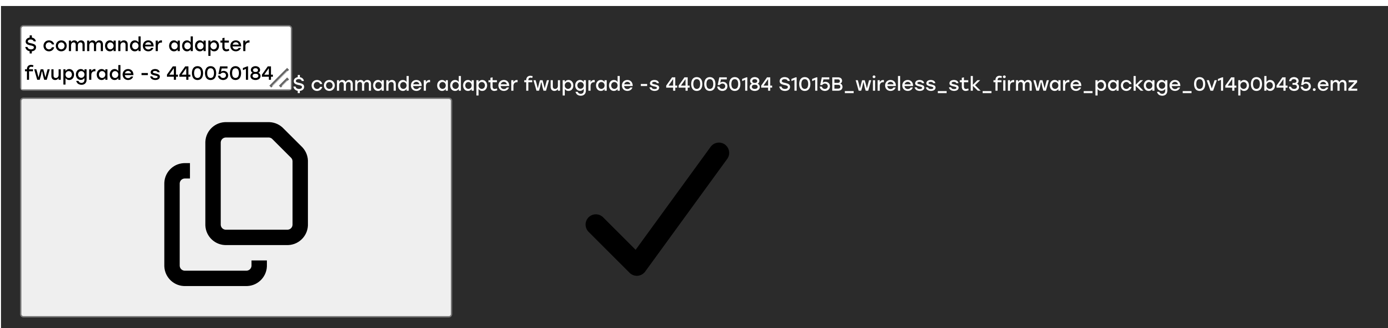
Updates the application running on the board controller on the kit to a new version provided in an .emz file by Silicon Labs.

Command Line Syntax



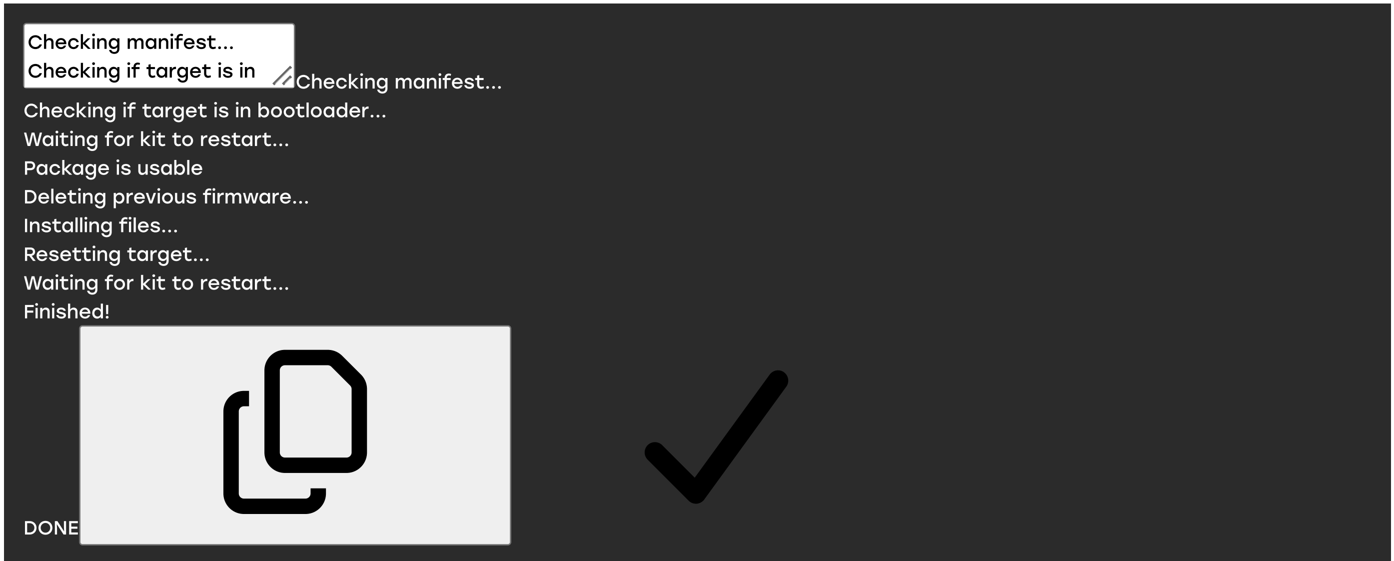
```
$ commander adapter fwupgrade --serialno <J-Link serial number> <filename>
```

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
```

Copyright © 2026 Silicon Laboratories. All rights reserved.

## Kit Information Probe

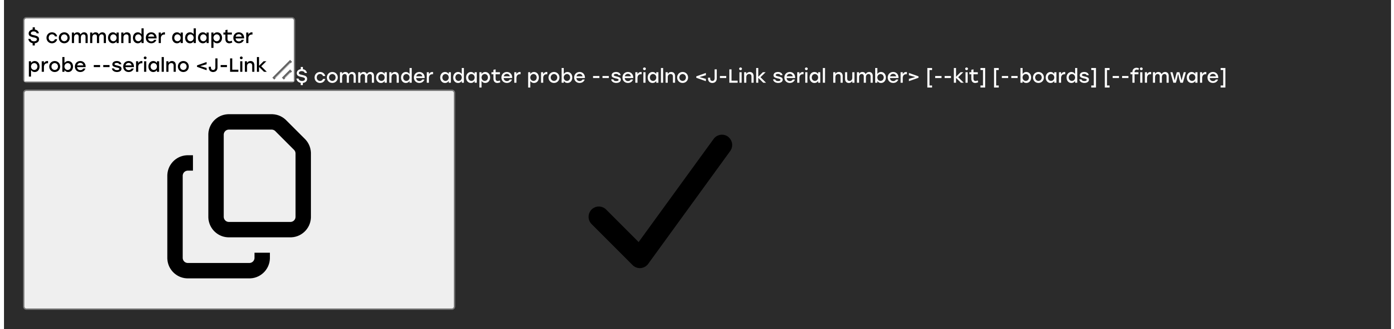
Retrieves information about a connected kit. Lists information about the kit part number and name, connected boards, and firmware version.

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

The `VCOM Port` line informs which virtual COM port name the kit has been assigned by the operating system. On Windows this is on the form COM<number>. On Linux and macOS, the name corresponds to a special file in the `/dev/` folder. E.g. `VCOM Port: ttyACM0` indicates that the serial port is available at `/dev/ttyACM0`. This line is not always available, and may be omitted from the output.

The nickname, IP address and MAC address of the adapter may be omitted from the output if the information is not available to Simplicity Commander.

Command Line Syntax



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

Command Line Input Example

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

Command Line Usage Output

Copyright © 2026 Silicon Laboratories. All rights reserved.

```
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
AEM Supported   : 1
VCOM Supported  : 1
IP Supported    : 1
VCOM Port       : COM3
Nickname        : Lab-01
IP Address      : 10.5.103.23
MAC Address     : D0:CF:5E:C0:FF:EE
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
Target Device   : EFR32MG1P233F256GM48-C0
DONE
```

## 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.

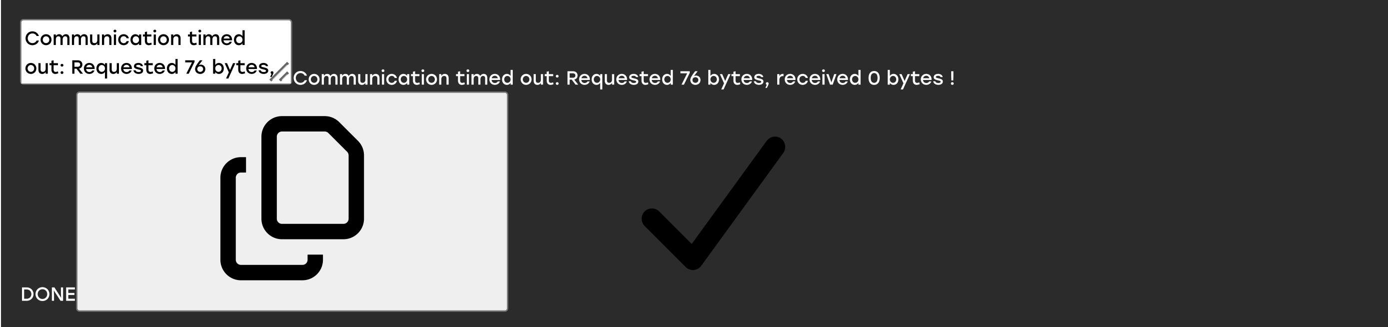
Command Line Syntax

```
$ commander adapter reset
```

Command Line Input Example

```
$ commander adapter reset
```

Command Line Output Example

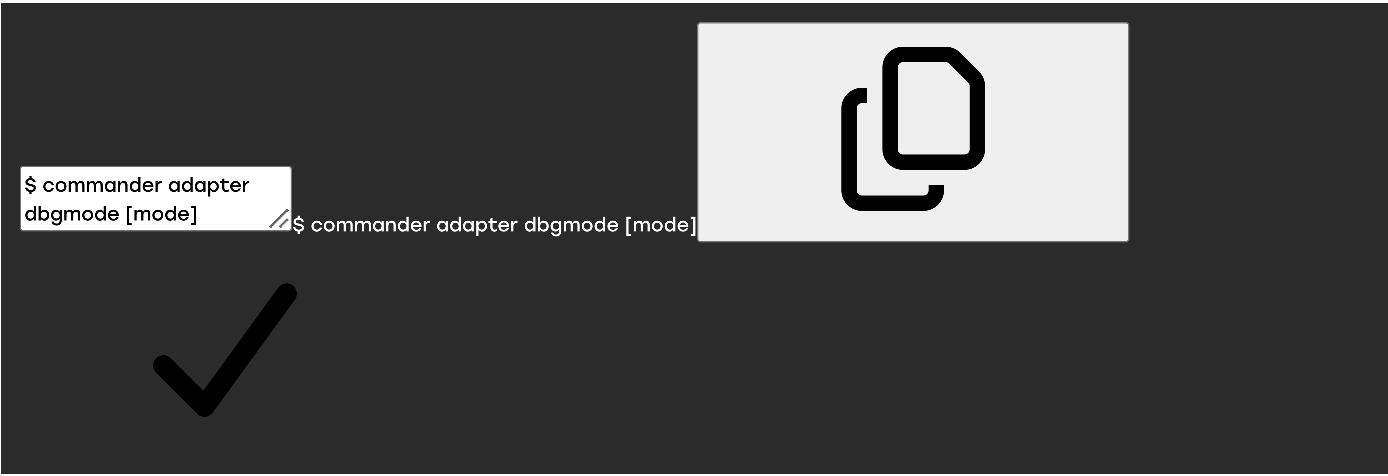


```
Communication timed out: Requested 76 bytes, received 0 bytes !
DONE
```

## 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 in addition to MINI for Wireless Pro Kits and TARGET for Development Kits. See the *Quick Start Guide* for your kit for a description of the debug modes it supports.

Command Line Syntax



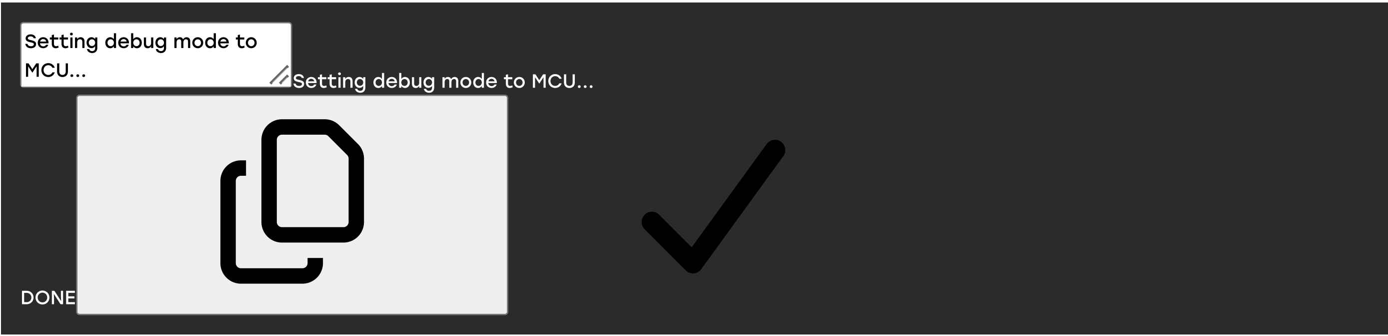
```
$ commander adapter dbgmode [mode]
```

Command Line Input Example

Copyright © 2026 Silicon Laboratories. All rights reserved.

```
$ commander adapter dbgmode MCU
```

Command Line Output Example



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

## List Adapter IP Configuration Command

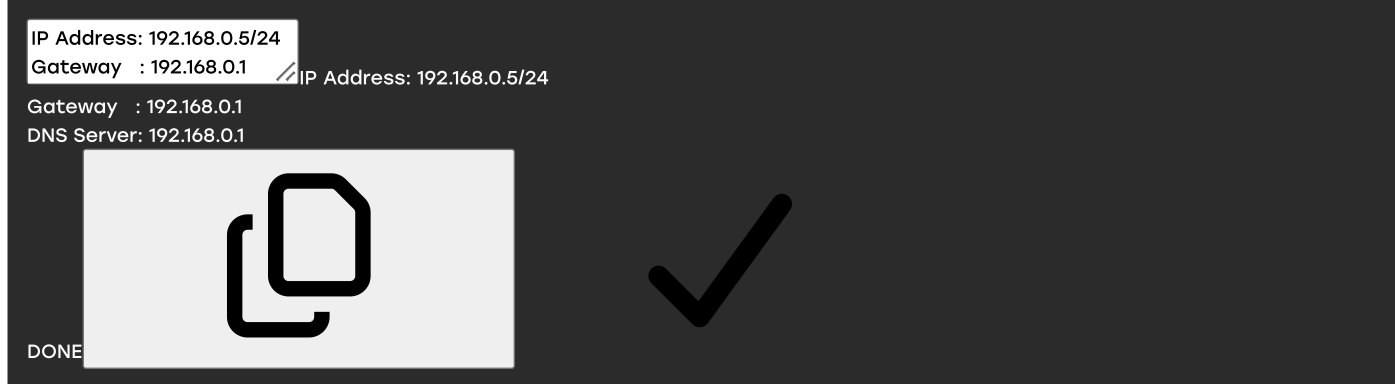
The `adapter ip` command gets or sets the IP configuration of the adapter. With no options, the current configuration is retrieved and displayed.

Command Line Syntax

```
$ commander adapter ip
```

Command Line Input Example

Copyright © 2026 Silicon Laboratories. All rights reserved.



```
IP Address: 192.168.0.5/24
Gateway   : 192.168.0.1
DNS Server: 192.168.0.1
DONE
```

## Adapter DHCP Command

This command sets up the adapter to use DHCP to automatically retrieve IP, gateway and DNS addresses. This is the default con-figuration. After enabling DHCP, the adapter must be restarted for the change to take effect.

Command Line Syntax

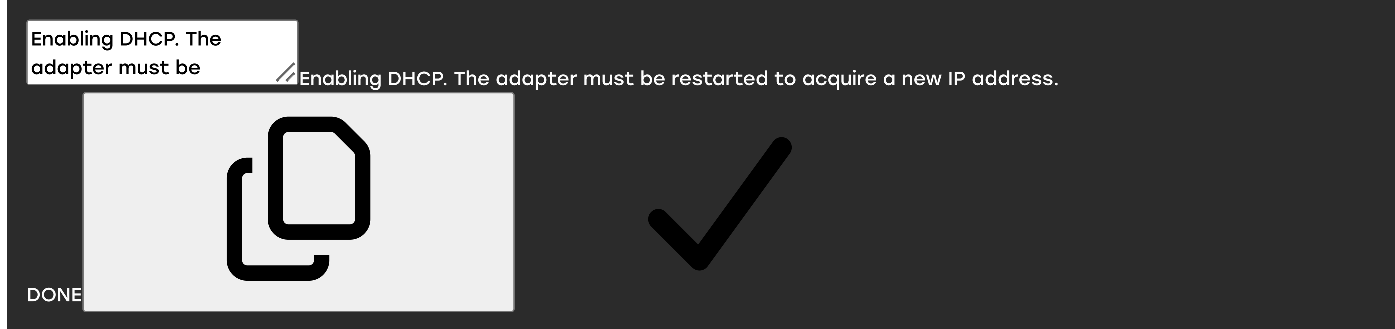
```
$ commander adapter ip --dhcp
```

Command Line Input Example

```
$ commander adapter ip --dhcp
```

Command Line Output Example

Copyright © 2026 Silicon Laboratories. All rights reserved.



```
Enabling DHCP. The adapter must be restarted to acquire a new IP address.
DONE
```

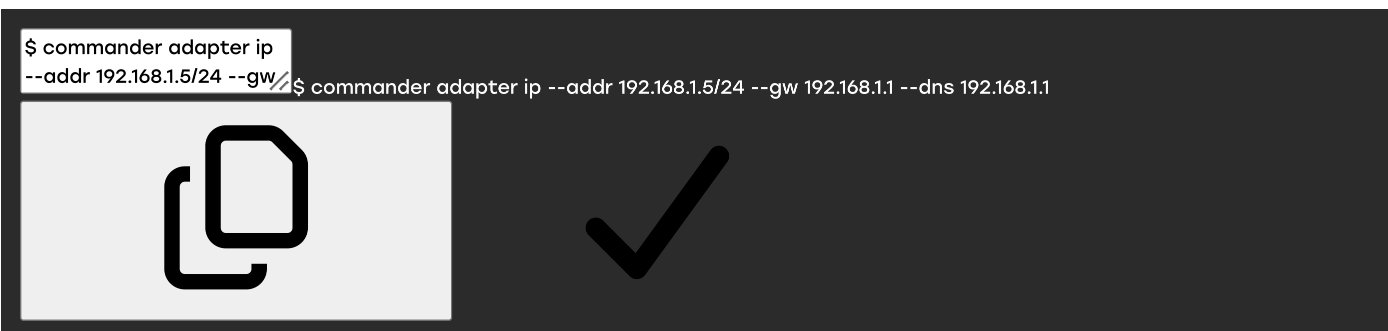
## Set Static IP Configuration Command

This command sets the IP address of the adapter in Classless Inter-Domain (CIDR) notation.

Command Line Syntax

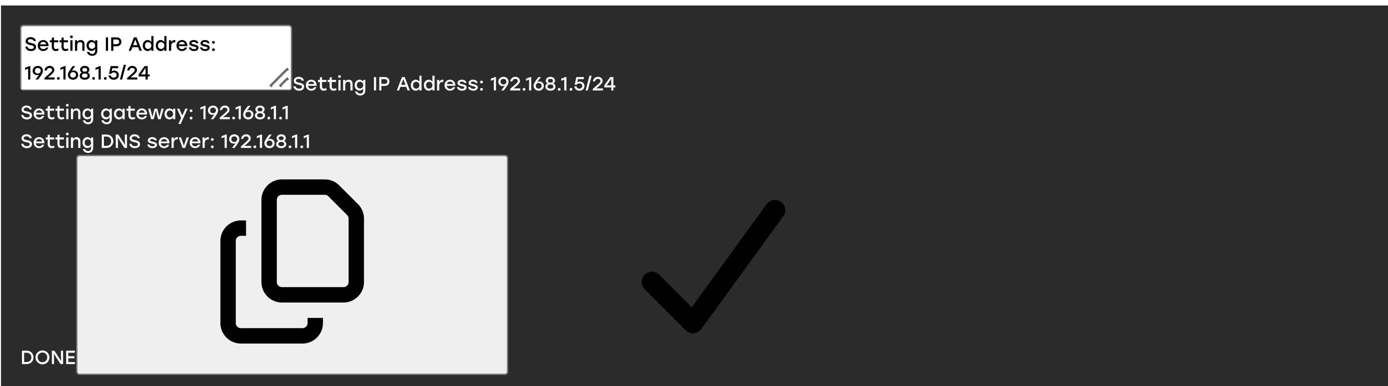
```
$ commander adapter ip --addr <IP address/prefix> [--gw <gateway address>] [--dns <dns server address>]
```

Command Line Input Example



```
$ commander adapter ip --addr 192.168.1.5/24 --gw 192.168.1.1 --dns 192.168.1.1
```

Command Line Output Example



```
Setting IP Address: 192.168.1.5/24
Setting gateway: 192.168.1.1
Setting DNS server: 192.168.1.1
DONE
```

## Get or Change Adapter Nickname

You can get, set and clear the adapter's nickname using the `adapter nick` command.

Copyright © 2026 Silicon Laboratories. All rights reserved.

If no new nickname is provided, the adapter's current nickname will be displayed.

If you provide a new nickname, this will be stored in the adapter.

Providing the `--clear` option will clear the adapter's stored nickname.

Command Line Syntax

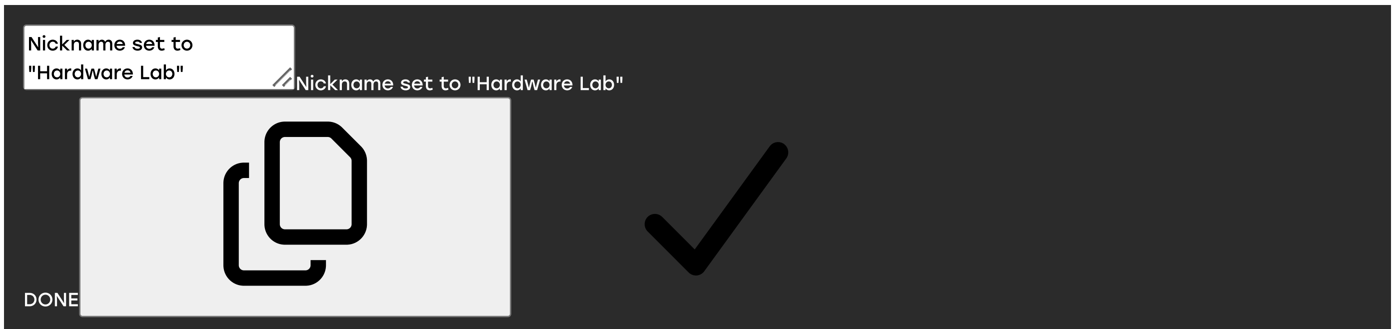
```
$ commander adapter nick [nickname] [--clear]
```

Command Line Input Example

```
$ commander adapter nick "Hardware Lab"
```

This command line sets the adapter nickname to "Hardware Lab".

Command Line Output Example



```
Nickname set to "Hardware Lab"
DONE
```

## Get or Change Target Voltage

You can get and set the voltage that the adapter supplies to the target device using the `adapter voltage` command.

If no voltage is provided, the currently configured voltage and the measured voltage will be displayed.

Copyright © 2026 Silicon Laboratories. All rights reserved.

If a voltage is provided, the voltage will be set. This setting does not persist across adapter reboots. After the voltage has been changed, the Advanced Energy Monitor (AEM) will automatically calibrate itself (see Calibrate the Advanced Energy Monitor for more details). Providing the option `--nocalibrate` will skip the automatic calibration.

> Note: Changing the target voltage is not supported on all adapter boards.

Command Line Syntax

```
$ commander adapter voltage [voltage] [--nocalibrate]
```

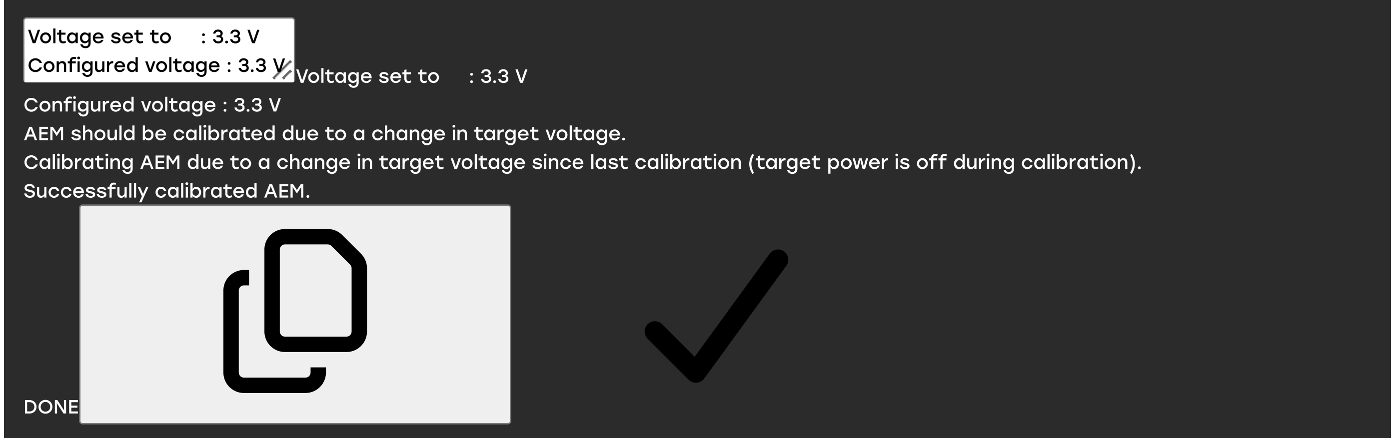
Command Line Input Example

```
$ commander adapter voltage 3.3
```

This command line sets the target voltage to 3.3 V, allowing for the AEM to calibrate itself after the new voltage has been set.

Command Line Output Example

Copyright © 2026 Silicon Laboratories. All rights reserved.




## Get or Change Target Power

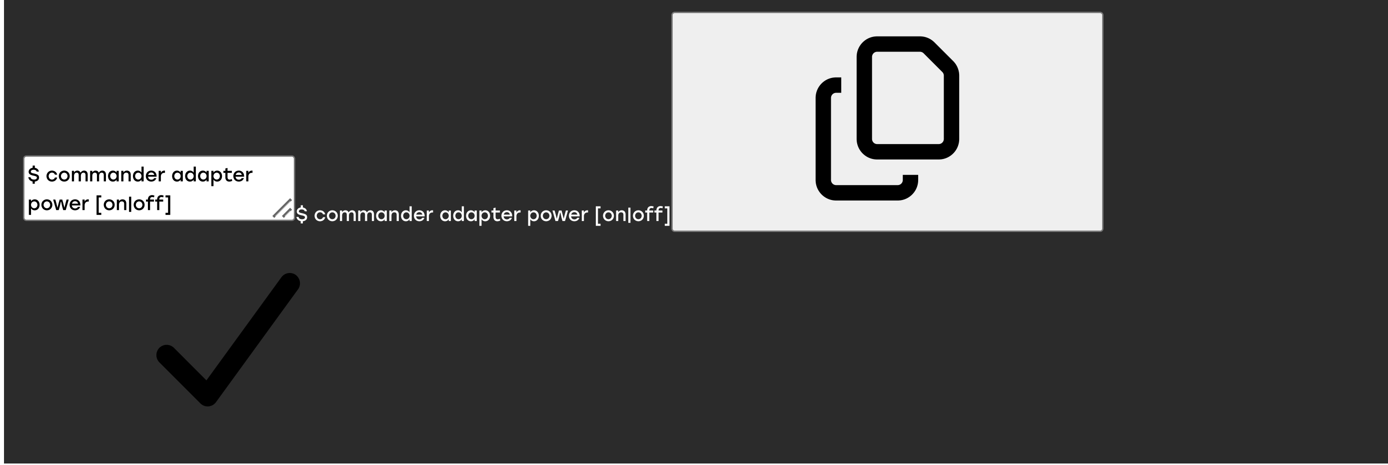
You can get and set the power state of the target device using the `adapter power` command.

If no argument is provided, the current target power state will be displayed.

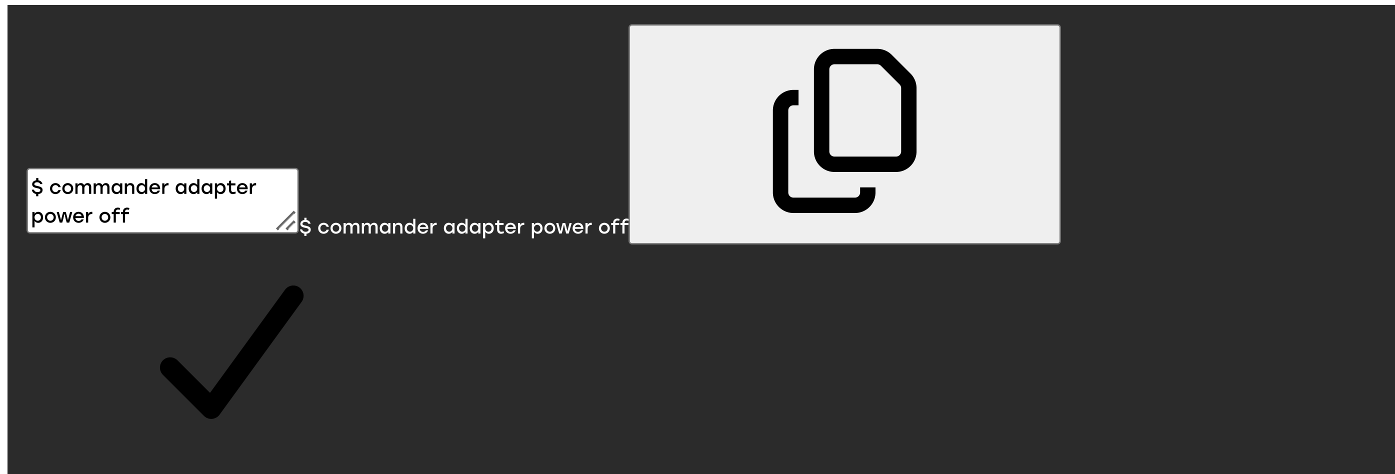
If 'on' or 'off' is provided, the target power will be enabled or disabled, respectively. This setting does not persist across adapter reboots.

> Note: Controlling target power is not supported on all adapter boards.

Command Line Syntax



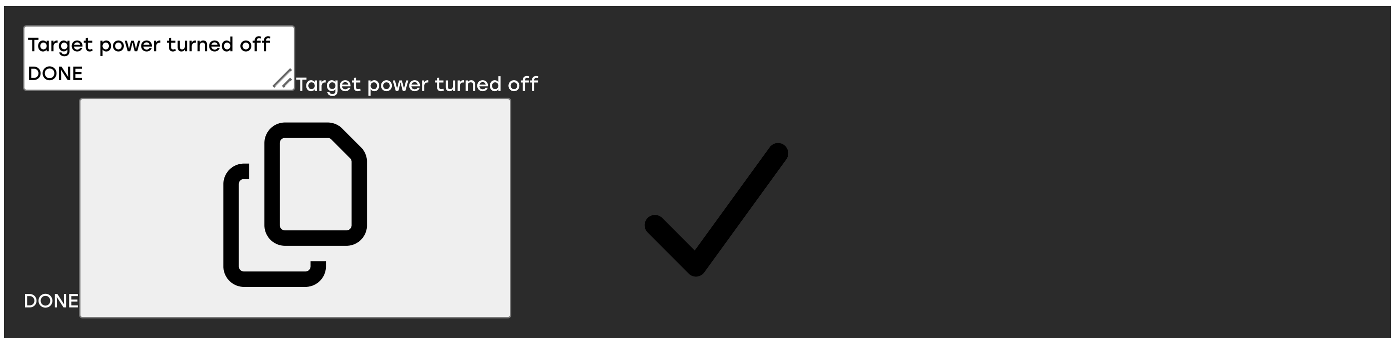

Command Line Input Example



```
$ commander adapter power off
```

This command line takes the RPS image 'app.rps' and loads it onto the device.

Command Line Output Example



```
Target power turned off
DONE
```

Copyright © 2026 Silicon Laboratories. All rights reserved.

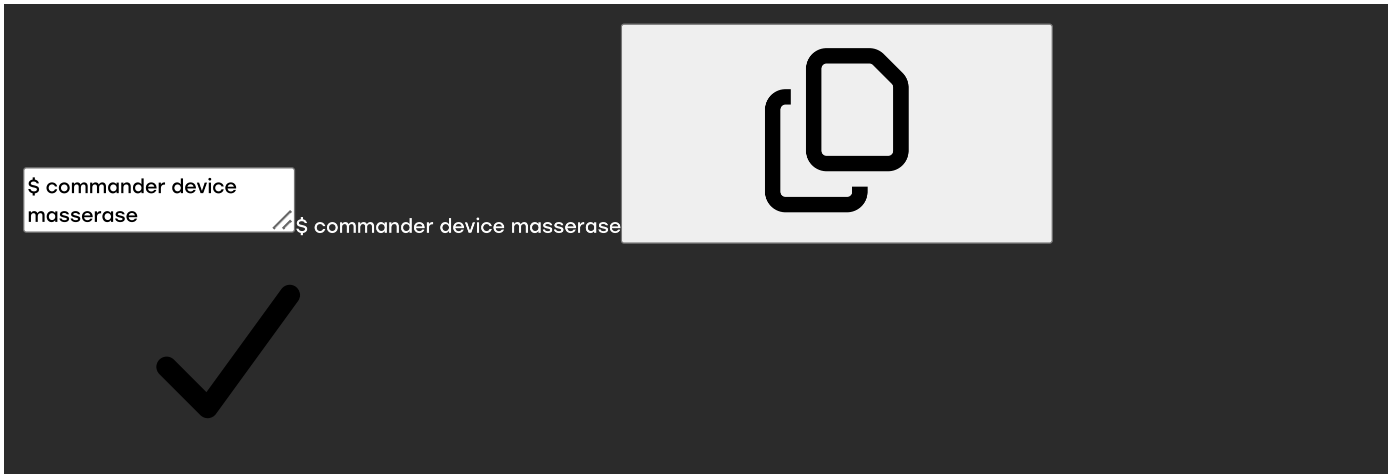
# Device Erase Commands

# Device Erase Commands

## 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 Syntax



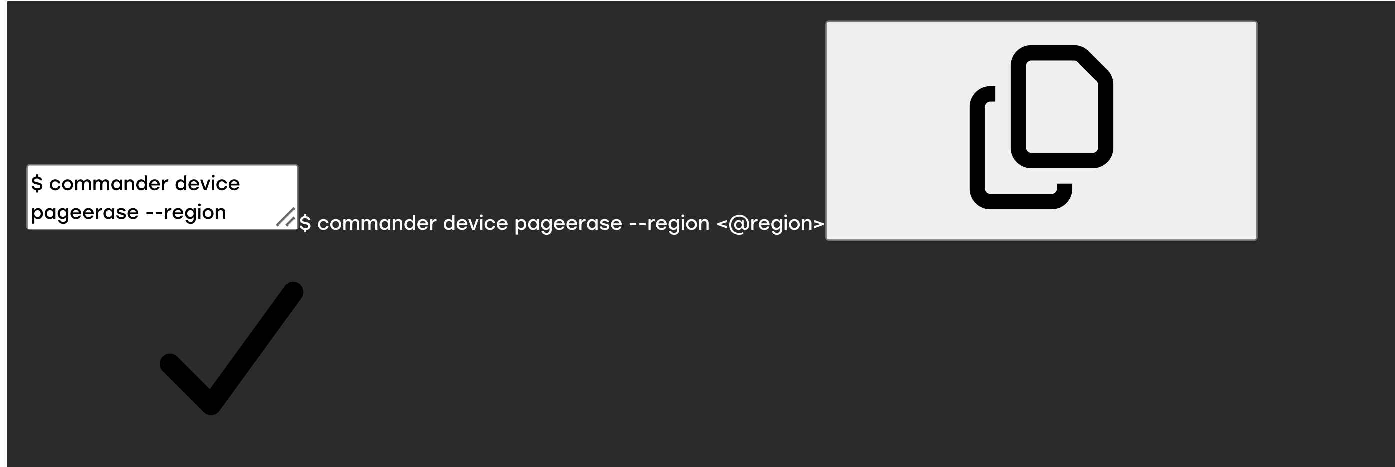

Command Line Usage Output

## Erase Region

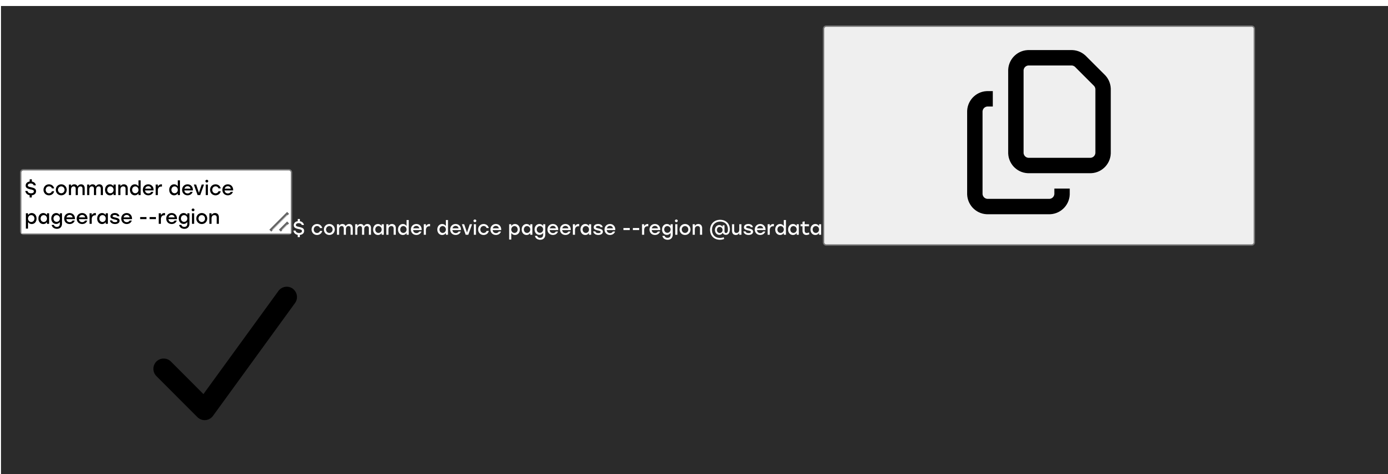
Erases a named region. For more information on the `--region` option, see Flash Verification Command.

Command Line Syntax

Copyright © 2026 Silicon Laboratories. All rights reserved.




Command Line Input Example




Command Line Output Example

## Erase Pages in Address Range

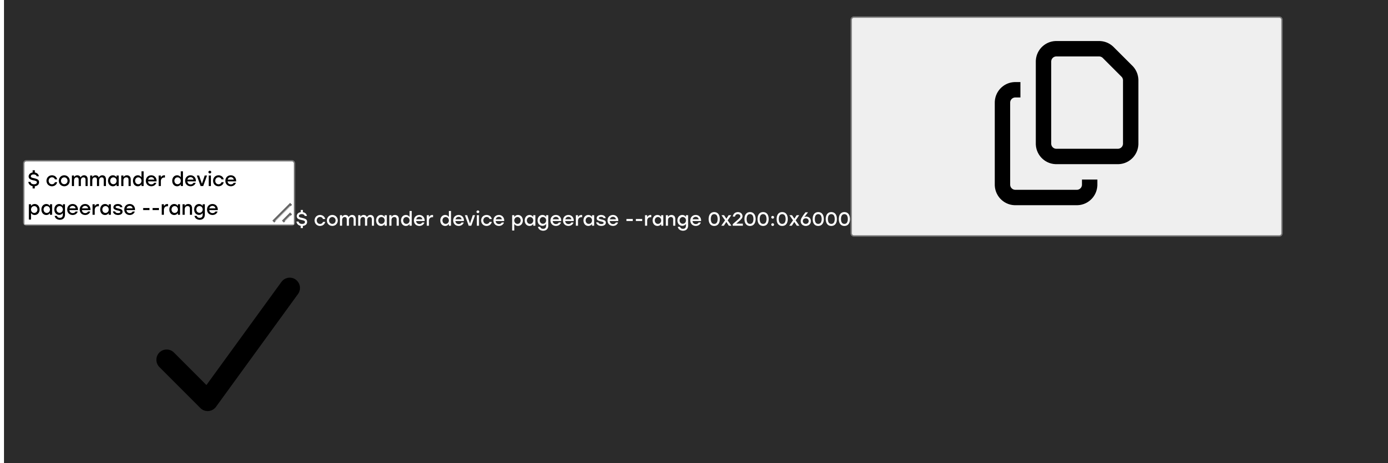
Erases all flash pages affected by the given memory range. If the given range doesn't match page boundaries, it will be extended to always erase entire pages.

Command Line Syntax

Copyright © 2026 Silicon Laboratories. All rights reserved.

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

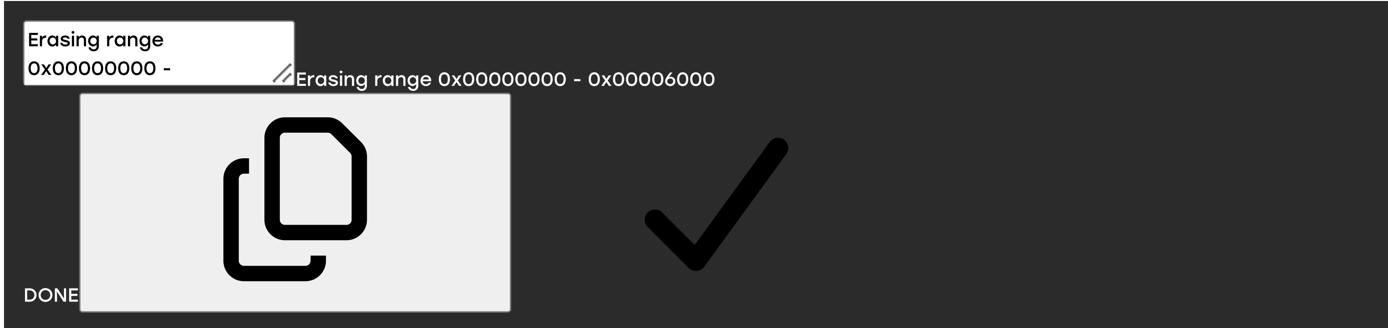
Command Line Input Example



```
$ 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
```

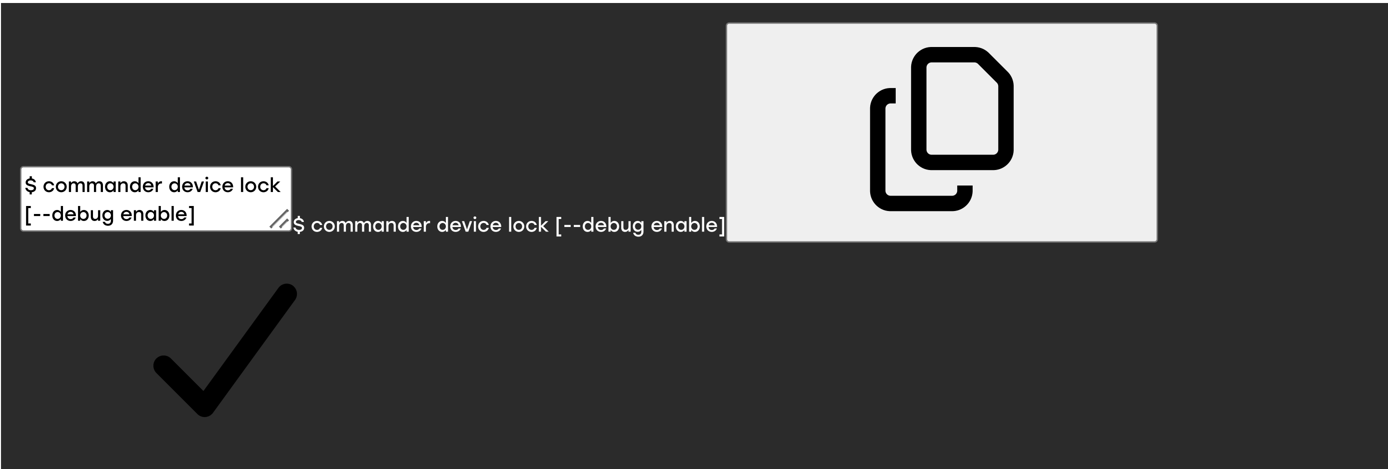
Device Lock and Protection Commands

# Device Lock and Protection Commands

## Debug Lock

Locks access to the debug interface of the device. This feature is only supported on EFM32 and EFR32 devices. The `--debug enable` option is no longer required as of Simplicity Commander version 1.8.

Command Line Syntax



```
$ commander device lock [--debug enable]
```

Command Line Usage Output

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

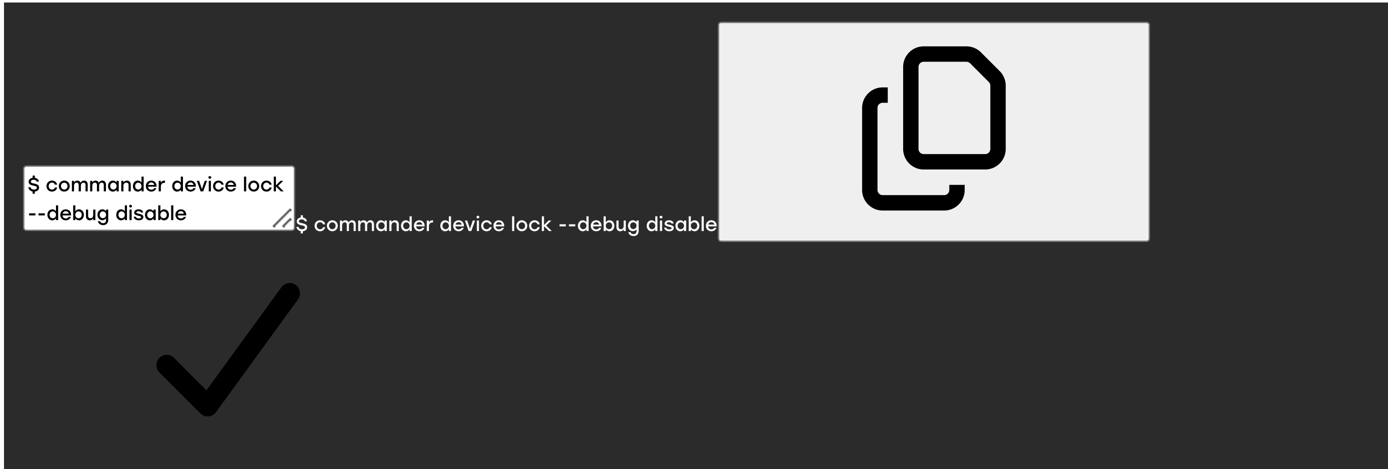
## Debug Unlock

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

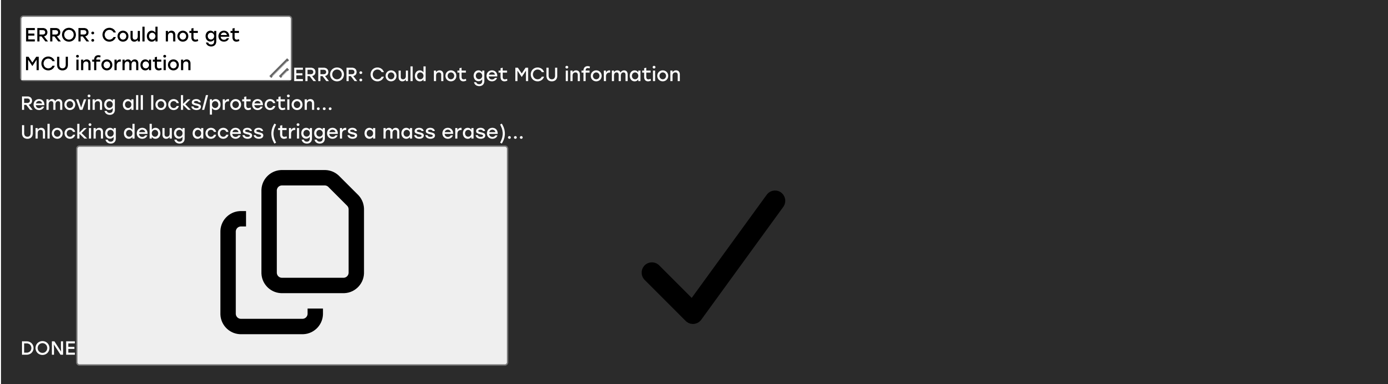
This feature is only supported on EFM32 and EFR32 devices.

Command Line Syntax

Copyright © 2026 Silicon Laboratories. All rights reserved.

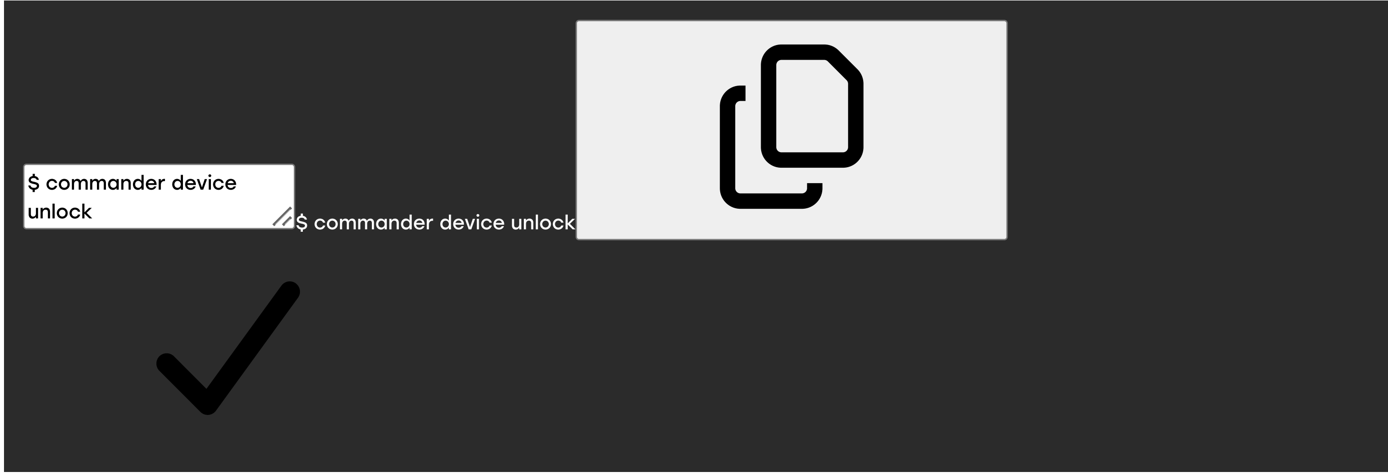



Command Line Usage Output




In Simplicity Commander version 1.8, an alternative command syntax was introduced.

Command Line Syntax




Command Line Usage Output

## Write Protect Flash Ranges

Protects all flash pages affected by the given memory range from any writes or erases. The available granularity of flash write protection is device-dependent. Consult the device reference manual for details. For EFM32 and EFR32 devices, for example, the write protect feature operates on flash pages. On EM3xx devices, this works on 8 kB or 16 kB blocks.

For all devices, if the given range doesn't match the block size supported by the device, it will be extended to always protect entire regions.

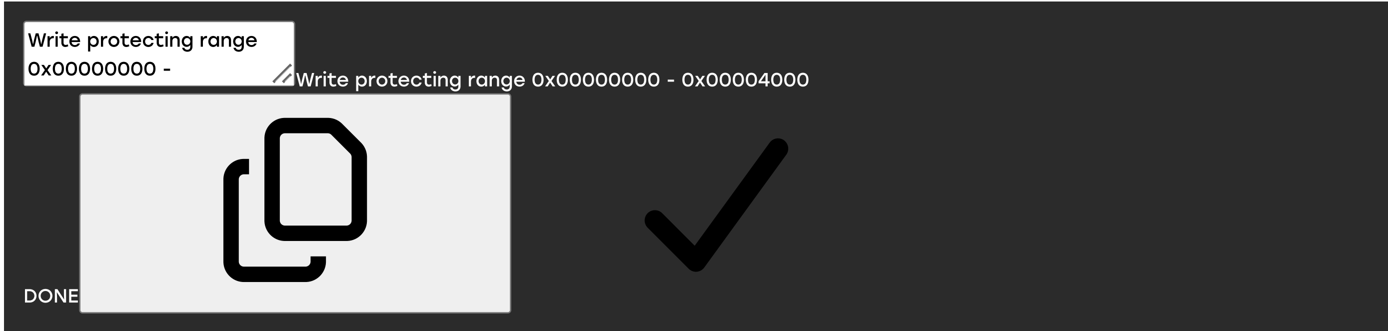
Command Line Syntax

Command Line Input Example

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

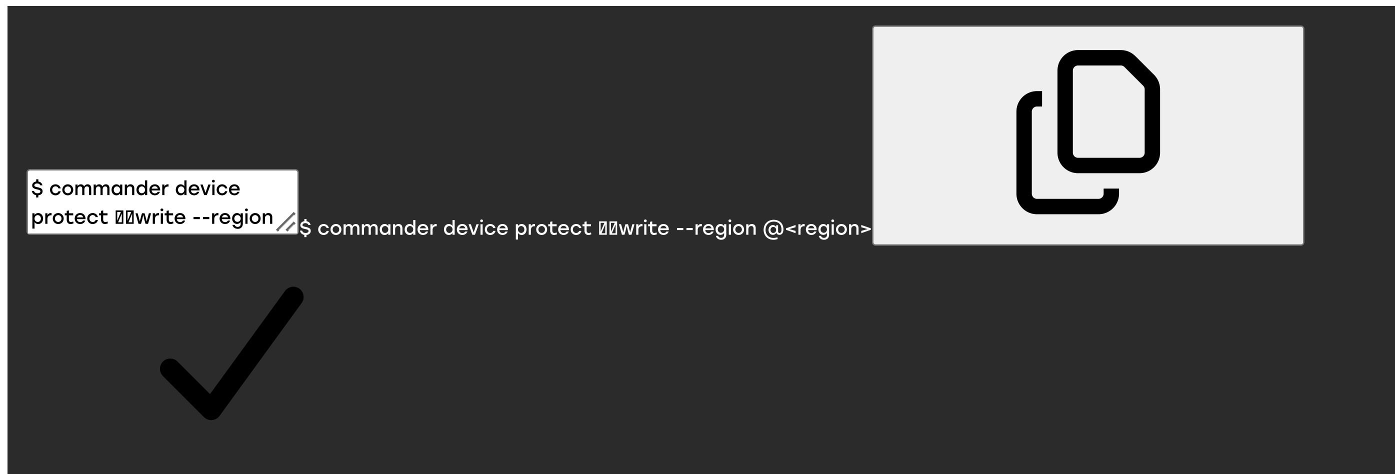
Copyright © 2026 Silicon Laboratories. All rights reserved.



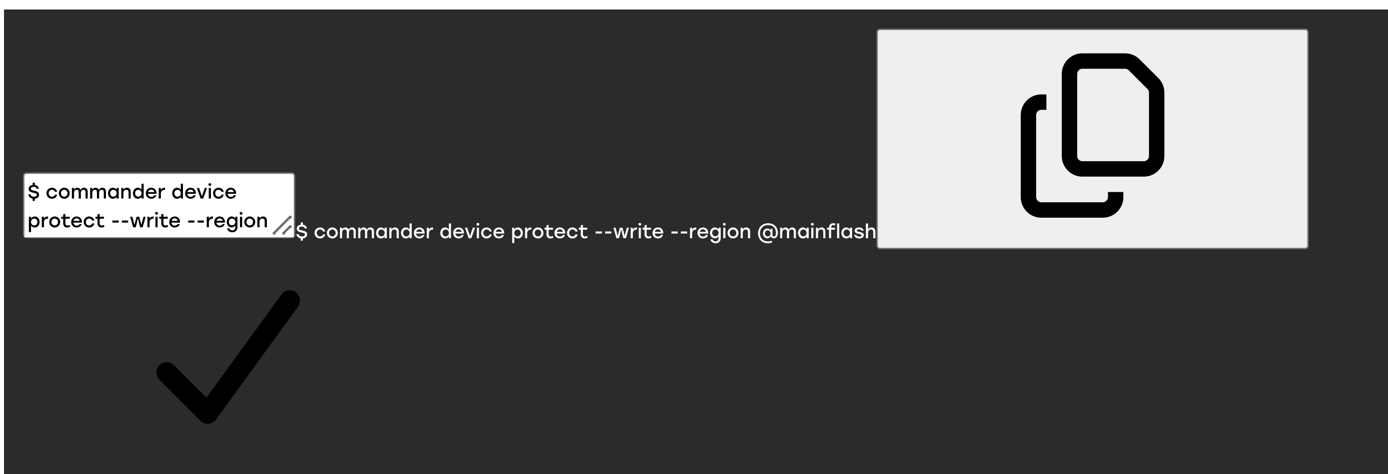

## Write Protect Flash Region

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

Command Line Syntax



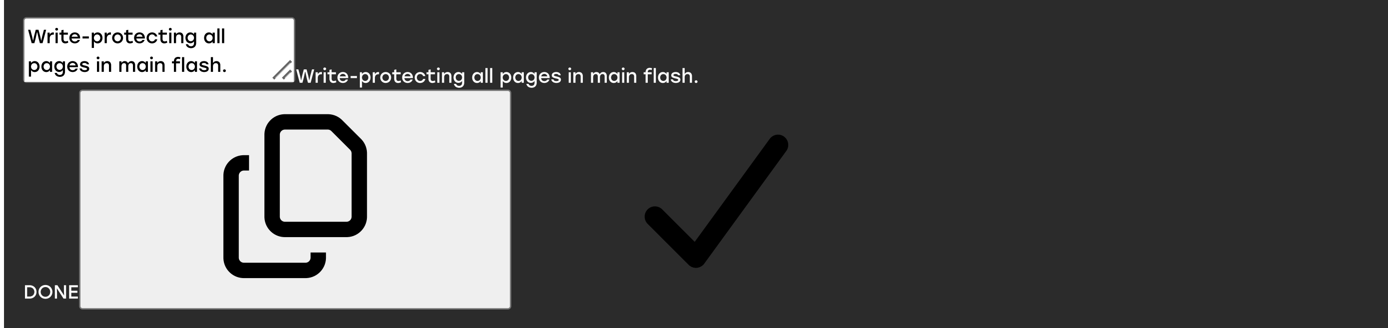

Command Line Input Example




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

Command Line Output Example

Copyright © 2026 Silicon Laboratories. All rights reserved.

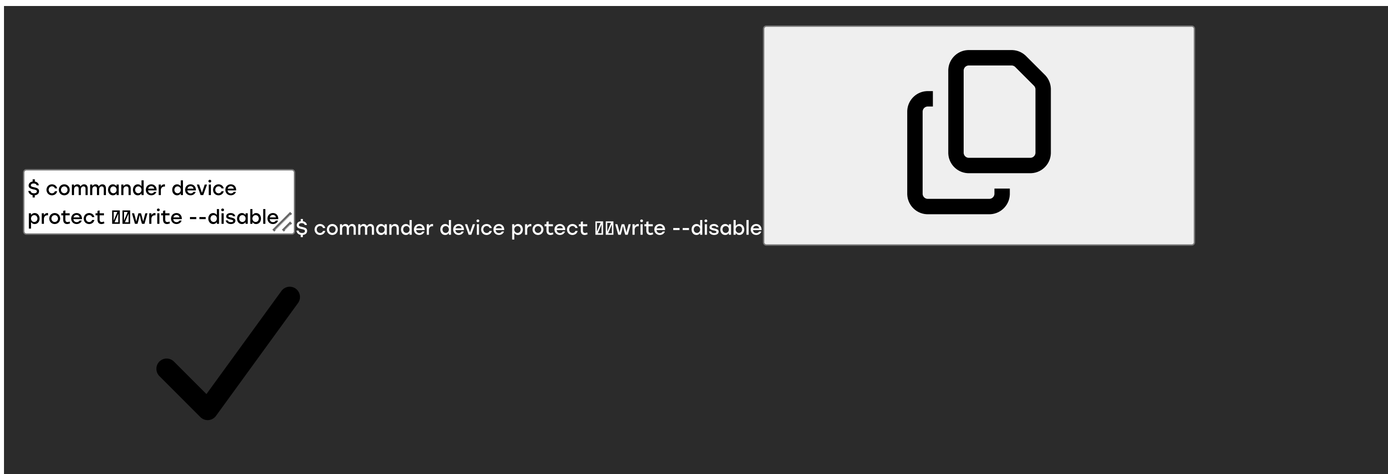


```
Write-protecting all pages in main flash.
DONE
```

## Disable Write Protection

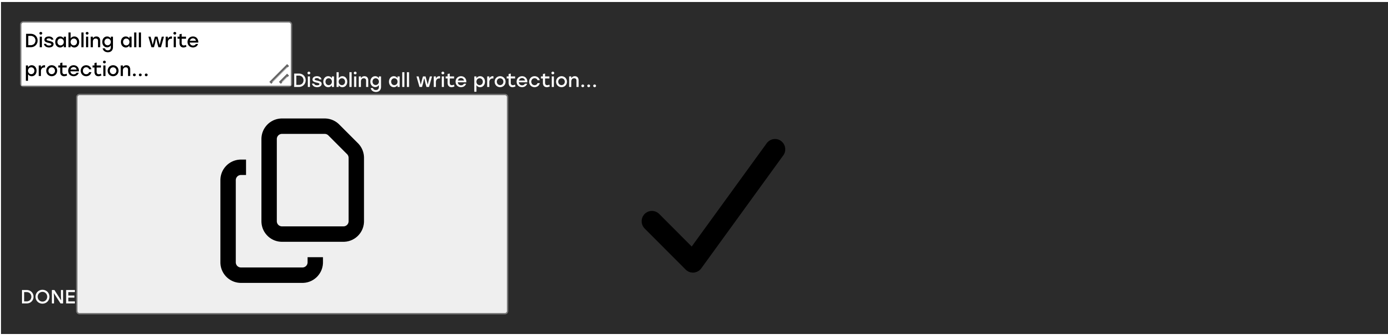
Disables write protection for all pages.

Command Line Syntax



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

Command Line Output Example



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

Copyright © 2026 Silicon Laboratories. All rights reserved.

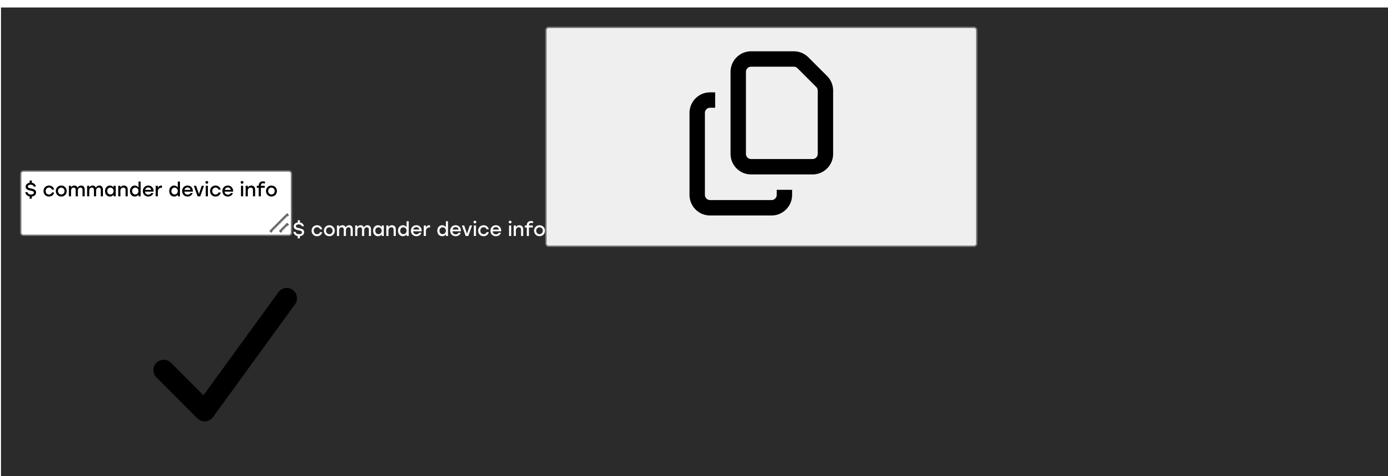
# Device Utility Commands

# Device Utility Commands

## Device Information Command

Shows detailed information about the target device.

Command Line Syntax



```
$ commander device info
```

Command Line Usage Output

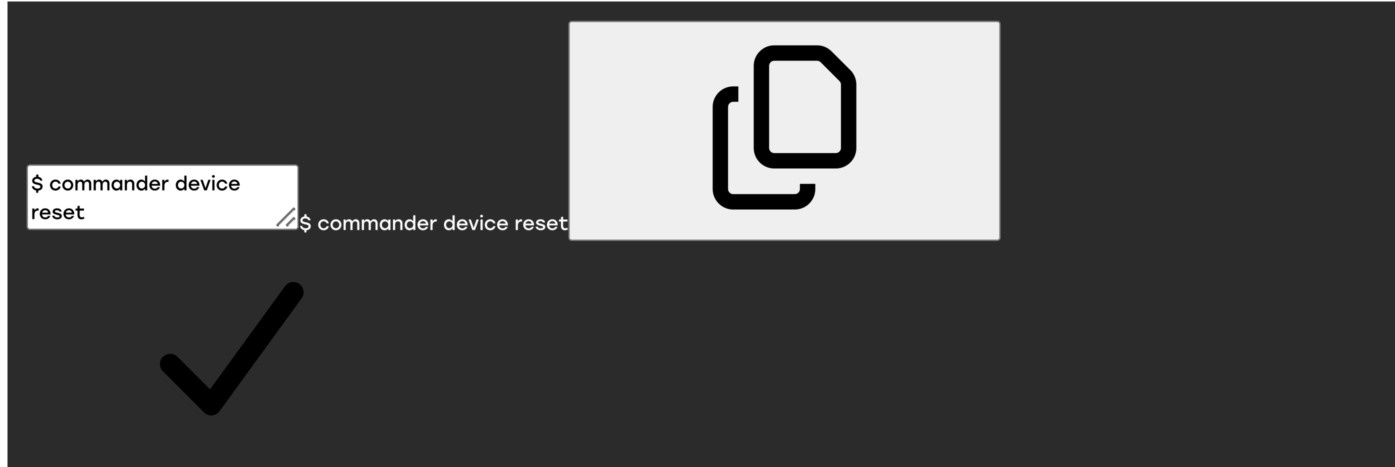
```
Part Number    : EFR32MG1P233F256GM48
Die Revision   : A0
Production Ver : 0
Flash Size     : 256 kB
SRAM Size      : 32 kB
Unique ID      : 000b57000003b2f0
DONE
```

## Device Reset Command

Resets a device using a pin reset.

Command Line Syntax

Copyright © 2026 Silicon Laboratories. All rights reserved.



```
$ commander device reset
```

Command Line Usage Output

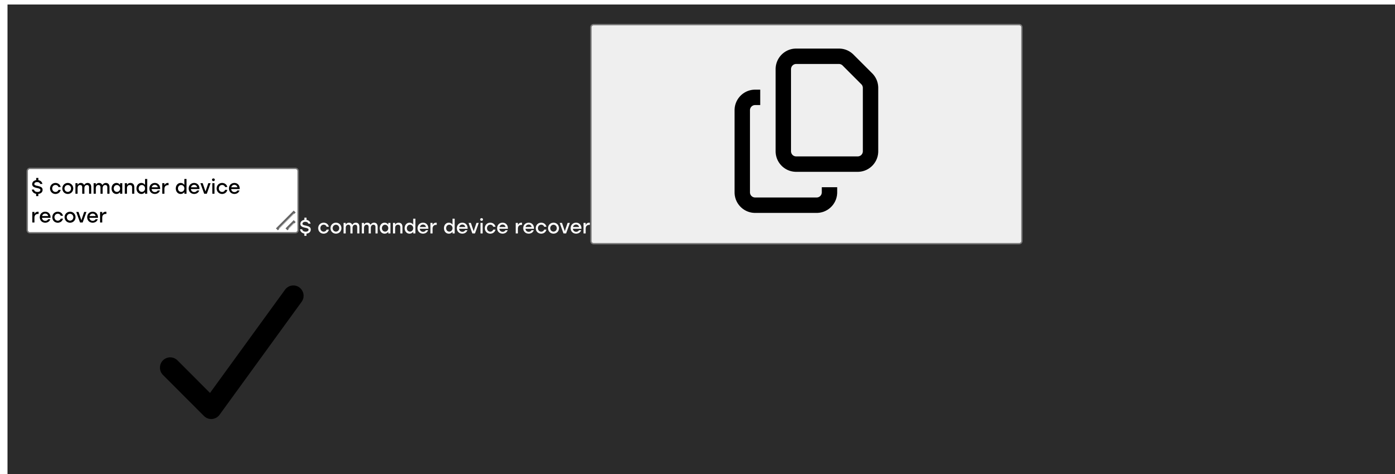
```
Resetting chip...
DONE
```

## Device Recovery Command

On EFM32 and EFR32 devices, this 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.

On EM3xx devices, this command can be used to recover from option byte failure.

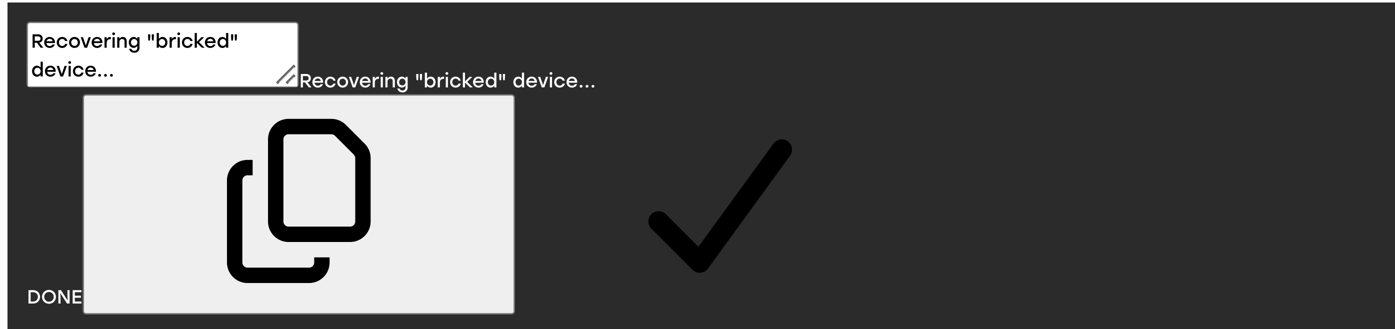
Command Line Syntax



```
$ commander device recover
```

Command Line Usage Output

Copyright © 2026 Silicon Laboratories. All rights reserved.

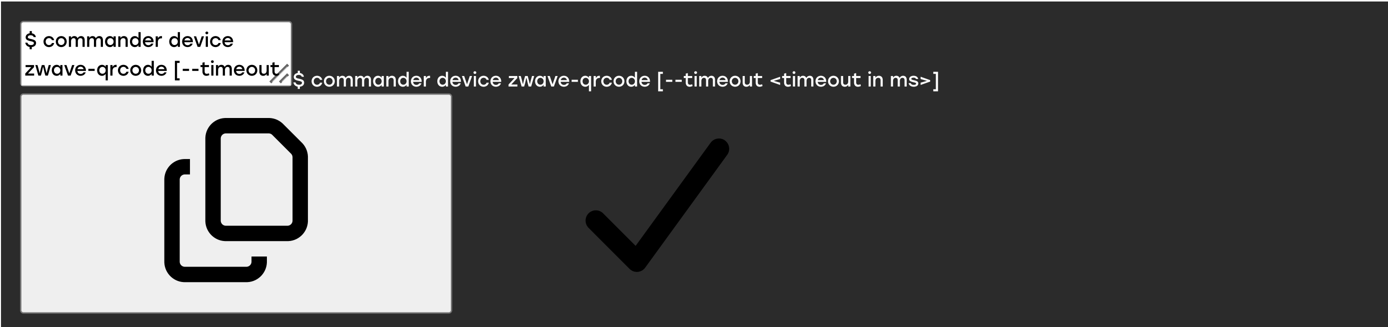



## Device Z-Wave QR Code Command

The Z-Wave QR code command is used to read out the QR code from all Z-Wave devices. The QR code is 90 bytes, displayed as ASCII characters, and stored in the TOKEN_MFG_ZW_QR_CODE manufacturing token.

The QR code is generated in the chip during initialization. When the QR code is correctly initialized, the value of the manufacturing token TOKEN_MFG_ZW_INITIALIZED is changed from 0xFF to 0x00. The optional `--timeout` option is used to indicate how long Simplicity Commander should wait for the QR code to be initialized. If no time is given, the default is 5000 ms.

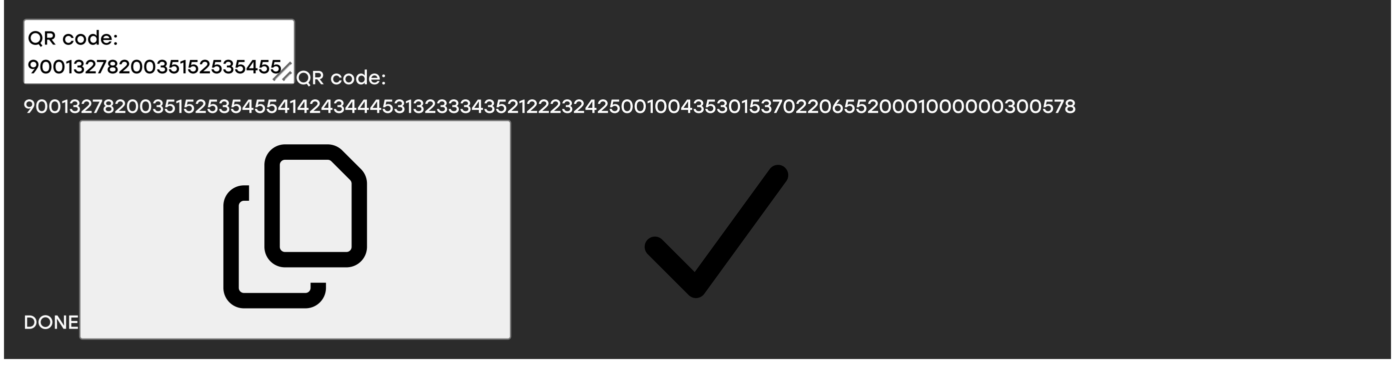
Command Line Syntax




Command Line Input Example

Command Line Usage Output

Copyright © 2026 Silicon Laboratories. All rights reserved.




Copyright © 2026 Silicon Laboratories. All rights reserved.

External SPI Flash Commands

# 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:

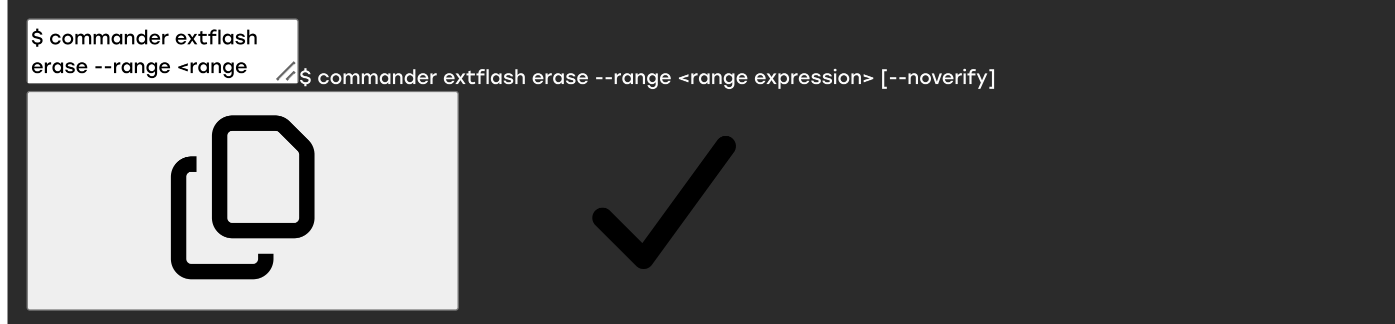
- The integrated SPI flash on EFR32MG1x632 and EFR32MG1x732 devices
- The MX25 SPI flash on EFR32 radio boards

## 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 Syntax

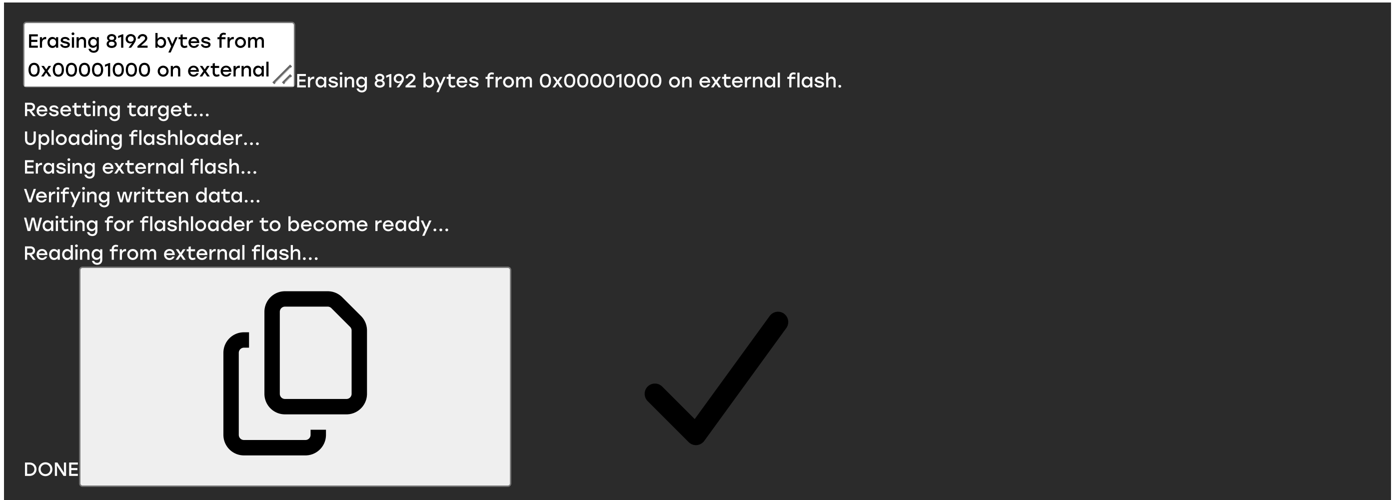


```
$ commander extflash erase --range <range expression> [--noverify]
```

Command Line Input Example

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

Command Line Output Example

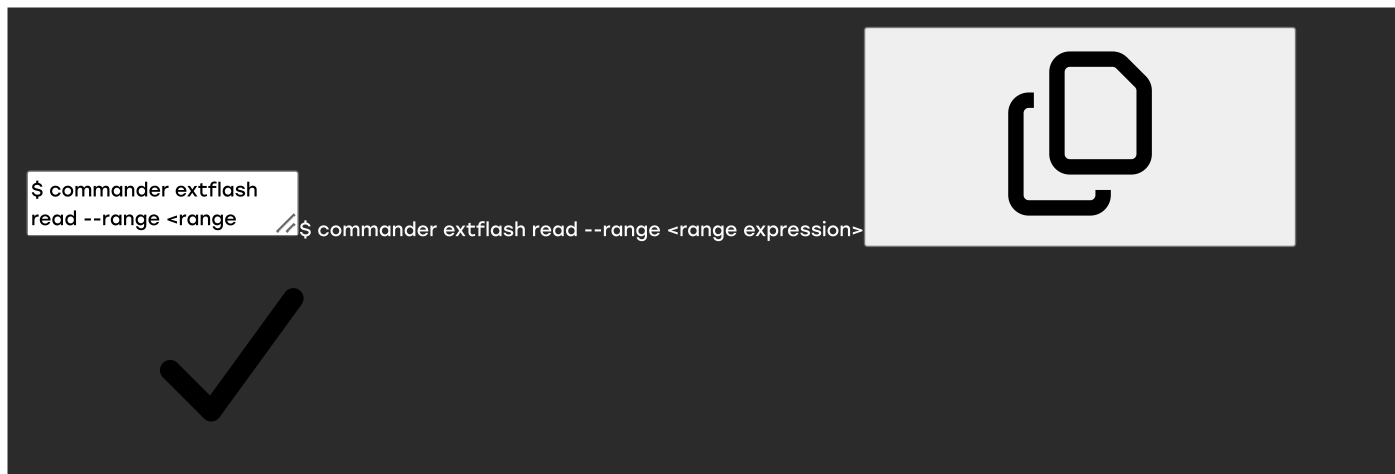


```
Erasing 8192 bytes from 0x00001000 on external flash.
Resetting target...
Uploading flashloader...
Erasing external flash...
Verifying written data...
Waiting for flashloader to become ready...
Reading from external flash...
DONE
```

## Read External SPI Flash Command

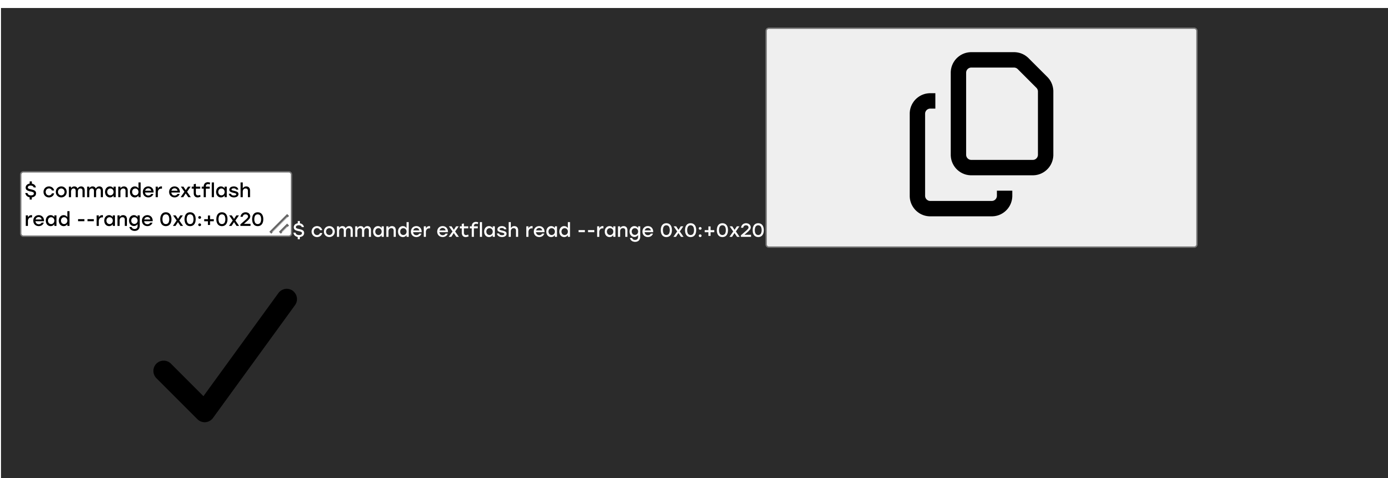
Use this command to read from external flash.

Command Line Syntax



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

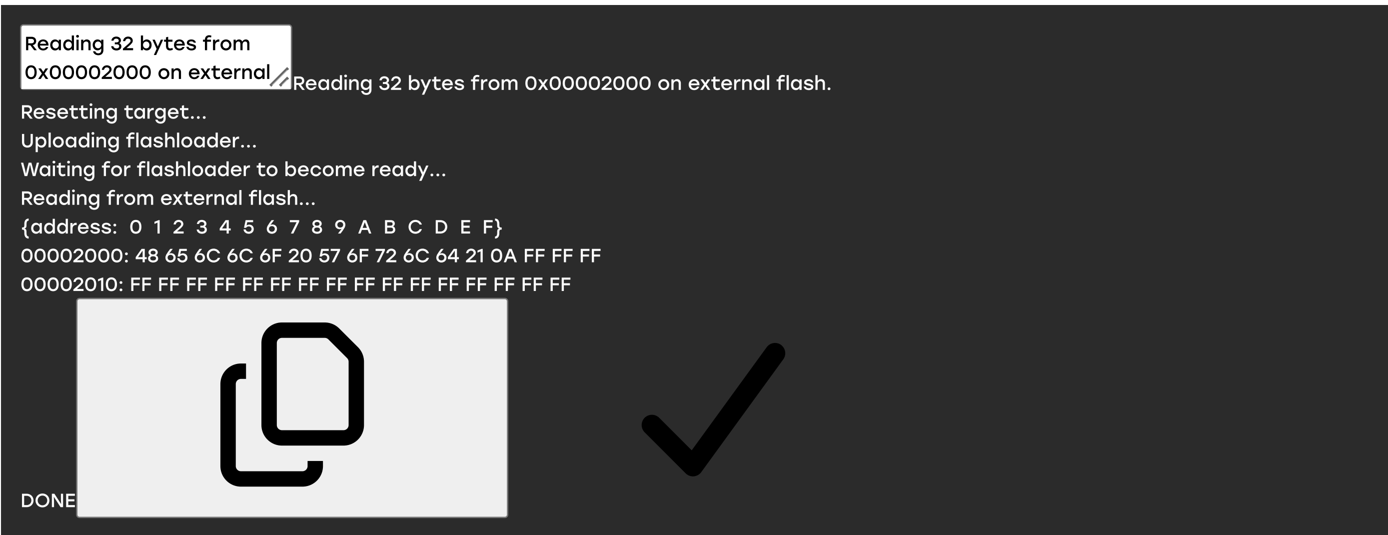
Command Line Input Example



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

Command Line Output Example

Copyright © 2026 Silicon Laboratories. All rights reserved.




## Write External SPI Flash Command

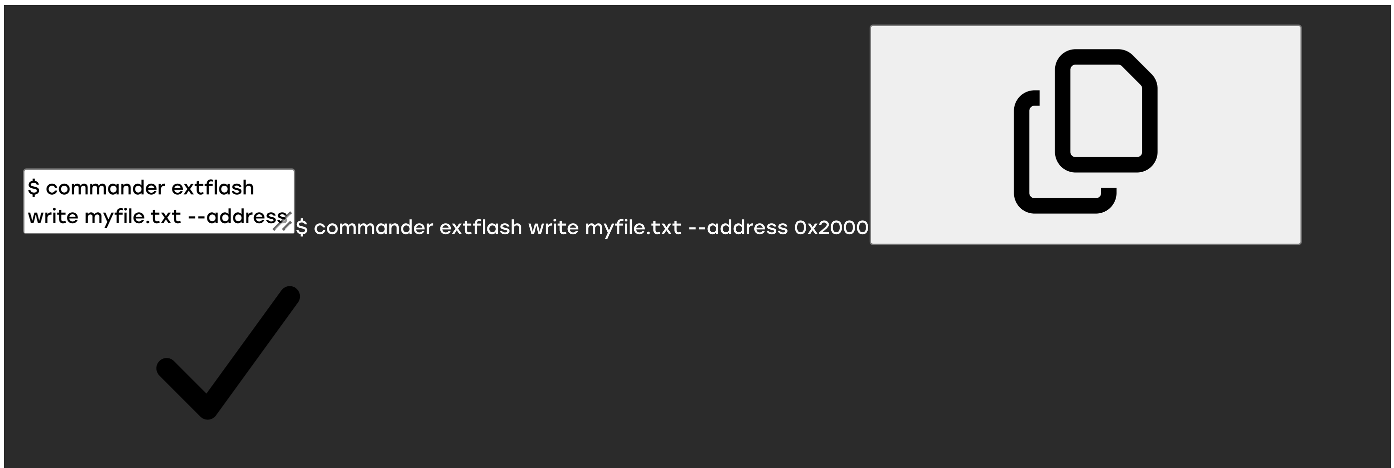
Use this command to write to external flash.

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 the address option is given the value is interpreted as a hexadecimal number. 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.

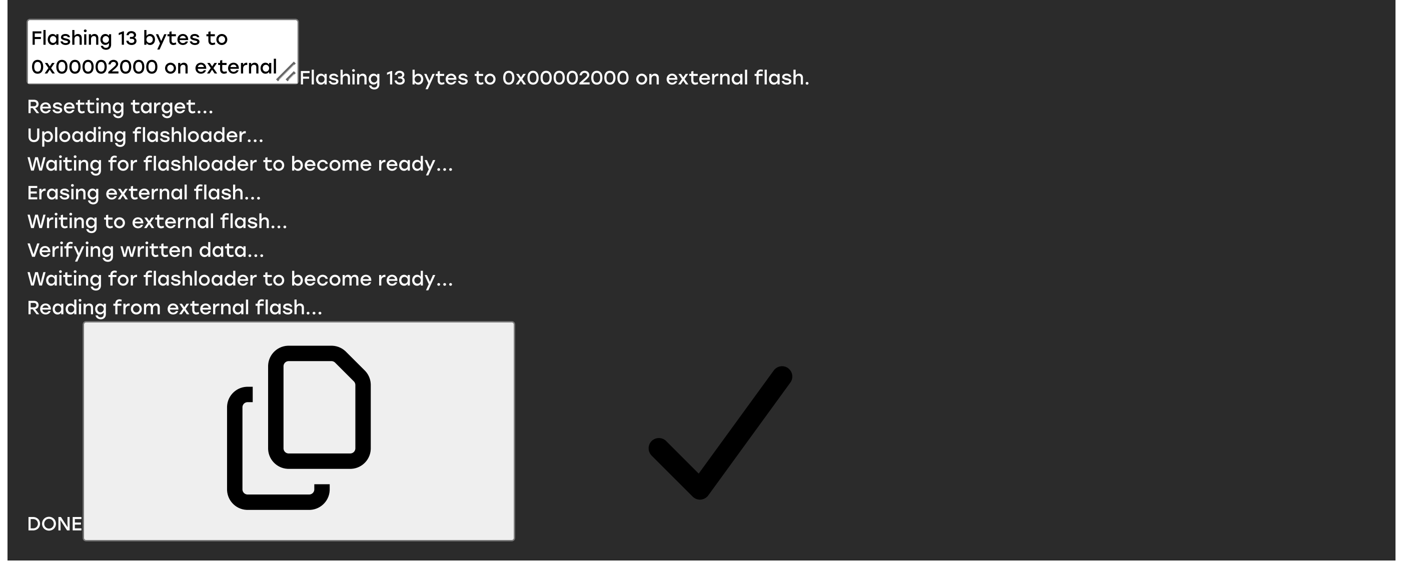
Command Line Syntax

Command Line Input Example




Copyright © 2026 Silicon Laboratories. All rights reserved.

Command Line Output Example



Advanced Energy Monitor Commands

# Advanced Energy Monitor Commands

Simplicity Commander supports reading and logging current measurement data from the Advanced Energy Monitor (AEM) of the adapter.

## Measure Average Current in a Time Window

The `aem measure` command measures the average current in a time window. The `--windowlength` is in milliseconds (ms) and is defined as the duration where current samples will be measured and averaged. The default is 100 ms if no time is given. Ongoing measurements can be terminated by pressing CTRL+C.

Command Line Syntax

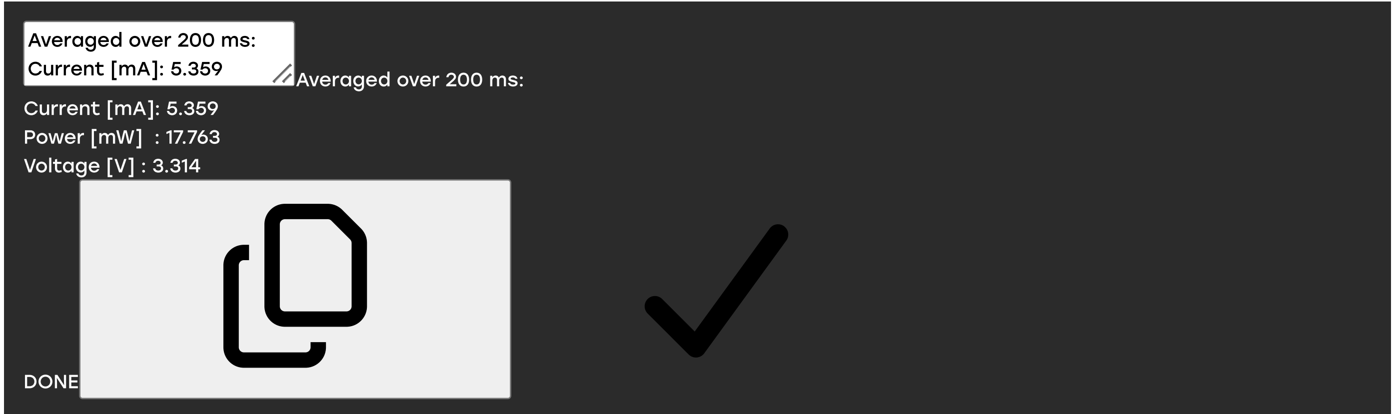
```
$ commander aem measure [--windowlength <time in ms>]
```

Command Line Input Example

```
$ commander aem measure --windowlength 200
```

Command Line Output Example

Copyright © 2026 Silicon Laboratories. All rights reserved.




## Log Current Measurements as Time Series Data

The `aem dump` command continuously measures the current and logs the measurement data (voltage and current). If `--outfile` is provided, the data is logged in the specified output file, otherwise the data is streamed to the terminal window. If `--duration` is provided, the logging will stop after the specified time, otherwise the logging will continue indefinitely. In both cases, the logging may be terminated at any time by pressing CTRL+C. The `--noheader` option can be passed to omit the column header from being included in the output file. The options described here are also applicable for use with the other options related to the aem dump command, but will be omitted in the next sections for the sake of brevity.

If the `--datarate` option is provided, the rate of which the current measurements are logged is set to the specified rate. This will collect samples over a time equal to the reciprocal of the provided data rate, and average these samples before storing the data in the output log (terminal or specified output file). The data rate must be equal to or larger than 1 Hz, and also equal to or less than the AEM sampling rate of the adapter in question. If `--datarate` is not provided, the command will default to the AEM sampling rate of the adapter.

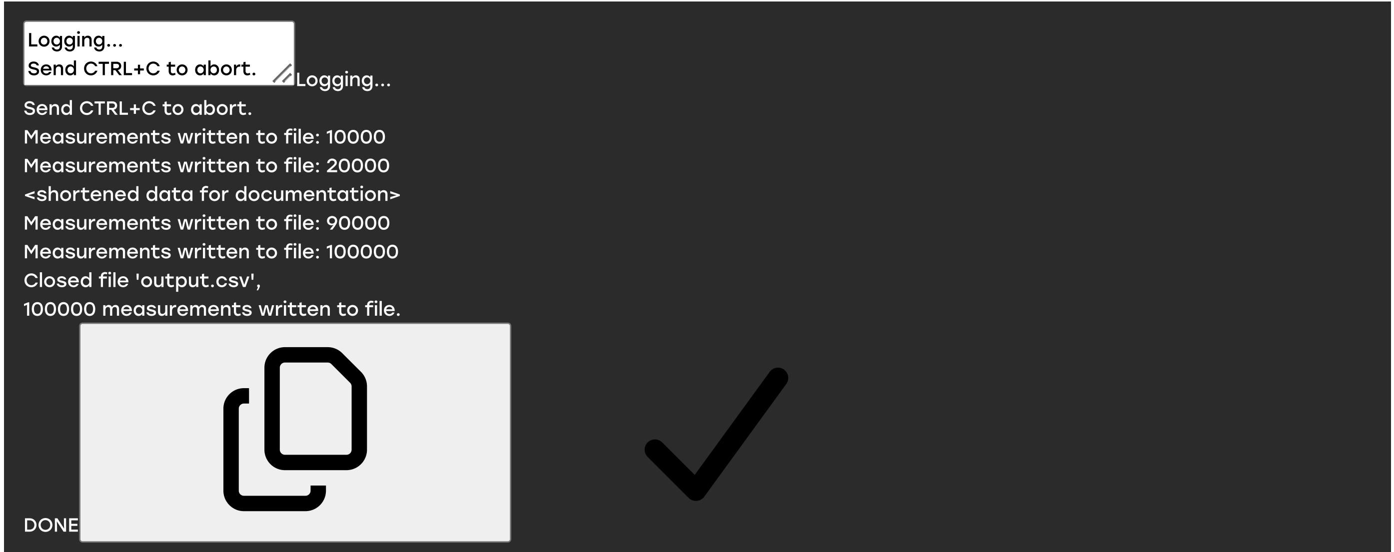
The output file must be of either .txt or .csv format.

Command Line Syntax

Command Line Input Example

This command will log AEM measurements for 10 seconds and store the data in 'output.csv', including a column header.

Command Line Output Example




## Start Logging on Trigger Event

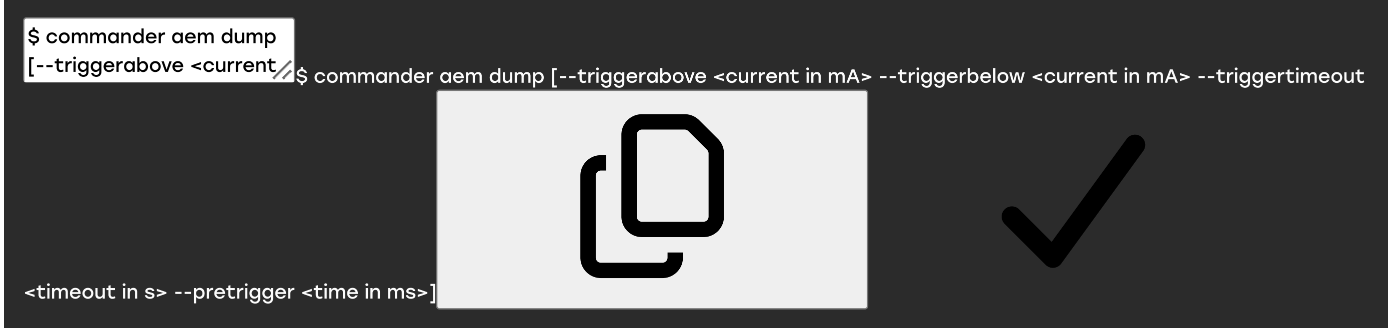
Trigger parameters can be set up to start logging when either the current is above a certain current threshold (in mA) or below a certain threshold. The `--triggerabove` and `--triggerbelow` options can be included to specify the type of triggering event. Only one of these options can be used when issuing this command.

Using the `--triggertimeout` option, you can specify how long the command should wait for the trigger event before timing out. Additionally, the `--pretrigger` option may be included to allow for logging of the measurements in the specified time window leading up to the trigger event. If the actual trigger event occurs before the length of the pre-trigger time has passed (after the command was executed), the actual included pre-trigger data will be the data that was collected from the execution of the command until the trigger event has occurred.

> Note: Pay attention to the units used with the following options.

Command Line Syntax

Copyright © 2026 Silicon Laboratories. All rights reserved.



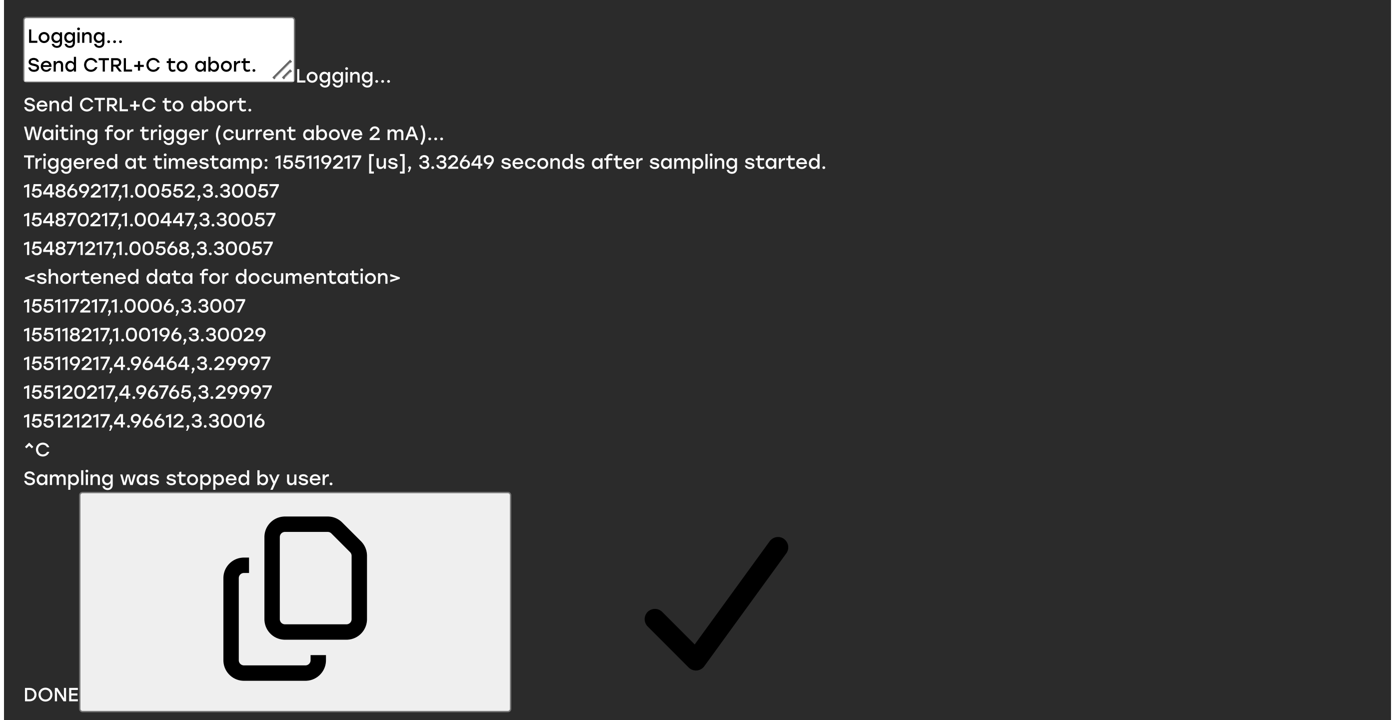
```
$ commander aem dump [--triggerabove <current in mA> --triggerbelow <current in mA> --triggertimeout <timeout in s> --pretrigger <time in ms>]
```

Command Line Input Example

```
$ commander aem dump --triggerabove 2 --triggertimeout 10 --pretrigger 250 --datarate 1000
```

This command logs AEM measurements at a rate of 1000 Hz to the console output (indefinitely), starting from when the measured current is greater than 2 mA (milliampere). Data recorded up to 250 ms (milliseconds) before the actual trigger event will also be logged. If no trigger condition is met, the command will time out after 10 seconds.

Command Line Output Example



```
Logging...
Send CTRL+C to abort.
Waiting for trigger (current above 2 mA)...
Triggered at timestamp: 155119217 [us], 3.32649 seconds after sampling started.
154869217,1.00552,3.30057
154870217,1.00447,3.30057
154871217,1.00568,3.30057
<shortened data for documentation>
155117217,1.0006,3.3007
155118217,1.00196,3.30029
155119217,4.96464,3.29997
155120217,4.96765,3.29997
155121217,4.96612,3.30016
^C
Sampling was stopped by user.
DONE
```

## Calibrate the Advanced Energy Monitor

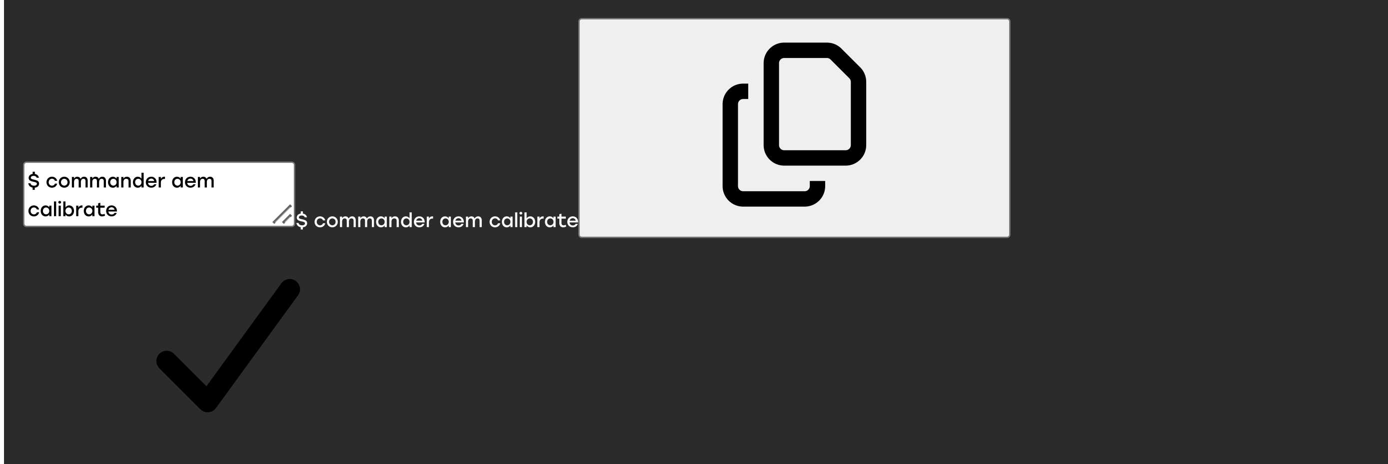
The Advanced Energy Monitor (AEM) can be calibrated using the `aem calibrate` command.

Copyright © 2026 Silicon Laboratories. All rights reserved.

In order to retain the AEM accuracy, calibration should be performed after changes in the target voltage are applied, or if the temperature of the adapter boards changes.

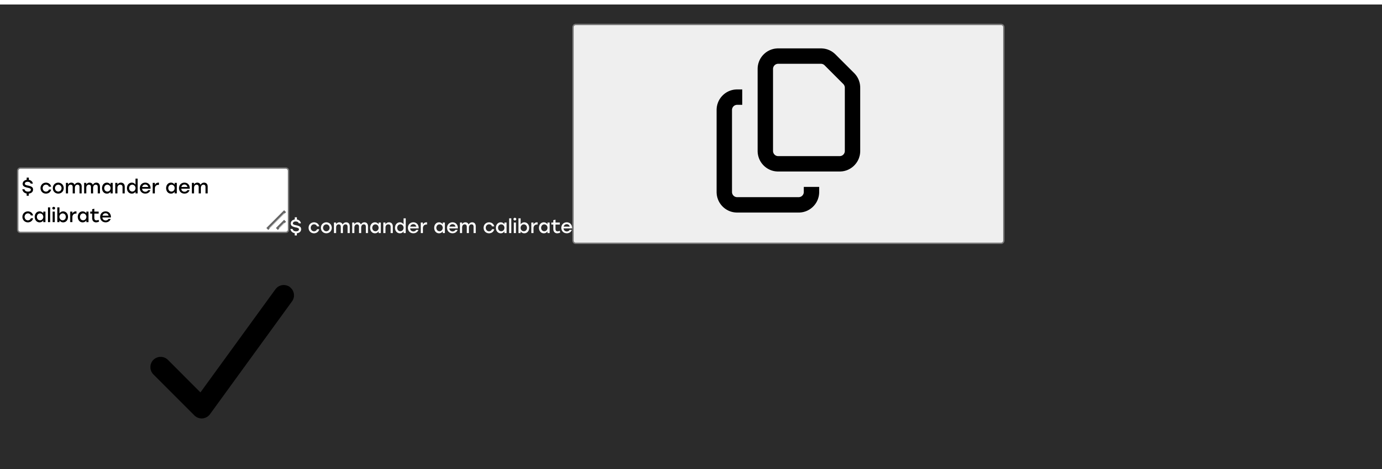
> Note: Target power will be turned off during AEM calibration.

Command Line Syntax



```
$ commander aem calibrate
```

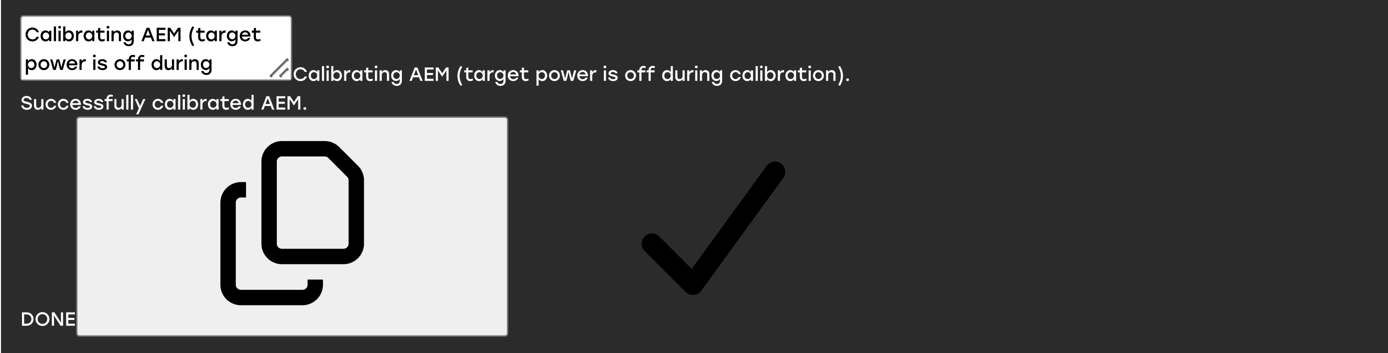
Command Line Input Example



```
$ commander aem calibrate
```

This command line initiates calibration of the AEM.

Command Line Output Example



```
Calibrating AEM (target power is off during calibration).
Successfully calibrated AEM.
DONE
```

Serial Wire Output Read Commands

# Serial Wire Output Read Commands

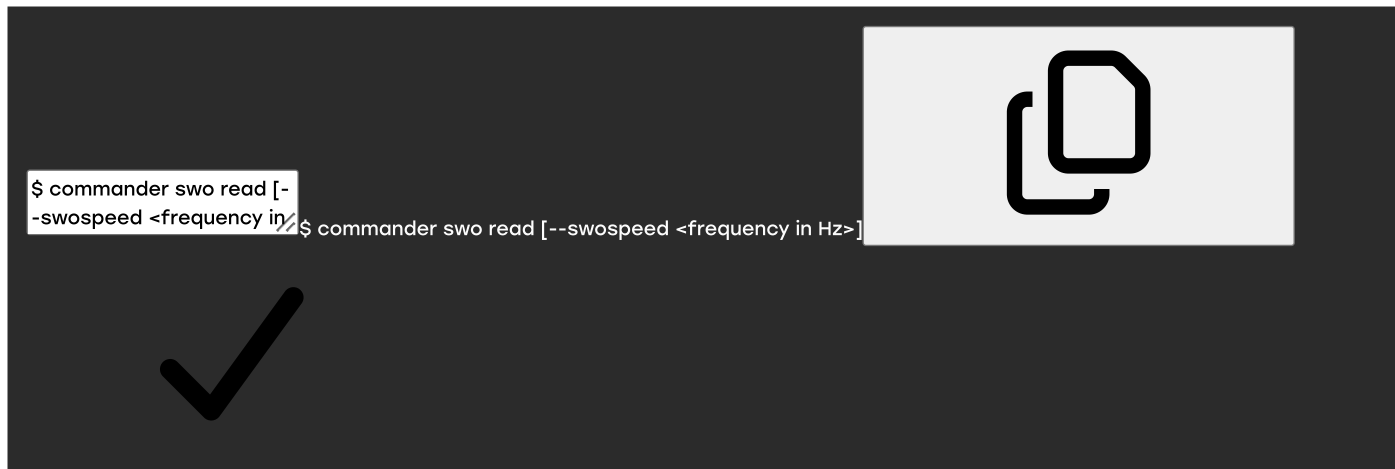
Simplicity Commander supports reading and dumping data received over Serial Wire Output (SWO) using the `swo read` command. When the command is executed, the target device is reset. The command will then read and dump SWO data until the application is terminated by pressing Ctrl+C, or one of the conditions described below is met.

By default, the target will be reset during initialization of the SWO connection. Providing the `--noreset` option will prevent this.

## Configure SWO Speed

This command sets the SWO speed frequency in Hz. The default SWO speed is 875000 Hz. The SWO speed must match the frequency used by the target application.

Command Line Syntax



```
$ commander swo read [--swospeed <frequency in Hz>]
```

Command Line Input Example

```
$ commander swo read --swospeed 1000000
```

Command Line Output Example


Copyright © 2026 Silicon Laboratories. All rights reserved.

```
<data written by the target application at 1 MHz>
CTRL+C entered, terminating SWO connection...
DONE
```

## Read SWO Until Timeout

This command sets the number of seconds for the adapter to wait without receiving data before it times out. The default is to never time out.

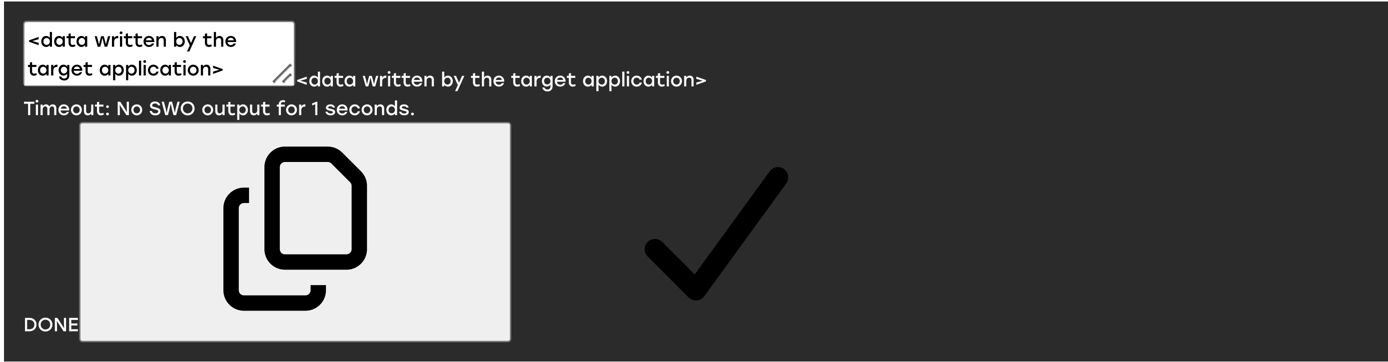
Command Line Syntax

```
$ commander swo read [--timeout <timeout in s>]
```

Command Line Input Example

```
$ commander swo read --timeout 1
```

Command Line Output Example

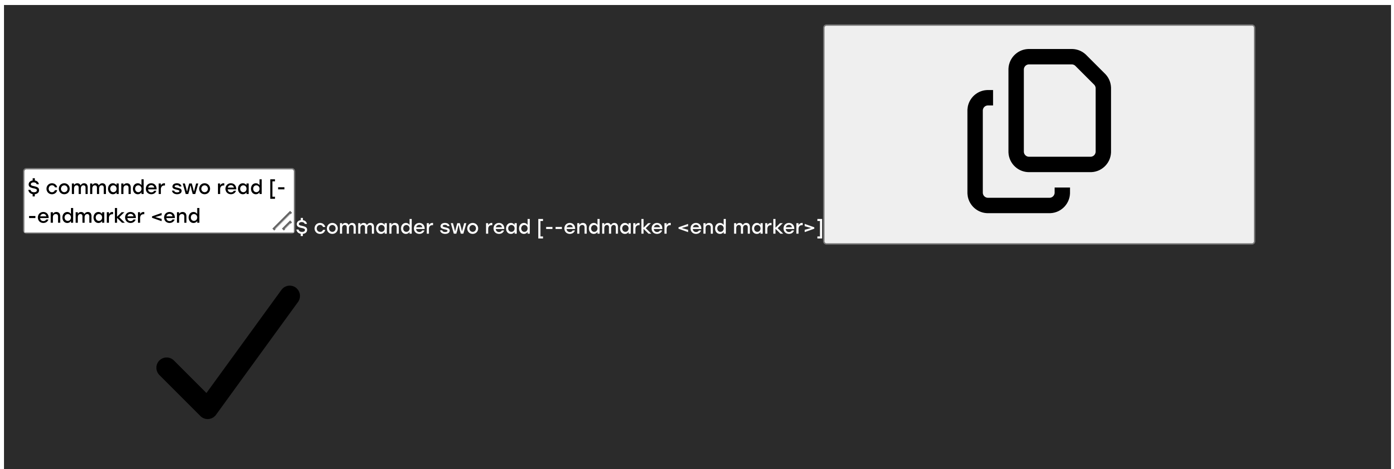


```
<data written by the target application>
Timeout: No SWO output for 1 seconds.
DONE
```

## Read SWO Until a Marker Is Found

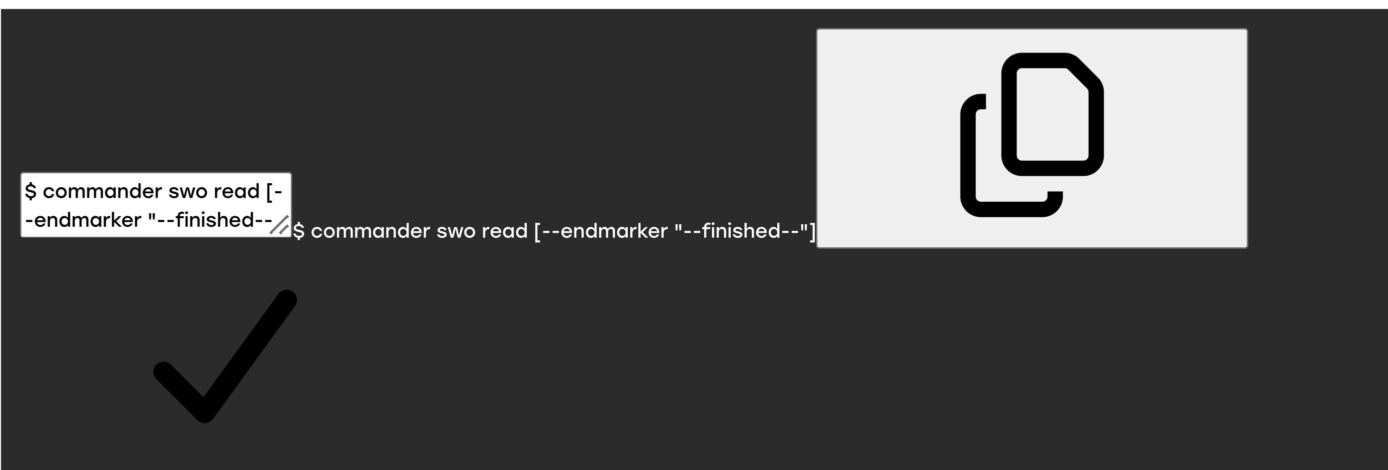
If the `--endmarker` option is used, the command will terminate after finding the specified string in the SWO stream.

Command Line Syntax



```
$ commander swo read [--endmarker <end marker>]
```

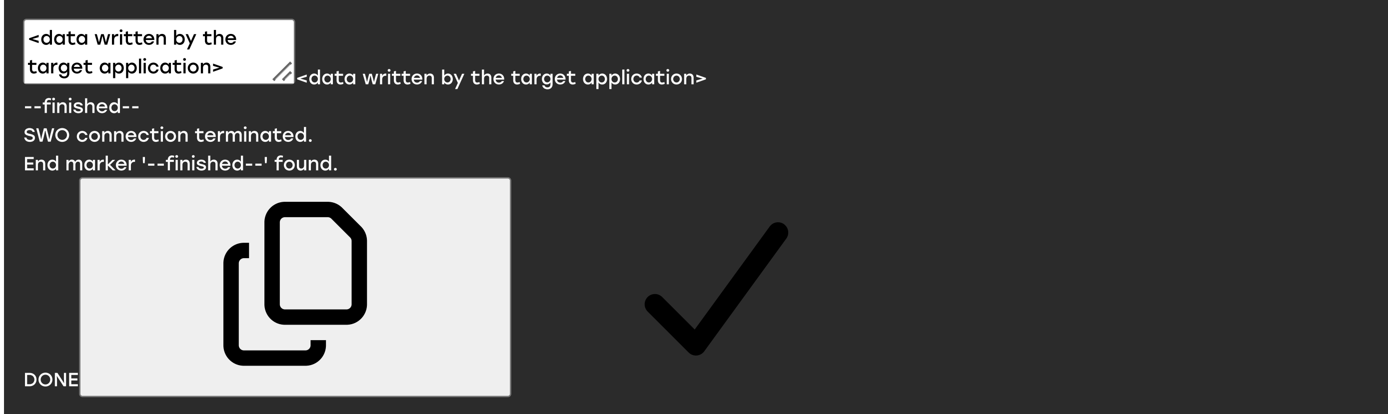
Command Line Input Example



```
$ commander swo read [--endmarker "--finished--"]
```

Command Line Output Example

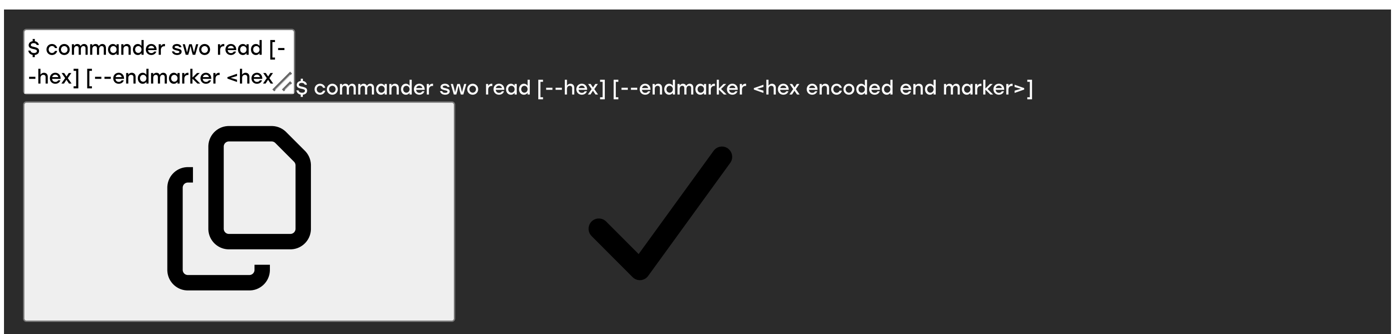

Copyright © 2026 Silicon Laboratories. All rights reserved.




## Dump Hex Encoded SWO Output

If the `--hex` option is used, all input and output is converted to a hexadecimal string. This is useful if the target dumps binary data. If the `--hex` option is used, `--endmarker` must also be hex-encoded.

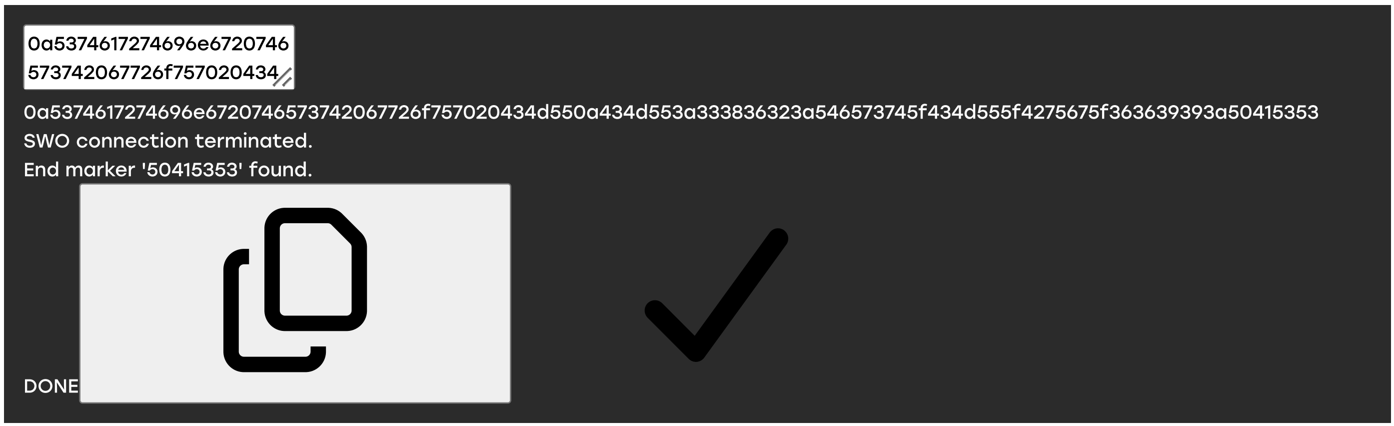
Command Line Syntax




Command Line Input Example

Command Line Output Example

Copyright © 2026 Silicon Laboratories. All rights reserved.


0a5374617274696e6720746
573742067726f757020434
0a5374617274696e672074657374206772f757020434d550a434d553a333836323a546573745f434d555f4275675f363639393a50415353
SWO connection terminated.
End marker '50415353' found.
DONE

NVM3 Commands

# NVM3 Commands

The Third Generation Non-Volatile Memory (NVM3) module in the Gecko SDK provides a way to store data in non-volatile memory (flash) on EFM32, EFR32, and SiWx91x devices. Refer to *UG103.7: Non-Volatile Memory Fundamentals* or *AN1135: Using Third Generation Non-Volatile Memory (NVM3) Data Storage in Dynamic Multiprotocol Applications* for more details on NVM3.

Simplicity Commander supports initializing, modifying, and reading NVM3 from a device.

Initializing and modifying NVM3 data can be applicable on the production line to set an initial state.

Reading out the NVM3 data area from a device and parsing the NVM3 data to extract stored values can be useful in a debugging scenario where you may need to find out the stored state of an application that has been running for some time.

## On-Device and Offline NVM3 Processing

NVM3 is implemented differently across device families, which leads to different user interfaces in Simplicity Commander.

### Series 1 and Series 2 (EFR32/EFM32), and SiWx91x Devices

On these devices, NVM3 is stored unencrypted in flash. Commander can read and write the NVM3 data directly from flash, and all NVM3 processing can be done "offline" on the host computer. The offline commands `initfile`, `dump`, `parse`, `set`, and `delete` can be used for offline NVM3 manipulation for these devices, while the on-device commands `readdevice`, `writedevice`, and `deletedevice` can be used to manipulate NVM3 data directly on the device.

### Series 3 (SixG3xx) Devices

On these devices, NVM3 is stored encrypted in flash. The key used for encryption is accessible only by the SE. Therefore, all NVM3 parsing and manipulation must happen on-chip. Simplicity Commander handles this by loading RAM code that performs NVM3 parsing and manipulation. The commands `readdevice`, `writedevice`, and `deletedevice` are suitable for NVM3 manipulation on these devices.

## Offline NVM3 Processing

#### Workflow to Initialize NVM3 on a Blank Device

1. Use `commander nvm3 initfile` to initialize an empty NVM3 area.
2. Use `commander nvm3 set` to set NVM3 objects in the file.
3. Use `commander flash` to write the NVM3 area with data to the device.

#### Workflow to Read and Modify NVM3 data on a Device

1. Use `commander nvm3 dump` to find and read NVM3 data from the device.
2. Use `commander nvm3 parse` to check the state of NVM3 objects in the file.
3. Use `commander nvm3 set` to set or modify NVM3 objects in the file.
4. Use `commander nvm3 delete` to delete NVM3 objects in the file.
5. Use `commander flash` to write the modified NVM3 area with data to the device.

Copyright © 2026 Silicon Laboratories. All rights reserved.

### On-Device NVM3 Processing

#### Workflow to Initialize NVM3 on a Blank Device

1. Use `commander nvm3 writedevice` to initialize an NVM3 area with data as given by command line options or a text file.

#### Workflow to Read and Modify NVM3 data on a Device

1. Use `commander nvm3 readdevice` to find and parse NVM3 data on the device.
2. Use `commander nvm3 writedevice` to set or modify NVM3 objects on the device.
3. Use `commander nvm3 deletedevice` to delete NVM3 objects on the device.

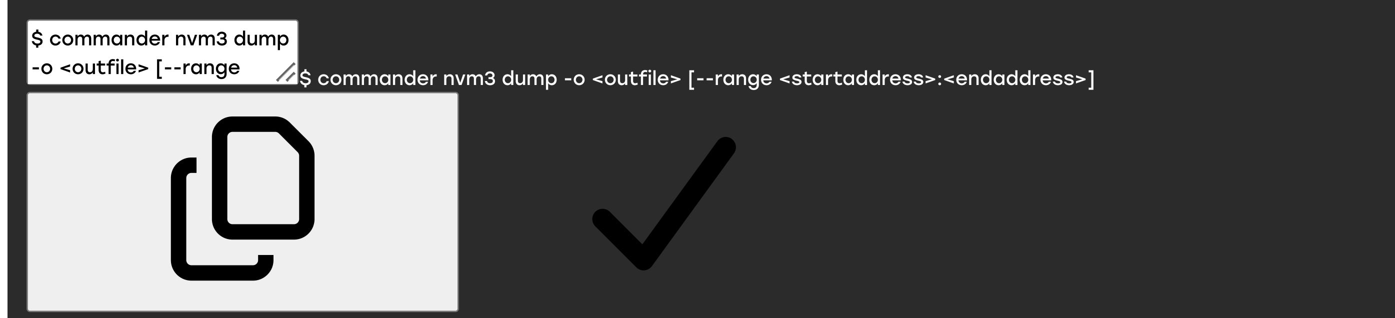
## Dump NVM3 Data From a Device

This command searches for an NVM3 area in the device's flash and dumps the content to a file in .bin, .s37 or .hex format.

The optional `--range` parameter can be used to specify the memory range where Simplicity Commander should search for NVM3 data. If no range is given, the entire flash is searched.

This command was named `nvm3 read` prior to version 1.17.1 of Simplicity Commander. The old command name remains supported as an alias.

Command Line Syntax



```
$ commander nvm3 dump -o <outfile> [--range <startaddress>:<endaddress>]
```

Command Line Input Example

```
$ commander nvm3 dump -o my_nvm3_data.s37
```

This example scans device flash and searches for a valid NVM3 area. When it is found, the NVM3 area is written to the file named `my_nvm3_data.s37`.

Command Line Output Example

Copyright © 2026 Silicon Laboratories. All rights reserved.

## Parse NVM3 Data from a File

This command takes an image file containing NVM3 data and parses the contents. The parsed NVM3 objects are printed to standard out.

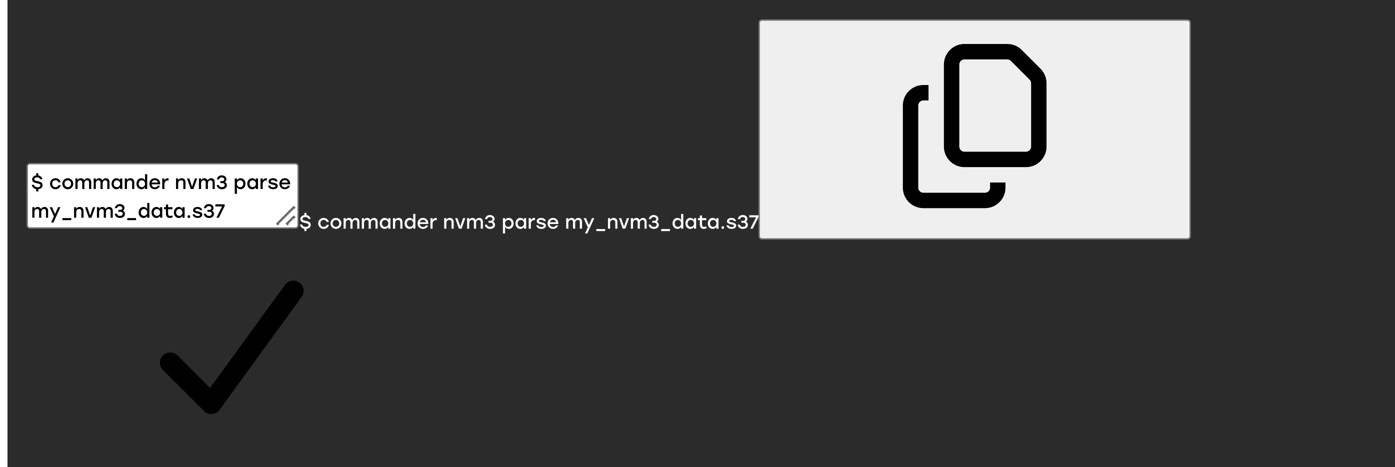
The optional `--range` parameter can be used to specify the memory range where Simplicity Commander should search for NVM3 data. If no range is given, the entire file is searched.

The optional `--key` parameter can be used to specify specific NVM3 keys to look up. It can be used multiple times to look up more than one key at a time. Objects with more than eight bytes of data will be truncated when listing all objects. Use the `--key` parameter to select objects whose data should be displayed.

The optional `--nvm3file` parameter can be used to save the NVM3 objects to a text file in the same format as described below for the `set` command. If the `--key` parameter is used, only the specified objects are saved to the file.

Command Line Syntax

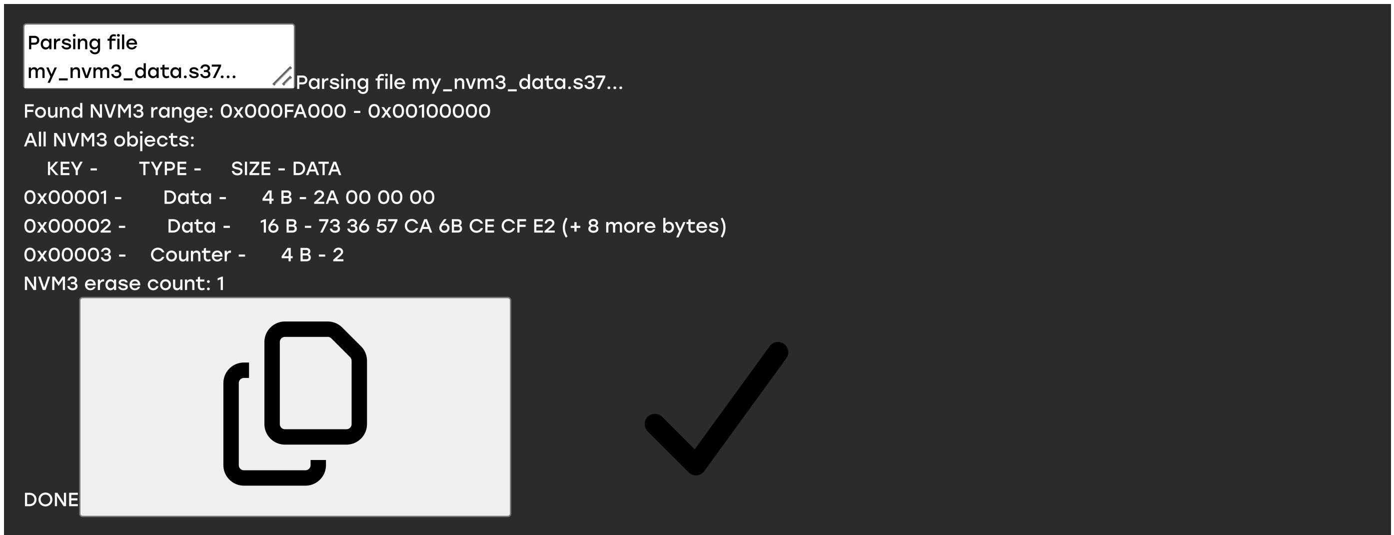
Command Line Input Example




Scans through the given file and searches for valid NVM3 data. When it is found, the data is parsed and printed to standard out.

Copyright © 2026 Silicon Laboratories. All rights reserved.

Command Line Output Example




## Initialize NVM3 Area in a File

The `nvm3 initfile` command creates a blank NVM3 area in an image file. For example, this feature is useful to create a file that the `nvm3 set` command can work on to create a default set of NVM3 data that can be written during production.

The size and location of the NVM3 area must be given and must match the size and location used in the embedded application using the NVM3 area.

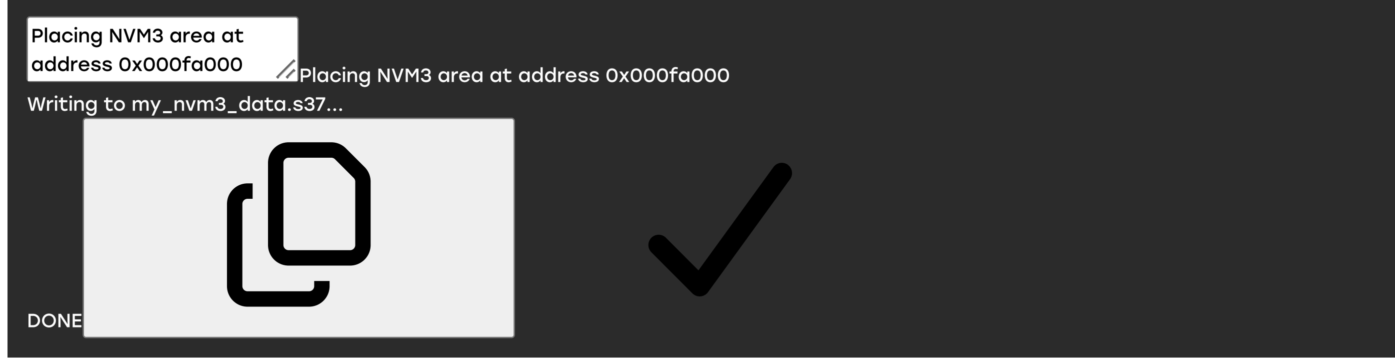
Command Line Syntax

Command Line Input Example

This creates a 24 kB NVM3 area spanning the flash address range 0xfa000 - 0x100000.

Command Line Output Example

Copyright © 2026 Silicon Laboratories. All rights reserved.




## Write NVM3 Data to a File Using a Text File

The nvm3 set command takes an image file containing an NVM3 data region and sets the value of one or more NVM3 objects. The objects may already exist, in which case the value is updated. If the object does not already exist, it is created. The definition of the data to write can be passed either as a text file ( --nvm3file ) or as command line parameters ( --object and --counter ).

The text file passed by the --nvm3file option must have the following format:

- Each line defines a single object or counter.
- Empty lines are ignored.
- Lines starting with # are ignored.

Each line in the file must have the following syntax:

<key>:<type>:<data>

<key> is the NVM3 object key which is the unique identifer used by the embedded application. It has a maximum size of 20 bits (maximum value 0xFFFFF).

<type> is the NVM3 object type. It can be one of two values: OBJ or CNT . OBJ indicates a plain byte array. CNT indicates an NVM3 counter type (32-bit unsigned integer).

<data> is the value the object should be set to. For counter types, the value is interpreted as an unsigned integer which can be prefixed with 0x to indicate a hexadecimal value. Byte arrays are always parsed as hexadecimal and should not be prefixed with 0x.

Example File

This file sets the object with ID 0x1 to be a byte array of eight bytes in length with the contents above.

The object with ID 0x1000 is a counter with value 0x80 (128). The object with ID 0x1001 is a counter with value 42.

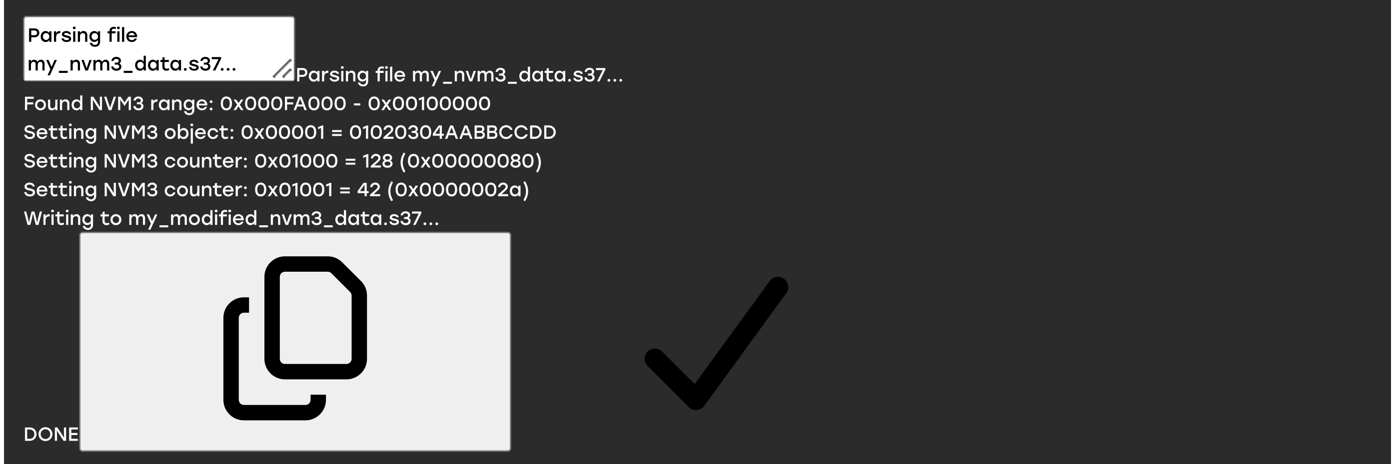
Command Line Syntax

Copyright © 2026 Silicon Laboratories. All rights reserved.

Command Line Input Example

`nvm3_objects.txt` is parsed for NVM3 objects following the format described above. The given input image file is scanned for a valid NVM3 region. The objects defined in the text file are written into the NVM3 region and the modified output is written to the output image file.

Command Line Output Example




## Write NVM3 Data to a File Using CLI Options

In some cases, it may be more convenient to set the NVM3 object data directly from the command line without using a text file. In this instance, use the command line options `--object` and `--counter`.

The two options both use the same syntax: `<key>:<data>`. The definitions of `<key>` and `<data>` are the same as in the previous section. The only difference between the two formats is that the `<type>` field has been removed because it is given by the command line option name instead.

Simplicity Commander automatically finds the correct NVM3_MAX_OBJECT_SIZE based on the given size of NVM3 area.

Command Line Syntax

Copyright © 2026 Silicon Laboratories. All rights reserved.

```
$ commander nvm3 set <input image file> --object <key>:<data> --counter <key>:<data> --outfile <image file>
```

Command Line Input Example

```
$ commander nvm3 set my_nvm3_data.s37 --object 0x1:01020304AABBCCDD --counter 0x1000:0x80 --counter 0x01001:42 --outfile my_modified_nvm3_data.s37
```

All `--object` and `--counter` parameters are parsed according to the format above. The given input image file is scanned for a valid NVM3 region. The objects defined in the text file are written into the NVM3 region and the modified output is written to the output image file.

Command Line Output Example

```
Parsing file my_nvm3_data.s37...
Setting NVM3 object: 0x00001 = 01020304AABBCCDD
Setting NVM3 counter: 0x01000 = 128 (0x00000080)
Setting NVM3 counter: 0x01001 = 42 (0x0000002a)
Writing to my_modified_nvm3_data.s37...
DONE
```

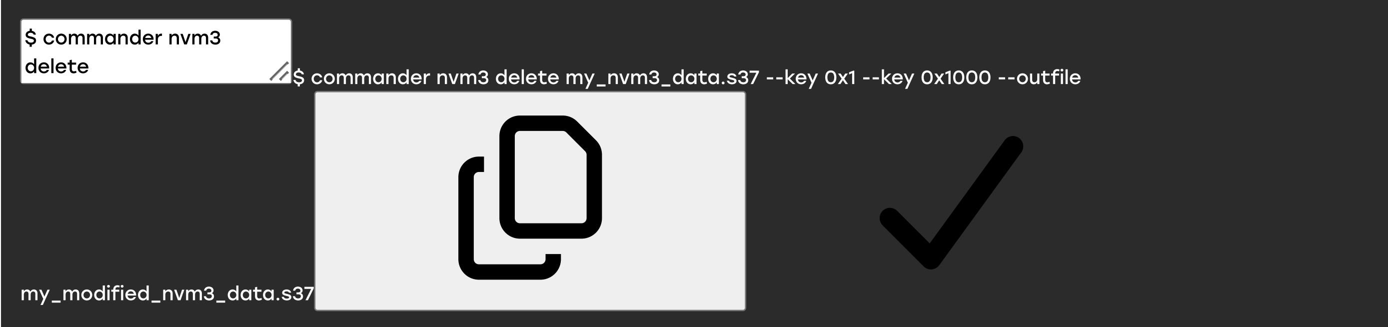
## Delete NVM3 Data From a File

You can delete NVM3 objects and counters from an NVM3 image file using the `nvm3 delete` command. The objects to delete are specified using the `--key` parameter. You can use it multiple times to delete more than one object at a time. If a key specified for deletion does not exist, it is simply ignored. You can also provide `--all` to delete all objects in the NVM3 area.

Command Line Syntax

Copyright © 2026 Silicon Laboratories. All rights reserved.

Command Line Input Example




This example deletes the objects with keys 0x1 and 0x1000 from the NVM3 area in the input image file. The modified NVM3 area is written to the output image file.

Command Line Output Example

## Read NVM3 Data From a Device

This command searches for an NVM3 area in the device's flash and parses the contents. The parsed NVM3 objects are printed to standard out.

The options for `readdevice` are largely the same as for `parse` .

The optional `--range` parameter specifies the memory range where Simplicity Commander should search for NVM3 data. If no range is given, the entire flash is searched.

The optional `--key` parameter specifies NVM3 keys to look up. You can use it multiple times to look up more than one key at a time. Objects with more than eight bytes of data are truncated when listing all objects. Use the `--key` parameter to select objects whose data should be displayed.

The optional `--nvm3file` parameter can be used to save the NVM3 objects to a text file in the same format as described above for the `set` command. If the `--key` parameter is used, only the specified objects are saved to the file.

Copyright © 2026 Silicon Laboratories. All rights reserved.

Command Line Syntax

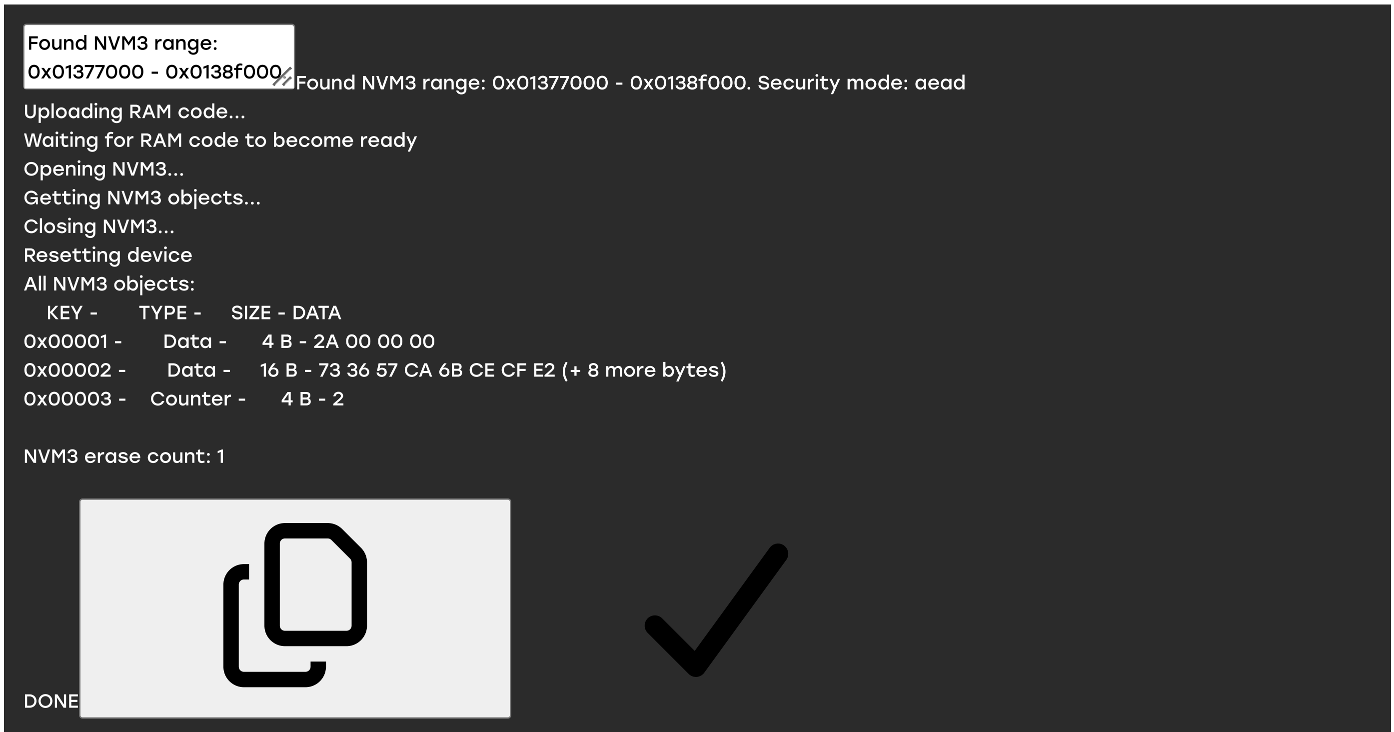
```
$ commander nvm3 readdevice [--range <startaddress>:<endaddress>] [--key <object key>] [--nvm3file <filename>]
```

Command Line Input Example

```
$ commander nvm3 readdevice
```

This example scans through device flash and searches for a valid NVM3 area. When it is found, RAM code is uploaded to the device, and the data is parsed and printed to standard out.

Command Line Output Example



```
Found NVM3 range: 0x01377000 - 0x0138f000. Security mode: aead
Uploading RAM code...
Waiting for RAM code to become ready
Opening NVM3...
Getting NVM3 objects...
Closing NVM3...
Resetting device
All NVM3 objects:
   KEY -      TYPE -     SIZE - DATA
0x00001 -      Data -      4 B - 2A 00 00 00
0x00002 -      Data -     16 B - 73 36 57 CA 6B CE CF E2 (+ 8 more bytes)
0x00003 -   Counter -      4 B - 2

NVM3 erase count: 1
DONE
```

## Write NVM3 Data to a Device

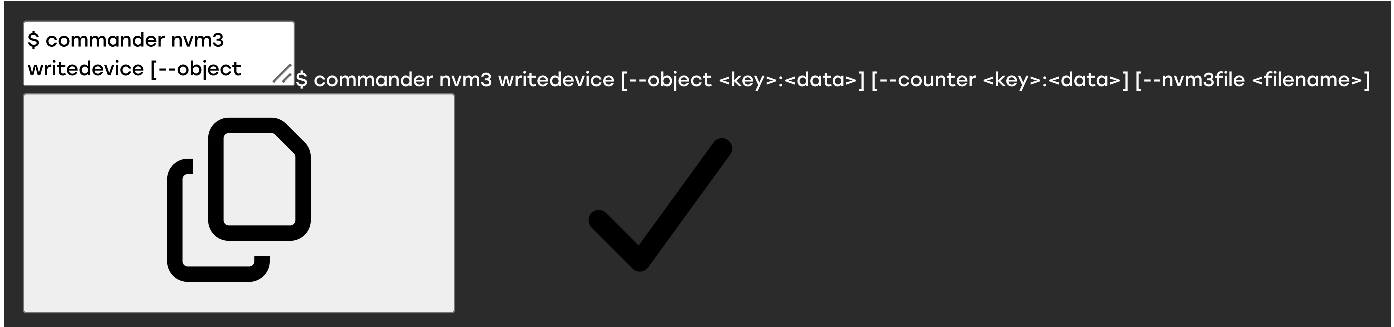
Copyright © 2026 Silicon Laboratories. All rights reserved.

The options for `writedevice` are largely the same as for `set`. You can use either `--nvm3file` and/or `--object` and `--counter` to specify data to be written. The syntax and details for these options are the same as for the `set` command as described above.

`nvm3 writedevice` will search the device for an existing NVM3 area and use that. If no NVM3 area is found, and `--range` is given to provide a start and end address for NVM3, the NVM3 area is initialized in the given range.

> Note: `nvm3 writedevice` interacts with the NVM3 instance in the device's flash. Take care to *either* let the app run for long enough to fully initialize its NVM3 area, *or* let Simplicity Commander initialize the NVM3 area from scratch (i.e., run `nvm3 writedevice` before flashing the app).

Command Line Syntax



```
$ commander nvm3 writedevice [--object <key>:<data>] [--counter <key>:<data>] [--nvm3file <filename>]
```

Command Line Input Example

```
$ commander nvm3 writedevice --object 0x1:01020304AABBCCDD --counter 0x1000:0x80 --counter 0x01001:42
```

All `--object` and `--counter` parameters are parsed. The device flash is scanned for a valid NVM3 region. RAM code is uploaded to the device. The objects and counters are written into the NVM3 region on the device. Finally, the device is reset.

Command Line Output Example

Copyright © 2026 Silicon Laboratories. All rights reserved.

## Delete NVM3 Data From a Device

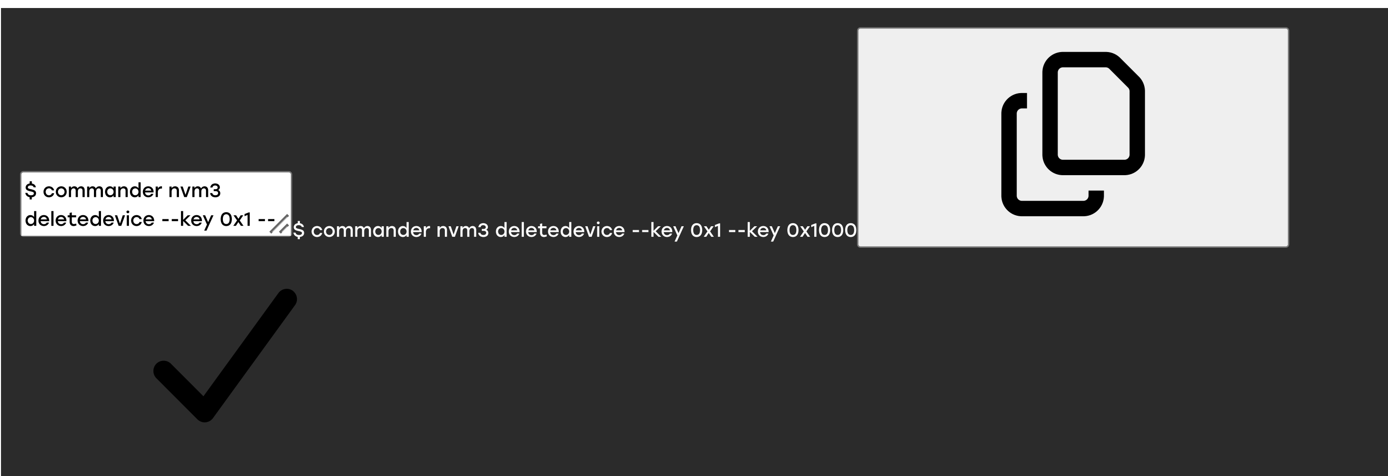
You can delete NVM3 objects and counters from an NVM3 area on a device using the `nvm3 deletedevice` command. The objects to delete are specified using the `--key` parameter. You can use it multiple times to delete more than one object at a time. If a key specified for deletion does not exist, it is simply ignored. You can also provide `--all` to delete all objects in the NVM3 area.

> Note: `nvm3 deletedevice` interacts with the NVM3 instance in the device's flash. If there is no NVM3 area found on the device, the command will fail.

Command Line Syntax

Command Line Input Example

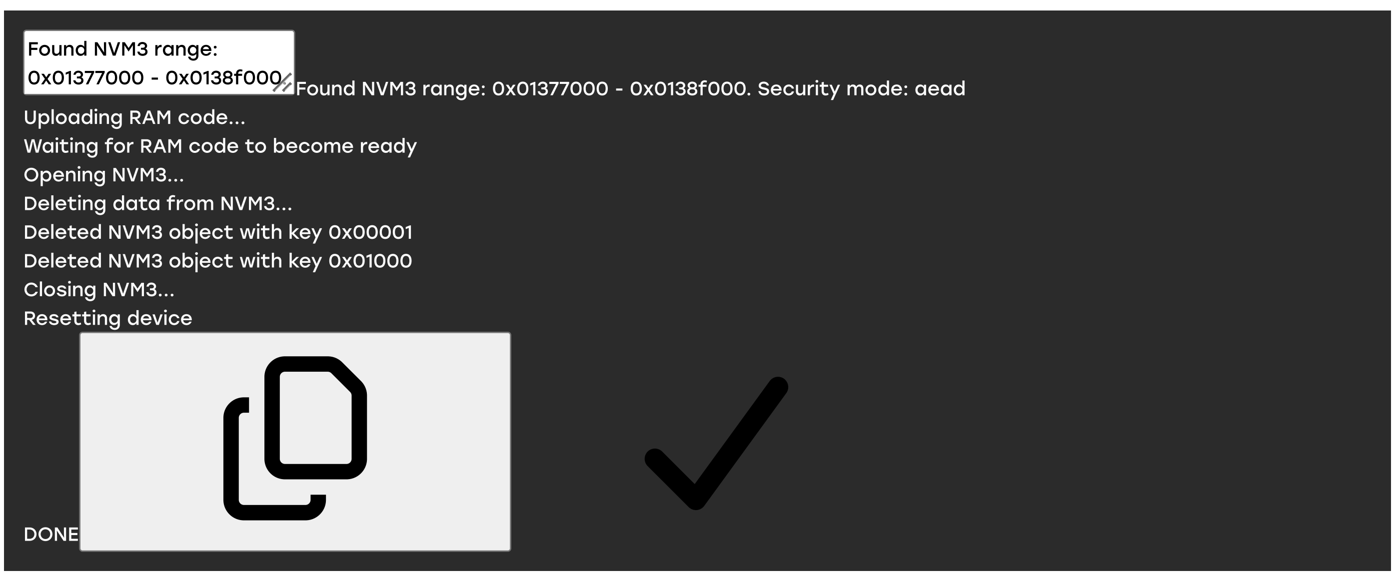
Copyright © 2026 Silicon Laboratories. All rights reserved.



```
$ commander nvm3 deletedevice --key 0x1 --key 0x1000
```

This example deletes the objects with keys 0x1 and 0x1000 from the NVM3 area on the device.

Command Line Output Example



```
Found NVM3 range: 0x01377000 - 0x0138f000. Security mode: aead
Uploading RAM code...
Waiting for RAM code to become ready
Opening NVM3...
Deleting data from NVM3...
Deleted NVM3 object with key 0x00001
Deleted NVM3 object with key 0x01000
Closing NVM3...
Resetting device
DONE
```

Copyright © 2026 Silicon Laboratories. All rights reserved.

CTUNE Commands

# CTUNE Commands

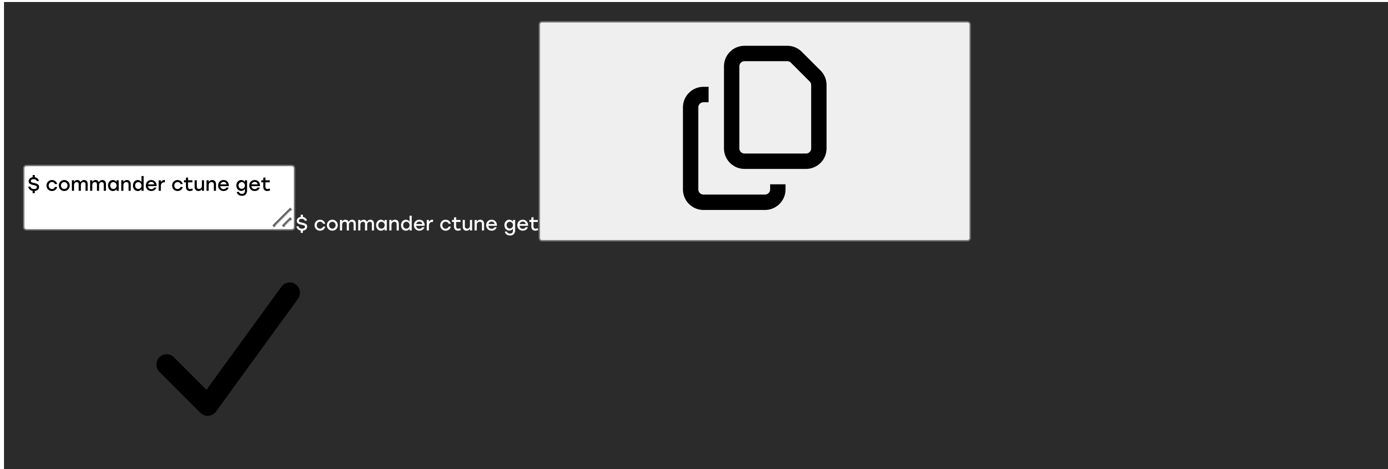
Wireless Gecko (EFR32™) portfolio devices support configuring the crystal oscillator load capacitance in software. The crystal oscillator load capacitor tuning (CTUNE) values are tuned during the production test of both Wireless Gecko-based modules and Silicon Labs Wireless Starter Kit (WSTK) radio boards. For modules, the optimal value for each device is written to the Device Information (DI) page in flash. For radio boards, the optimal value for each board is written to an EEPROM that is inaccessible to the software running on the target device, but readable by Simplicity Commander. The `ctune` commands support reading out the stored CTUNE values from these locations, and writing and reading the CTUNE manufacturing token.

## CTUNE Get Command

This command retrieves the CTUNE value stored in the Device Info page, the value stored in EEPROM on the board, and the value written to the CTUNE manufacturing token. The values are displayed.

Command Line Syntax

Command Line Input Example




Command Line Output Example

Copyright © 2026 Silicon Laboratories. All rights reserved.

> Note: Not all devices have the CTUNE value stored in both the Device Info page and in EEPROM on the board. If this is the case, the value is displayed as "Not set".

## CTUNE Set Command

This command sets the CTUNE manufacturing token to the value specified by the value option.

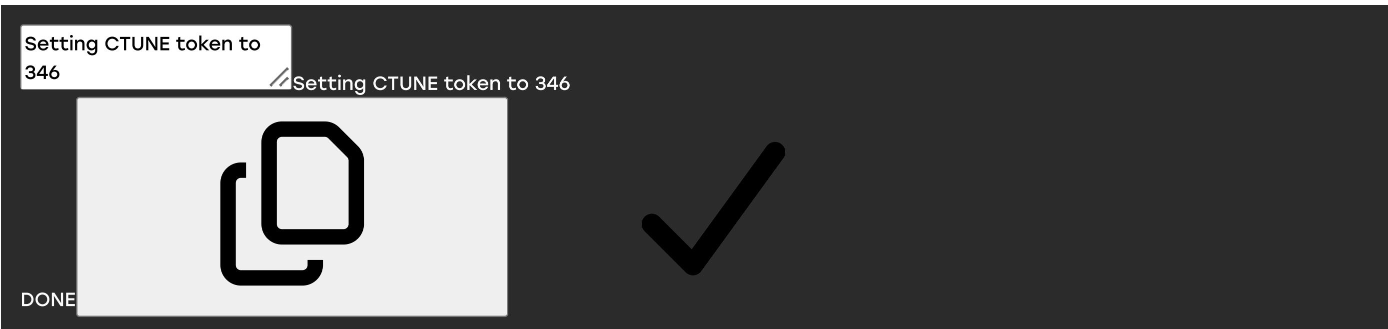
Command Line Syntax

```
$ commander ctune set <value>
```

Command Line Input Example

```
$ commander ctune set --value 346
```

Command Line Output Example



```
Setting CTUNE token to 346
DONE
```

## CTUNE Autoset Command

Copyright © 2026 Silicon Laboratories. All rights reserved.

This command retrieves the CTUNE value from EEPROM on the board and sets the CTUNE manufacturing token to this value.

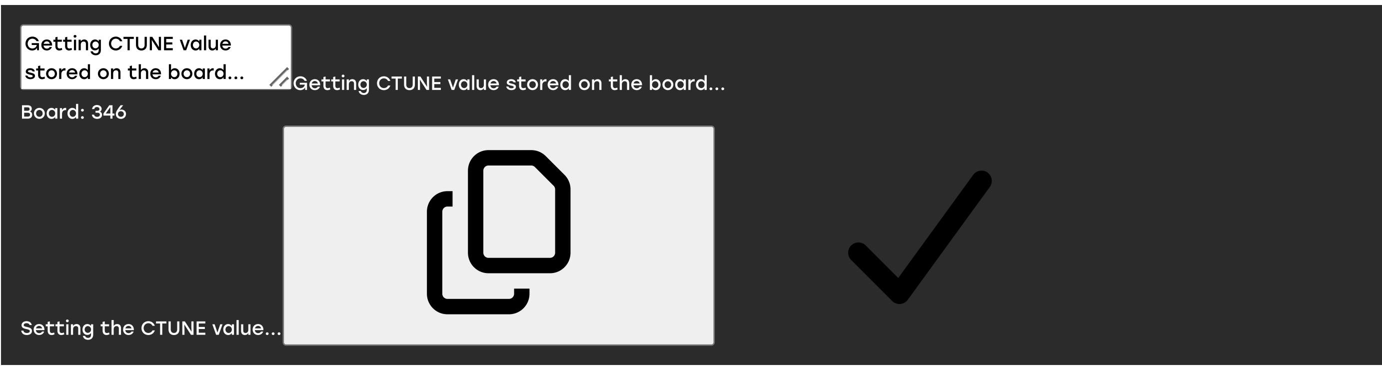
Command Line Syntax

```
$ commander ctune autoset
```

Command Line Input Example

```
$ commander ctune autoset
```

Command Line Output Example



```
Getting CTUNE value stored on the board...
Board: 346
Setting the CTUNE value...
```

Security Commands

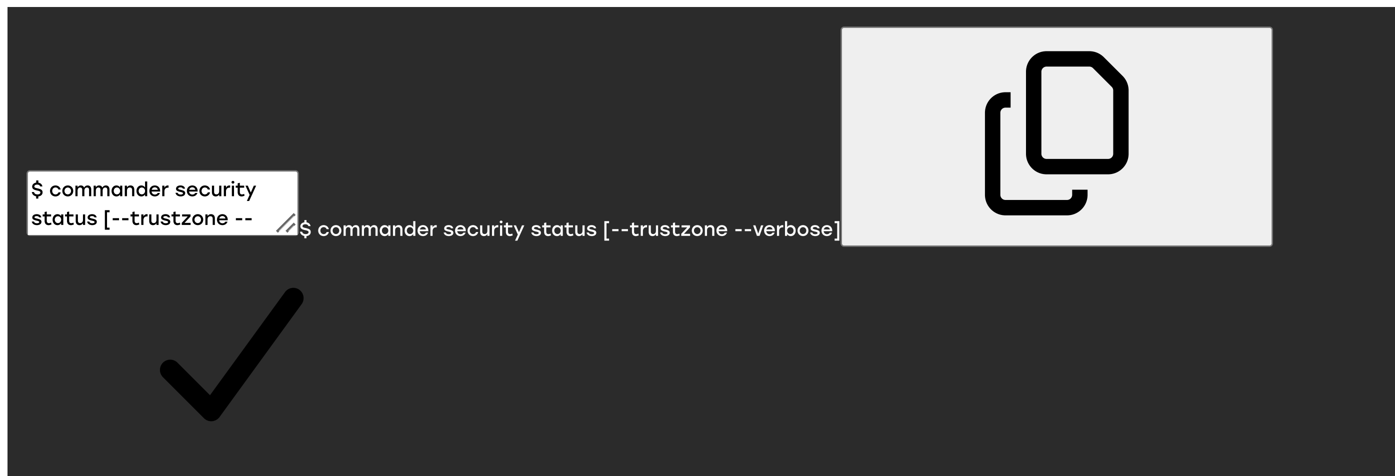
# Security Commands

## Get Device Status

This command prints Secure Engine device information status, including:

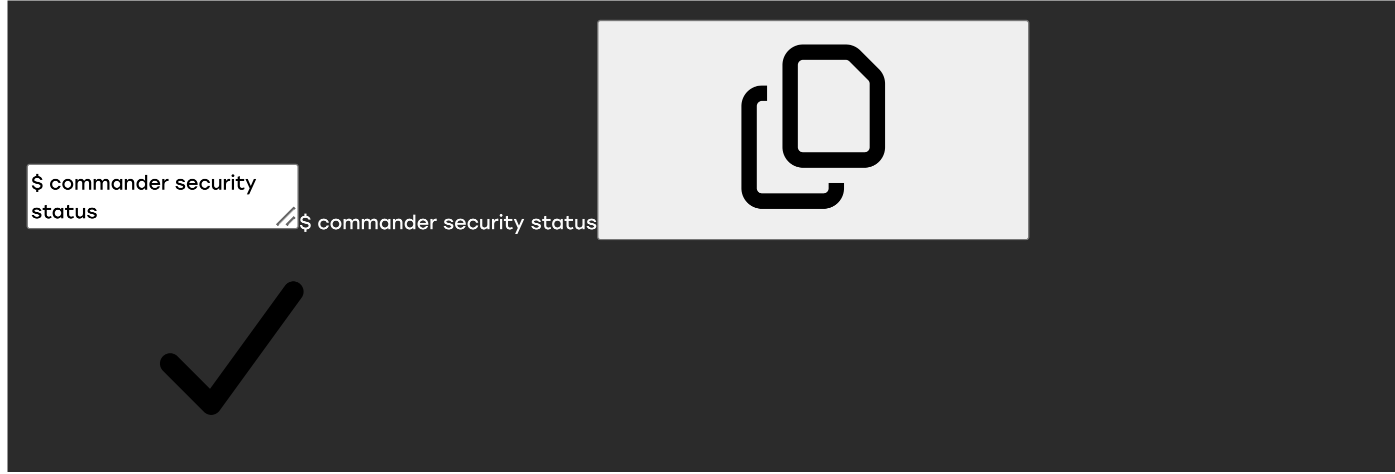
- Firmware version
- Serial number
- Device erase status
- Secure debug unlock status
- Tamper status
- Secure boot status

Command Line Syntax



```
$ commander security status [--trustzone --verbose]
```

Command Line Input Example



```
$ commander security status
```

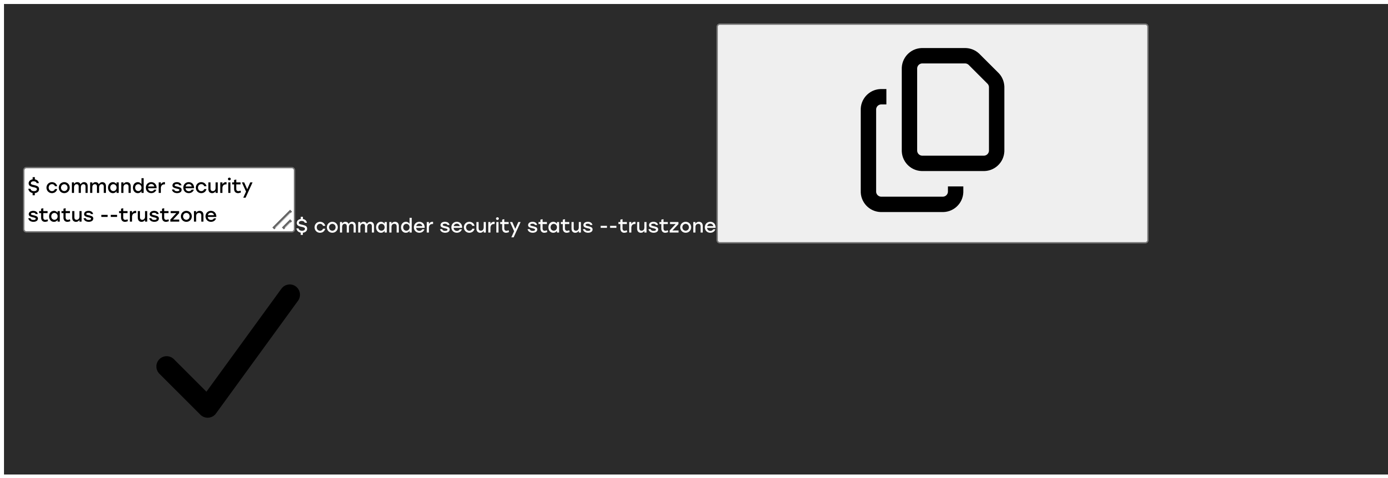
Command Line Output Example

Copyright © 2026 Silicon Laboratories. All rights reserved.

```
SE Firmware version : 1.1.3
Serial number       : 0000000000000000d0cf5efffe68a68b
Debug lock         : Disabled
Device erase        : Enabled
Secure debug unlock : Disabled
Tamper status       : OK
Secure boot         : Disabled
Boot status         : 0x20 - OK
DONE
```

- `Debug lock` : indicates whether debug access is Enabled (locked) or Disabled (unlocked).
- `Device erase` : indicates whether or not it is possible to regain debug access using the `device erase` command. If the device erase is enabled, this is possible. If the device is disabled, it is not possible.
- `Security debug unlock` : Enabled means that if the device is locked, debug access can be regained using the `security unlock` command. If both `device erase` and `secure debug unlock` are Disabled, it is not possible to regain debug access if the device is locked.
- `Tamper status` : indicates whether or not a tamper event is detected by the device. OK means no tamper event is detected.
- `Secure boot` : Enabled means that all images running on the device must be signed with the private sign key corresponding to the public sign key written to the device. Disabled means that images do not have to be signed with the sign key.
- `Boot status` : shows if secure boot failed or if the secure boot is OK.

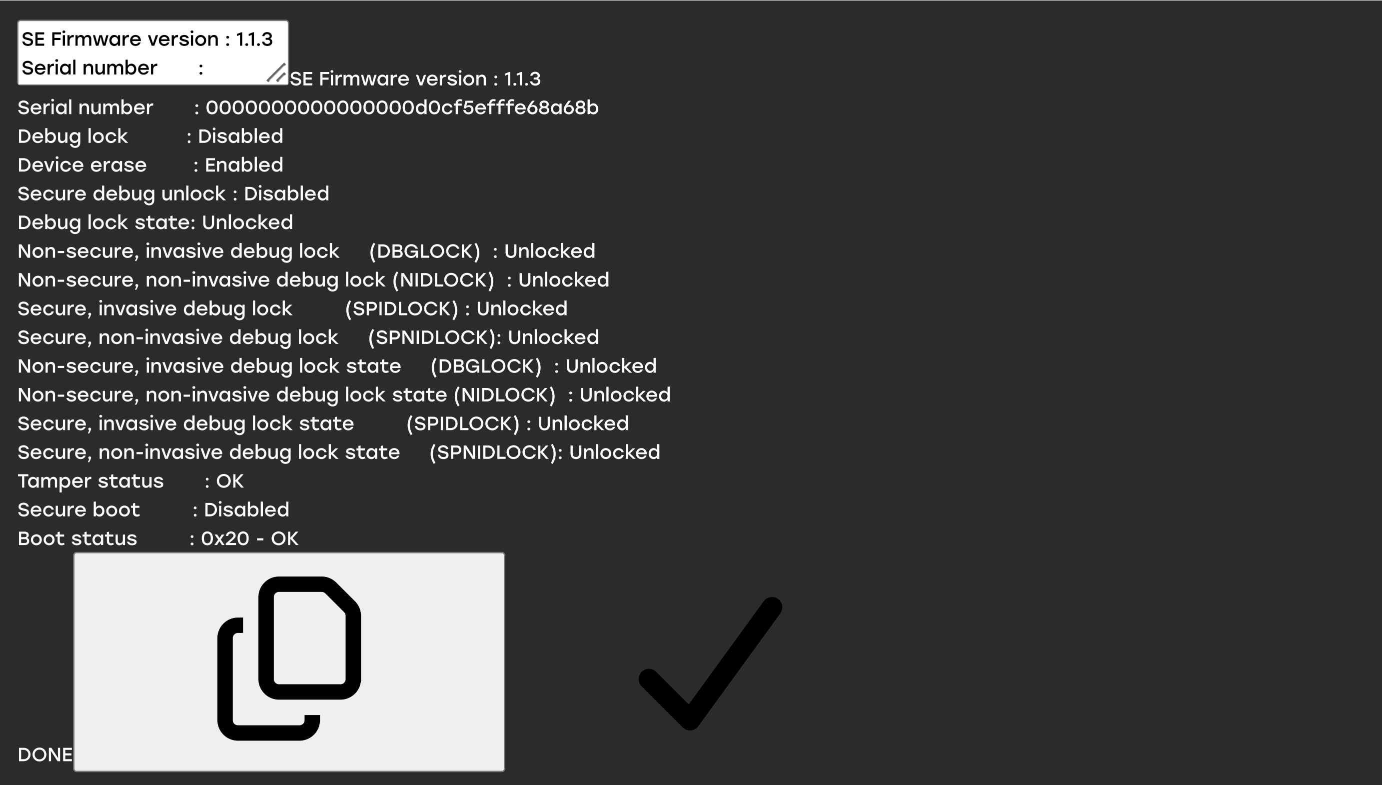
Command Line Input Example



```
$ commander security status --trustzone
```

Show the TrustZone status of the device.

Command Line Output Example

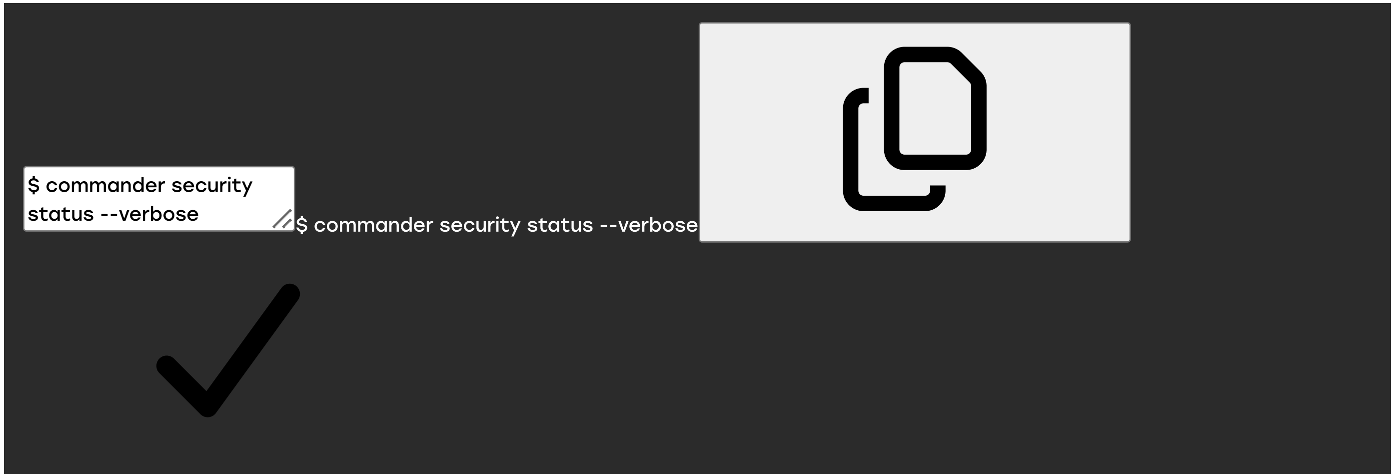


```
SE Firmware version : 1.1.3
Serial number       : 0000000000000000d0cf5efffe68a68b
Debug lock          : Disabled
Device erase        : Enabled
Secure debug unlock : Disabled
Debug lock state: Unlocked
Non-secure, invasive debug lock     (DBGLOCK)  : Unlocked
Non-secure, non-invasive debug lock (NIDLOCK)  : Unlocked
Secure, invasive debug lock         (SPIDLOCK) : Unlocked
Secure, non-invasive debug lock     (SPNIDLOCK): Unlocked
Non-secure, invasive debug lock state     (DBGLOCK)  : Unlocked
Non-secure, non-invasive debug lock state (NIDLOCK)  : Unlocked
Secure, invasive debug lock state         (SPIDLOCK) : Unlocked
Secure, non-invasive debug lock state     (SPNIDLOCK): Unlocked
Tamper status       : OK
Secure boot         : Disabled
Boot status         : 0x20 - OK
DONE
```

`Debug lock state` indicates whether the debug port is locked or unlocked.

The `TrustZone` debug lock configuration consists of the four modes `SPNIDLOCK`, `SPIDLOCK`, `NIDLOCK` and `DBGLOCK`. The top configurations specifies which mode has been locked by the `security lock --trustzone` command. The bottom configuration specifies the actual state of the mode, whether or not it has been unlocked.

Command Line Input Example



```
$ commander security status --verbose
```

Show verbose output of the security status.

Command Line Output Example

Copyright © 2026 Silicon Laboratories. All rights reserved.

```
SE Firmware version : 1.1.3
Serial number       : 0000000000000000d0cf5efffe68a68b
Debug lock          : Disabled
Device erase        : Enabled
Secure debug unlock : Disabled
Tamper status       : OK
Secure boot         : Disabled
Boot status         : 0x20 - OK
Verbose output: 00000000 00000000 00000000 FFFFFFFF 00000020 03020200 00000000 00000002 FFFFFFFF
DONE
```

`Verbose output` is the entire output from the Secure Engine of the device.

## Generate Key Pair

This command has been deprecated. For more information on how to generate keys, see Generate a Signing Key and Key Generation.

## Write Public Key to Device

IMPORTANT: This is a one-time command. It cannot be run more than once per device.

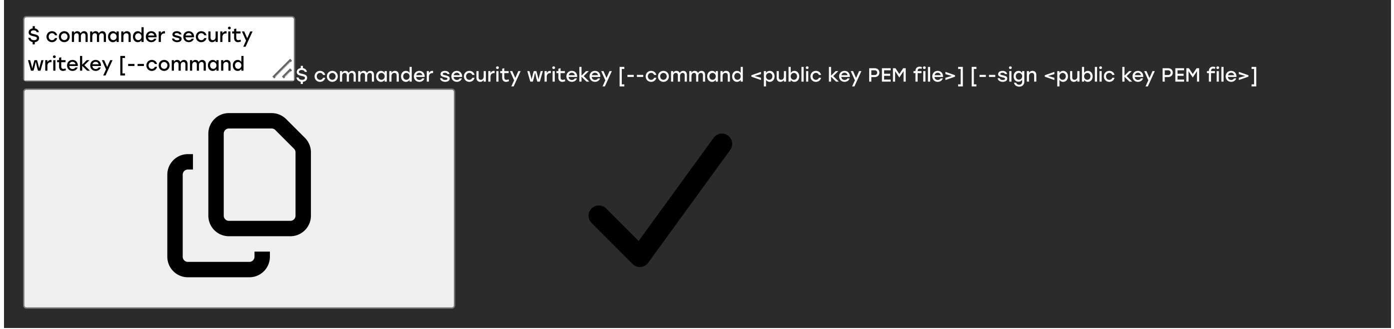
This one-time command permanently locks the device to this key pair. There are two different public keys that can be written to the device.

- Command key: the corresponding private key is used to create certificates to perform secure debug unlock.
- Sign key: the corresponding private key must sign all code that is to run on the device when Secure Boot is enabled.

When Secure Debug Unlock is enabled, a locked device may temporarily unlock debug access by creating a certificate signed by the private command key.

When Secure Boot is enabled, all code that runs on the device must be signed by the private sign key.

Command Line Syntax



```
$ commander security writekey [--command <public key PEM file>] [--sign <public key PEM file>]
```

Command Line Input Example

Copyright © 2026 Silicon Laboratories. All rights reserved.

```
$ commander security writekey --command command_public_key.pem
```

Command Line Output Example

```
Device has serial number 000000000000000014b457fffed50c35
================================================================================
Please look through any warnings before proceeding.
THIS IS A ONE-TIME command, all code to be run on the device must be signed by this key.
Type 'continue' and hit enter to proceed or Ctrl-C to abort:
================================================================================
continue
DONE
```

## Read Public Key from Device

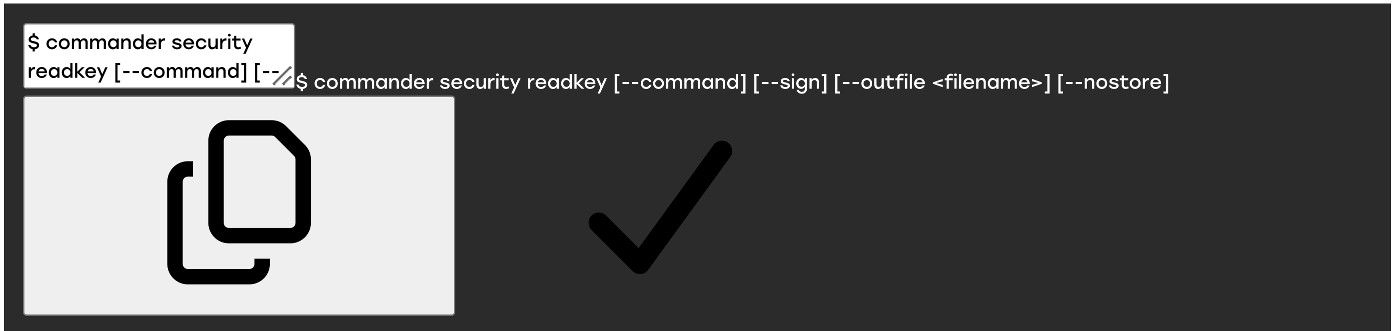
This command reads out a public key from the device. There are two different public keys that can be stored on the device using the `commander security writekey` command.

- Command key: the corresponding private key is used to create certificates to perform secure debug unlock or disable tamper.
- Sign key: the corresponding private key must sign all code that is to run on the device when Secure Boot is enabled.

By providing an output file, the key will be written to the file. Otherwise, the key will be printed to the Command Line Interface (CLI) as a byte array.

If the optional `--nostore` option is not used, the key will also be stored in the Security Store.

Command Line Syntax

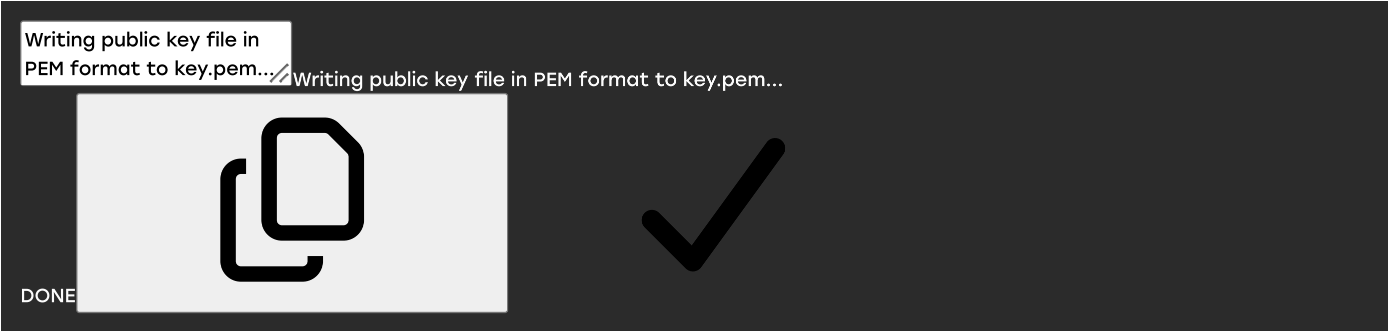


```
$ commander security readkey [--command] [--sign] [--outfile <filename>] [--nostore]
```

Command Line Input Example

```
$ commander readkey --command --outfile command_public_key.pem
```

Command Line Output Example



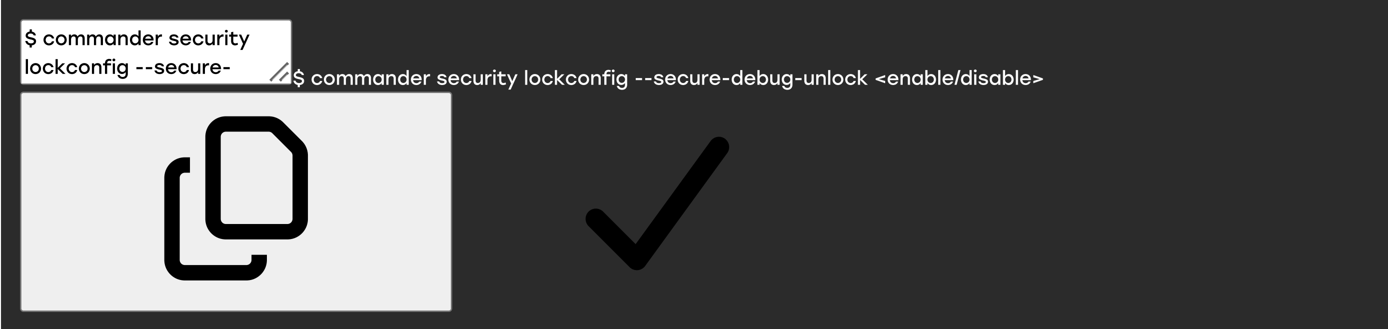
```
Writing public key file in PEM format to key.pem...
DONE
```

## Configure Lock Options

The `security lockconfig` command enables or disables secure debug unlock. When secure debug unlock is enabled, a locked device may be temporarily unlocked by running a `commander security unlock` command. If secure debug unlock is disabled, the only way to unlock a locked device is to run a `commander security erasedevice` command, given that device erase has not been disabled. If both device erase and secure debug unlock are disabled, there is no way to unlock debug access to a locked device.

> Note: Secure debug unlock must be enabled before the device is locked.

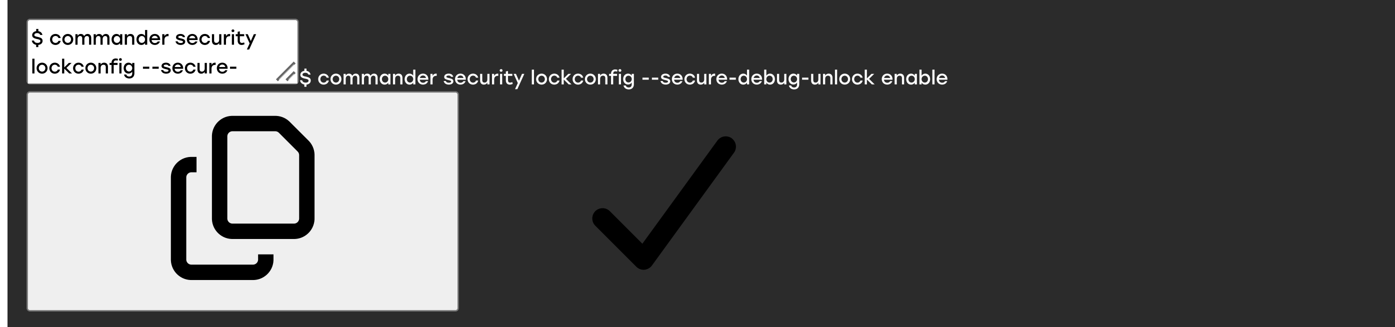
Command Line Syntax



```
$ commander security lockconfig --secure-debug-unlock <enable/disable>
```

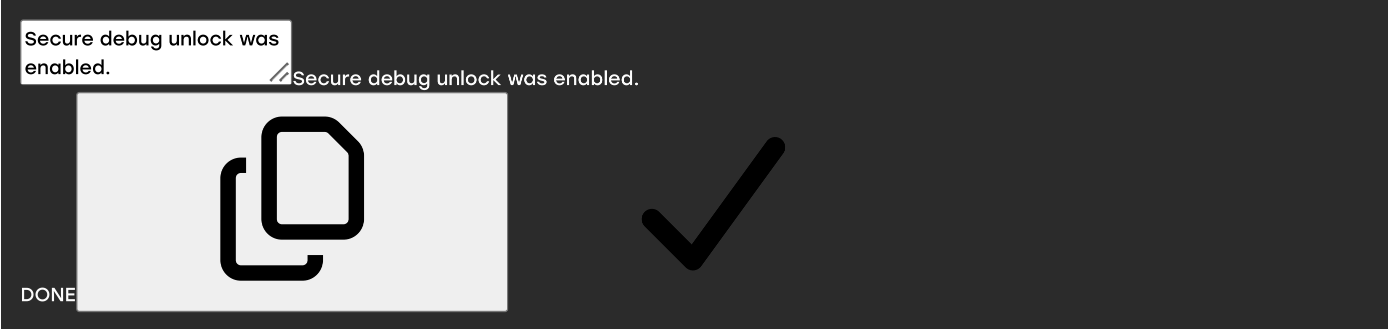
Command Line Input Example

Copyright © 2026 Silicon Laboratories. All rights reserved.



```
$ commander security lockconfig --secure-debug-unlock enable
```

Command Line Output Example



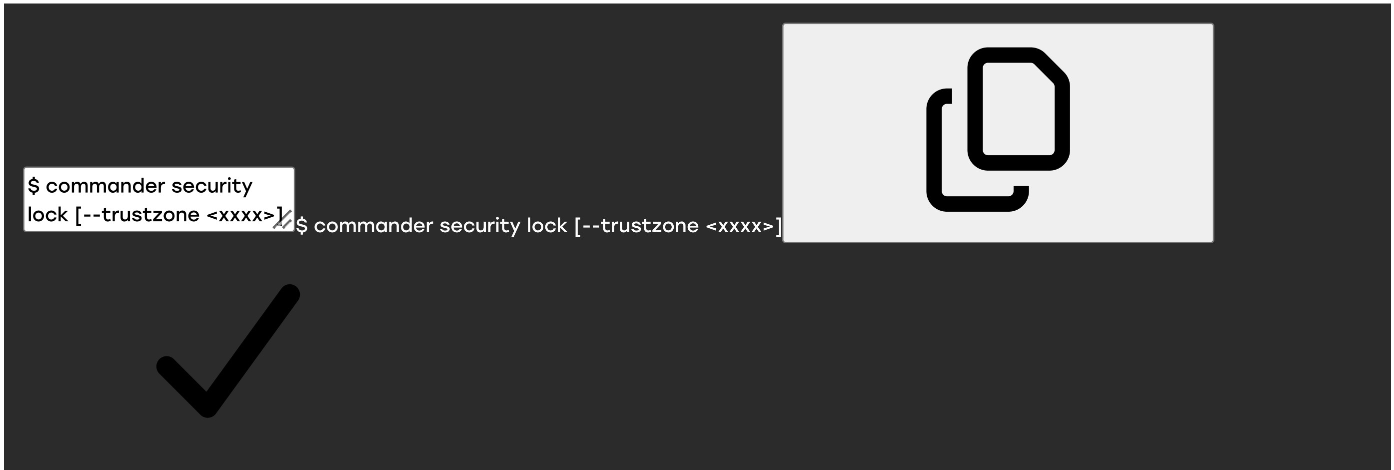
```
Secure debug unlock was enabled.
DONE
```

## Lock Debug Access

The `lock` command locks the debug interface on the device. If secure debug unlock has been enabled, the device may be unlocked using the `unlock` command. If device erase has not been disabled, the debug access may also be unlocked using the `commander security erasedevice` command. However, this also triggers a mass erase on the device.

The `--trustzone` option may be used to lock debug access to specific TrustZone modes. The bitmask to set TrustZone debug lock is defined as `<SPNIDLOCK, SPIDLOCK, NIDLOCK, DBGLOCK>`. If the bit is set to 1, debug access to the corresponding TrustZone mode will be locked. Set the bit to 0 to keep it open. By default all modes are open.

Command Line Syntax



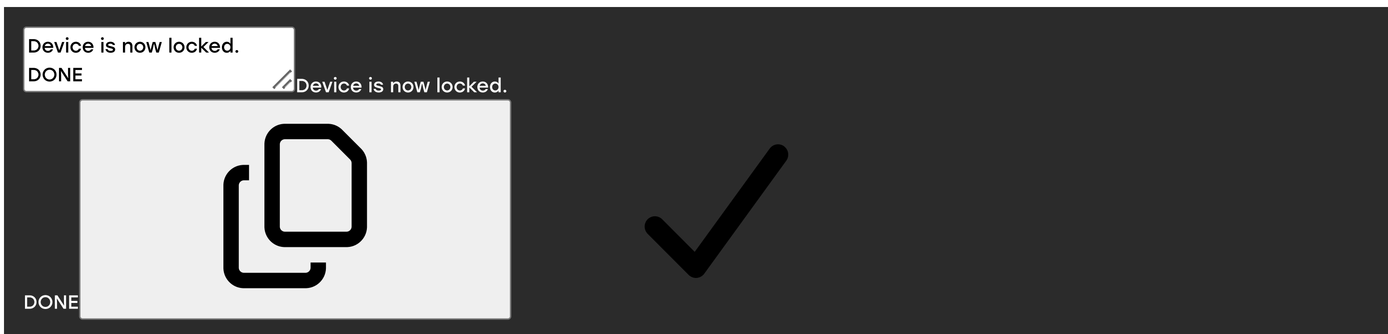
```
$ commander security lock [--trustzone <xxxx>]
```

Copyright © 2026 Silicon Laboratories. All rights reserved.

Command Line Input Example

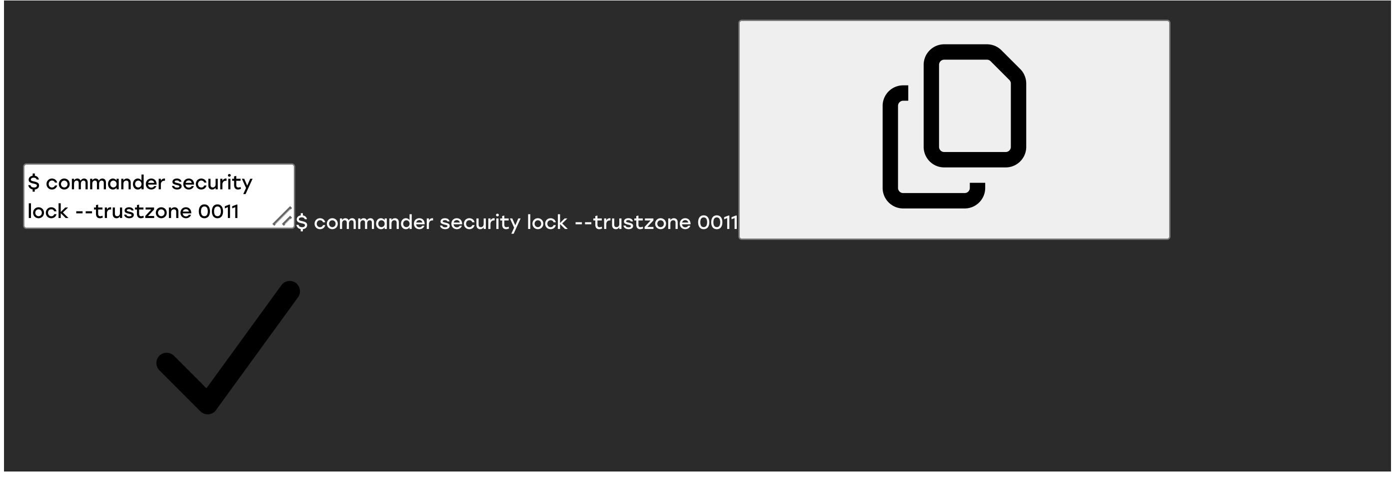
```
$ commander security lock
```

Command Line Output Example



```
Device is now locked.
DONE
```

Command Line Input Example

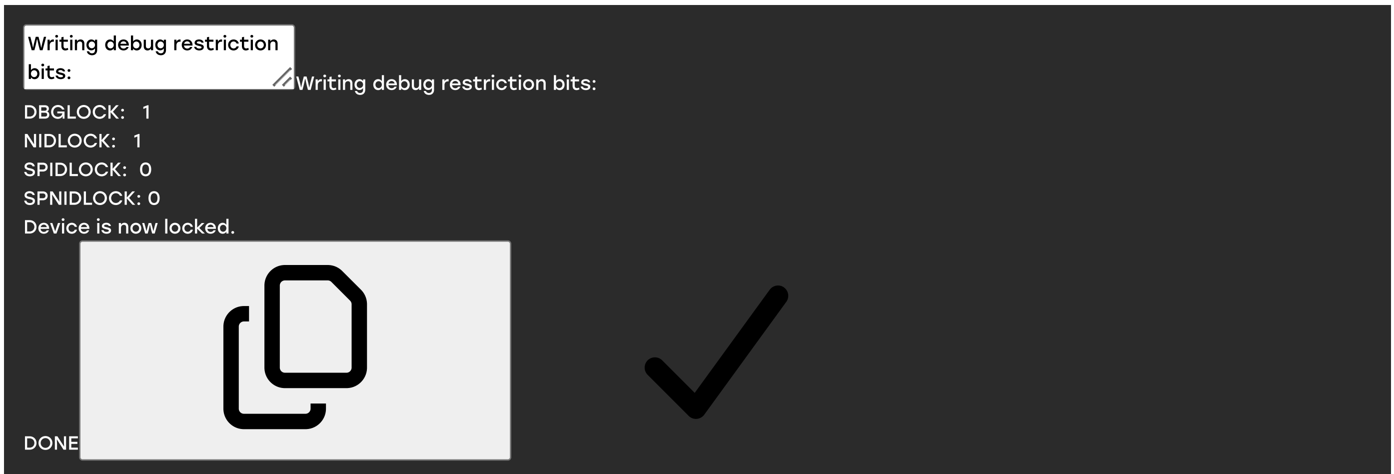


```
$ commander security lock --trustzone 0011
```

Debug access to TrustZone modes `DBGLOCK` and `NIDLOCK` are locked.

Command Line Output Example


Copyright © 2026 Silicon Laboratories. All rights reserved.



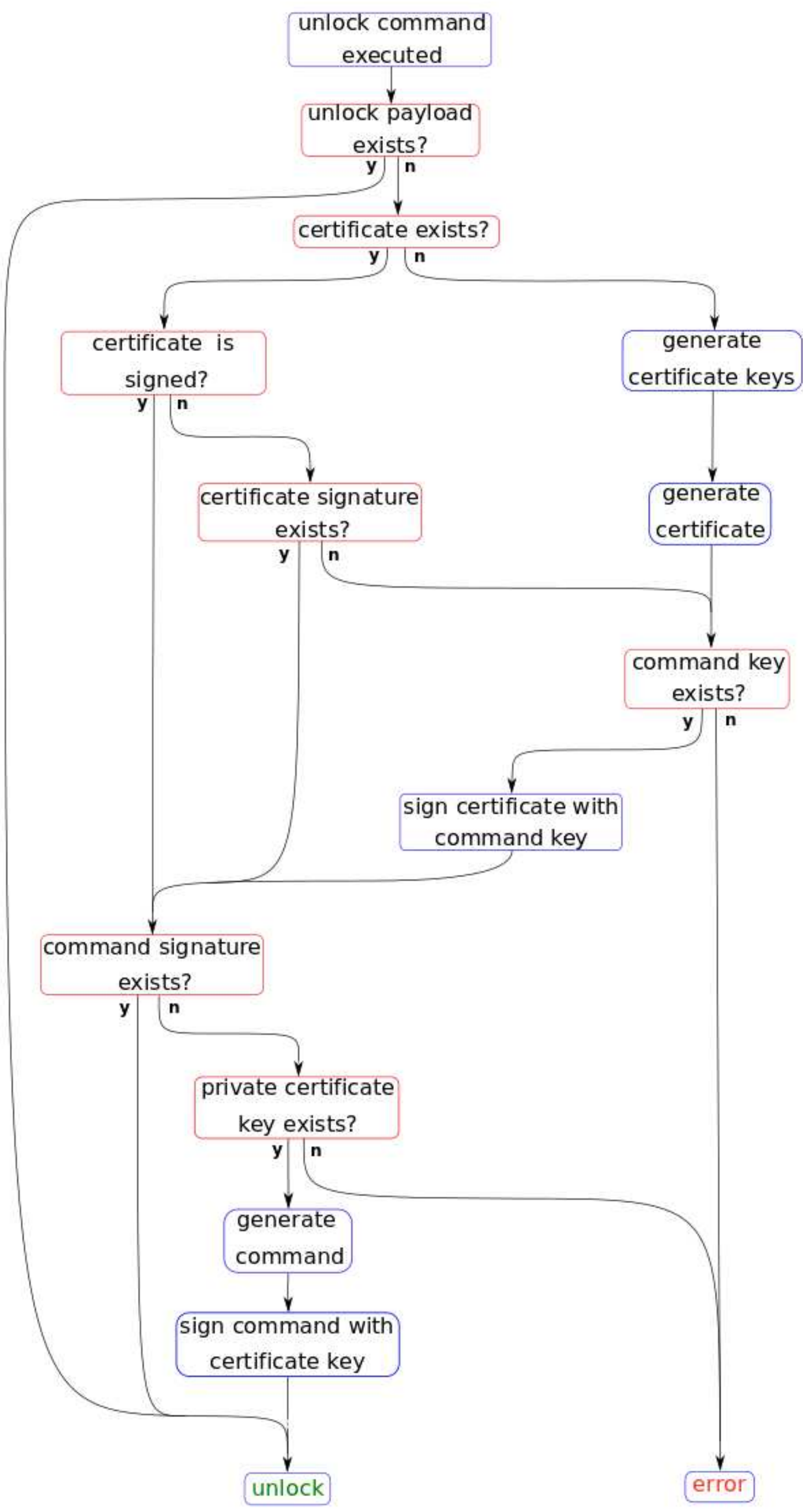

## Secure Debug Unlock

The security unlock command opens debug access on a locked device temporarily without erasing the flash content. When running the `commander security unlock` command, Simplicity Commander will use all available files in the Security Store and from command line options in an attempt to unlock debug access. If anything is missing, you will be asked to provide the file as an option to the command. All files generated or given as command line options are stored in the Security Store, unless the `--nostore` option is used.

For more information about Secure Debug, see *AN1190: EFR32xG21 Secure Debug*.

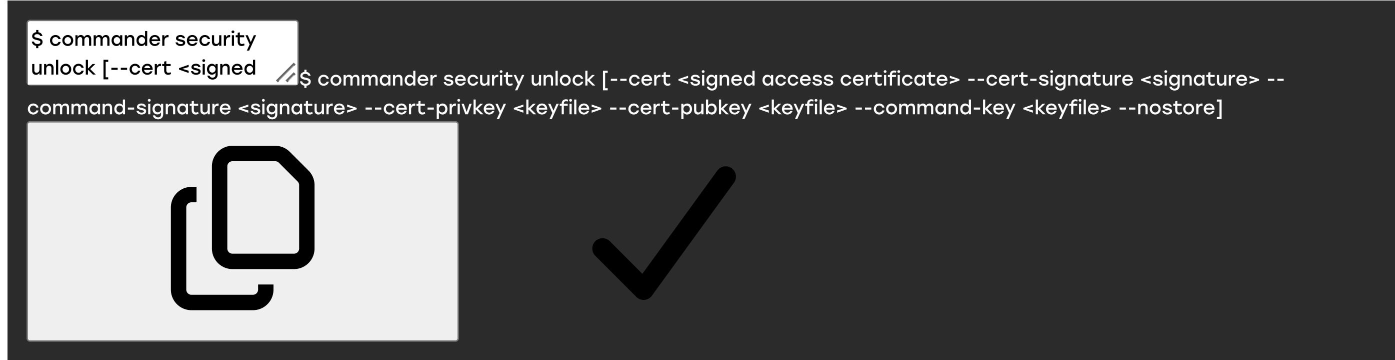
There are several different ways to unlock the debug access, as illustrated in the following figure. The blue fields are actions and the red fields are artifacts.

Copyright © 2026 Silicon Laboratories. All rights reserved.



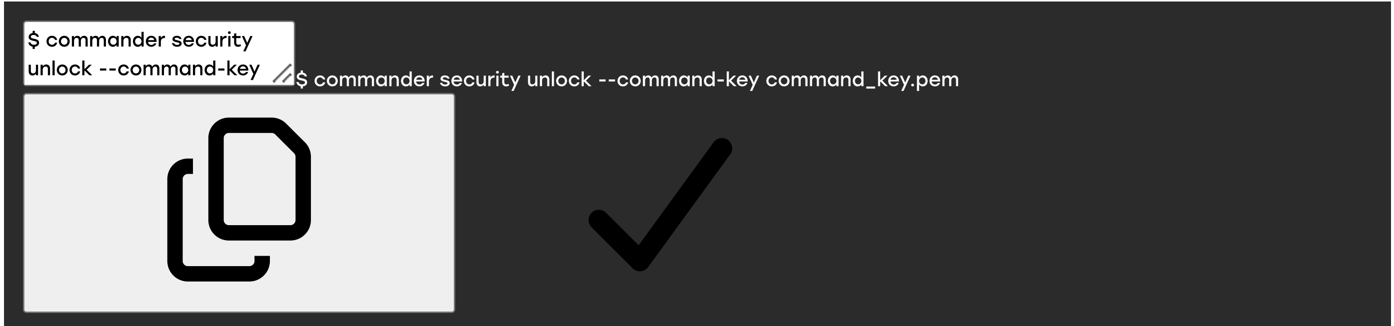

Command Line Syntax

Copyright © 2026 Silicon Laboratories. All rights reserved.



```
$ commander security unlock [--cert <signed access certificate> --cert-signature <signature> --command-signature <signature> --cert-privkey <keyfile> --cert-pubkey <keyfile> --command-key <keyfile> --nostore]
```

Command Line Input Example

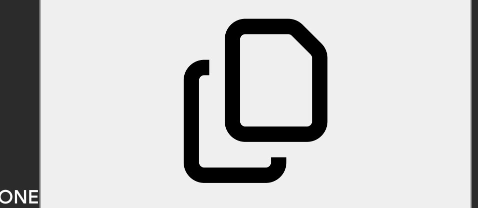


```
$ commander security unlock --command-key command_key.pem
```

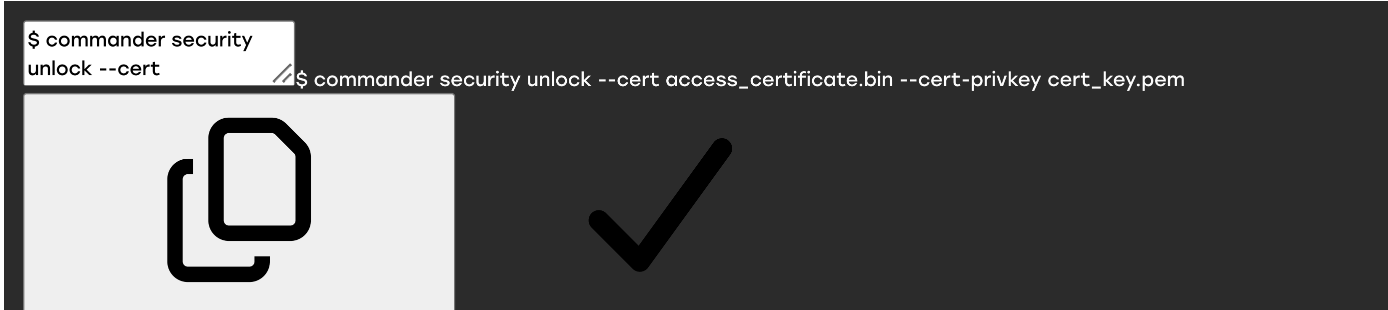
This example uses and generates a certificate and command signature on-the-fly using the provided command key to sign the certificate. All the generated files and the command key are stored in the Security Store.

Command Line Output Example

```
Command public key stored in:
/Users/example/Library/Preferences/SiliconLabs/commander/SecurityStore/device_0000000000000000d0cf5efffe68a68b/command_pu

Command private key stored in:
/Users/example/Library/Preferences/SiliconLabs/commander/SecurityStore/device_0000000000000000d0cf5efffe68a68b/command_ke

Authorization file written to Security Store:
/Users/example/Library/Preferences/SiliconLabs/commander/SecurityStore/device_0000000000000000d0cf5efffe68a68b/certificate_a

Generating ECC P256 key pair...
Cert public key stored at:
/Users/example/Library/Preferences/SiliconLabs/commander/SecurityStore/device_0000000000000000d0cf5efffe68a68b/cert_pubkey.p

Cert private key stored at:
/Users/example/Library/Preferences/SiliconLabs/commander/SecurityStore/device_0000000000000000d0cf5efffe68a68b/cert_key.pem

Command key matches public command key found on device. Signing certificate...
Certificate was signed with key:
test-cases/common/security_testfiles/command_key.pem
Successfully stored certificate
Certificate written to Security Store:
/Users/example/Library/Preferences/SiliconLabs/commander/SecurityStore/device_0000000000000000d0cf5efffe68a68b/access_certif

Created unsigned unlock command
Signed unlock command using
/Users/example/Library/Preferences/SiliconLabs/commander/SecurityStore/device_0000000000000000d0cf5efffe68a68b/cert_key.pem

Secure debug successfully unlocked
Command unlock payload was stored in Security Store
DONE
```



Command Line Input Example

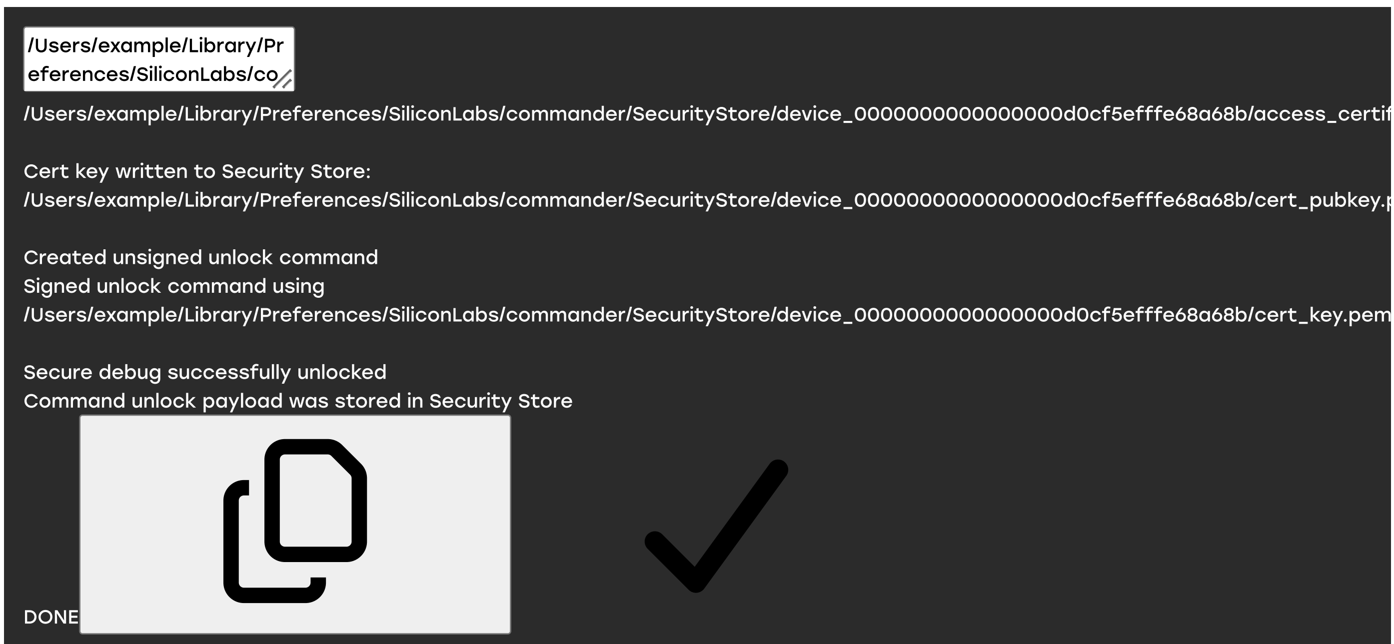


```
$ commander security unlock --cert access_certificate.bin --cert-privkey cert_key.pem
```

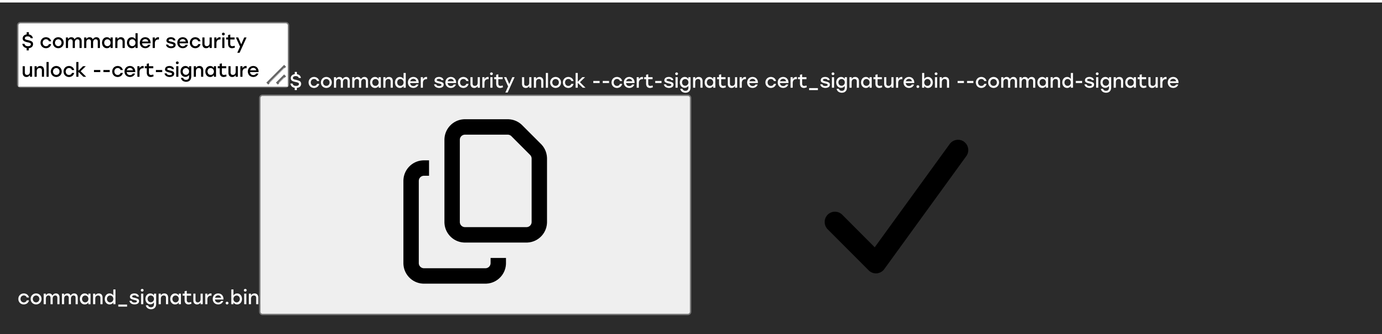
This example unlocks the device with a signed access certificate and the private certificate key corresponding to the public key in the access certificate. The certificate and key are stored in the Security Store.

Command Line Output Example

Copyright © 2026 Silicon Laboratories. All rights reserved.




Command Line Input Example



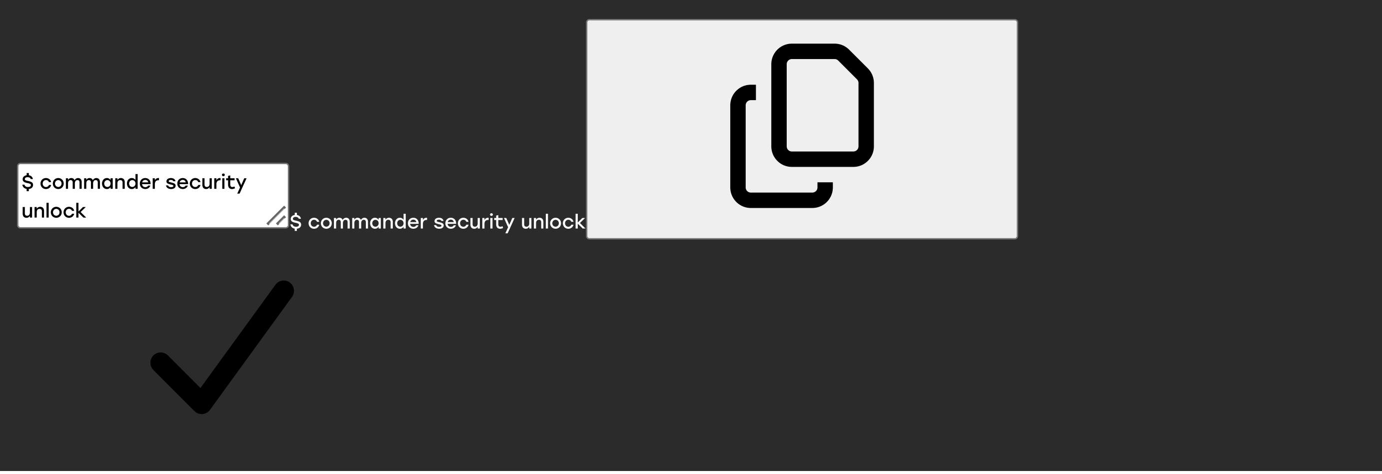

This example uses externally generated signatures for both the access certificate and command file. The access certificate signature is appended to the certificate and stored in the Security Store. The command signature is validated against the public key in the certificate.

Command Line Output Example

Copyright © 2026 Silicon Laboratories. All rights reserved.

```
Using certificate from Security Store:
/Users/example/Library/Preferences/SiliconLabs/commander/SecurityStore/device_0000000000000000d0cf5efffe68a68b/access_certif

Certificate in Security Store is not signed.
Moved existing file to:
/Users/example/Library/Preferences/SiliconLabs/commander/SecurityStore/device_0000000000000000d0cf5efffe68a68b/archive/acces

Signed certificate written to Security Store:
/Users/example/Library/Preferences/SiliconLabs/commander/SecurityStore/device_0000000000000000d0cf5efffe68a68b/access_certif

Command signature is valid
Secure debug successfully unlocked
Command unlock payload was stored in Security Store
```

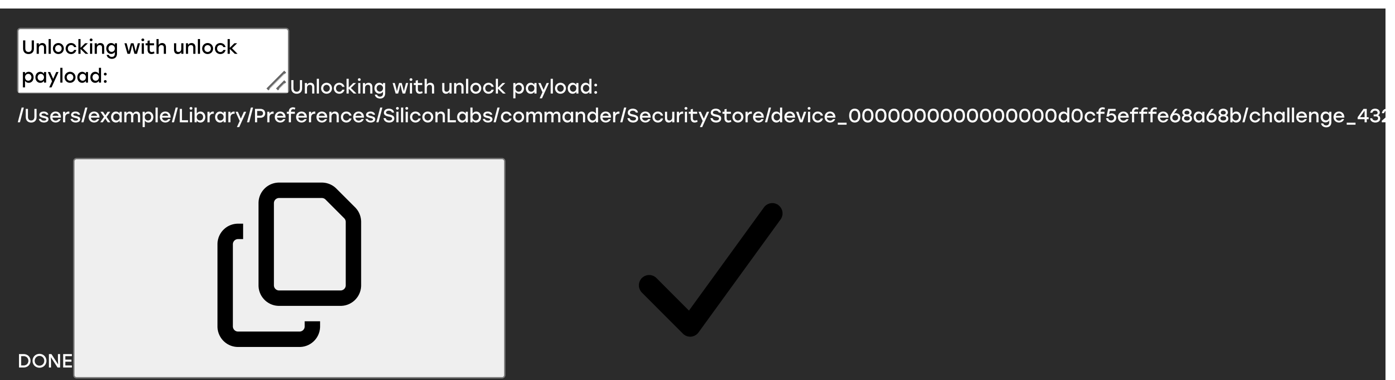
Command Line Input Example



```
$ commander security unlock
```

When the device has been unlocked with the current challenge, the unlock payload is stored in the Security Store. The next time the unlock command is run, the device is unlocked directly with the unlock payload.

Command Line Output Example



```
Unlocking with unlock payload:
/Users/example/Library/Preferences/SiliconLabs/commander/SecurityStore/device_0000000000000000d0cf5efffe68a68b/challenge_43
DONE
```

Copyright © 2026 Silicon Laboratories. All rights reserved.

## Disable Tamper

Secure Vault products are capable of detecting certain types of tamper events and responding to mitigate the attack. This provides an extra layer of protection against attacks that rely on physically tampering with the product.

Before this command can be executed, the tamper sources must be configured in the One-Time-Programmable (OTP) settings of the devices. See Write User Configuration for more information about how this is done.

The process of disabling tamper follows the same flow as the `security unlock` command. For more information about the flow, see Secure Debug Unlock.

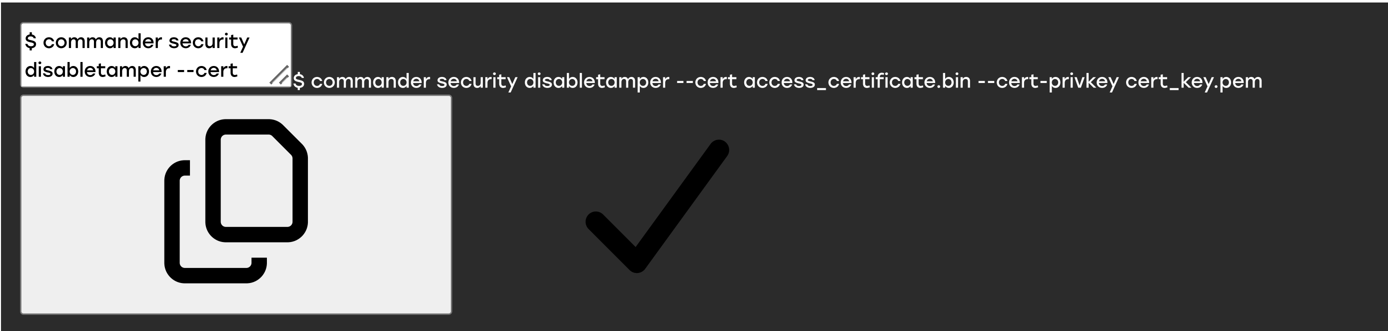
A certificate and a signed challenge are required to disable tamper. The certificate–including tamper authorizations–is generated and signed with a command key. The certificate contains a public key and the corresponding private key must be used to sign a challenge from the device to disable tamper sources. The `--disable-param` option determines which tamper sources to disable. If this option is not provided, Simplicity Commander will extract the tamper authorizations from the certificate and disable everything allowed by the certificate. If the certificate is not available, all sources will be disabled.

The tamper sources are disabled until the next Power On Reset.

Command Line Syntax

```
$ commander security disabletamper [--disable-param <disable-mask> --cert <signed access certificate> --cert-signature <signature> --commandsignature <signature> --cert-privkey <keyfile> --cert-pubkey <keyfile> --command-key <keyfile> --nostore]
```

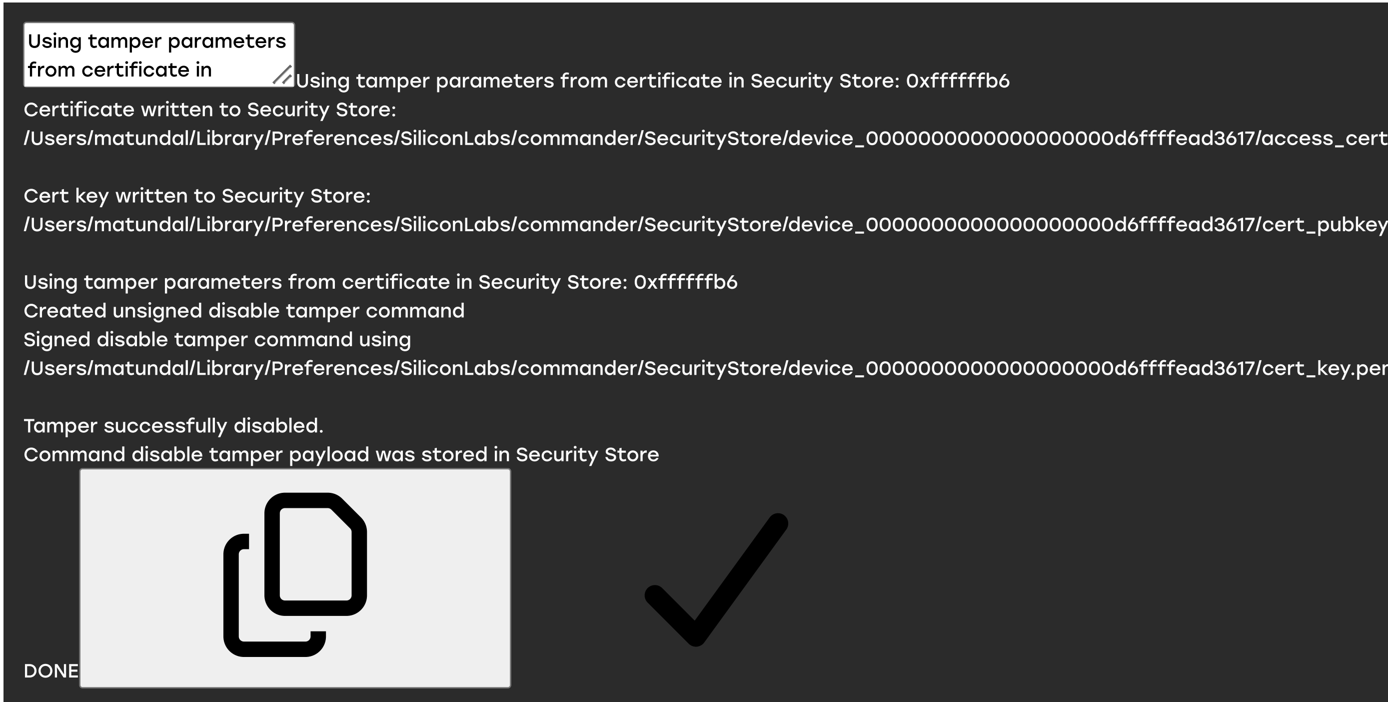
Command Line Input Example



```
$ commander security disabletamper --cert access_certificate.bin --cert-privkey cert_key.pem
```

Command Line Output Example

Copyright © 2026 Silicon Laboratories. All rights reserved.



```
Using tamper parameters from certificate in Security Store: 0xffffffb6
Certificate written to Security Store:
/Users/matundal/Library/Preferences/SiliconLabs/commander/SecurityStore/device_0000000000000000d6ffffead3617/access_certi

Cert key written to Security Store:
/Users/matundal/Library/Preferences/SiliconLabs/commander/SecurityStore/device_0000000000000000d6ffffead3617/cert_pubkey.

Using tamper parameters from certificate in Security Store: 0xffffffb6
Created unsigned disable tamper command
Signed disable tamper command using
/Users/matundal/Library/Preferences/SiliconLabs/commander/SecurityStore/device_0000000000000000d6ffffead3617/cert_key.pen

Tamper successfully disabled.
Command disable tamper payload was stored in Security Store
DONE
```

## Device Erase using Secure Engine

This command performs a device mass erase and resets the debug configuration to its initial unlocked state.

The complete flash and RAM of the system is cleared, excluding the user data page and one-time programmable commissioning information in the Secure Engine.

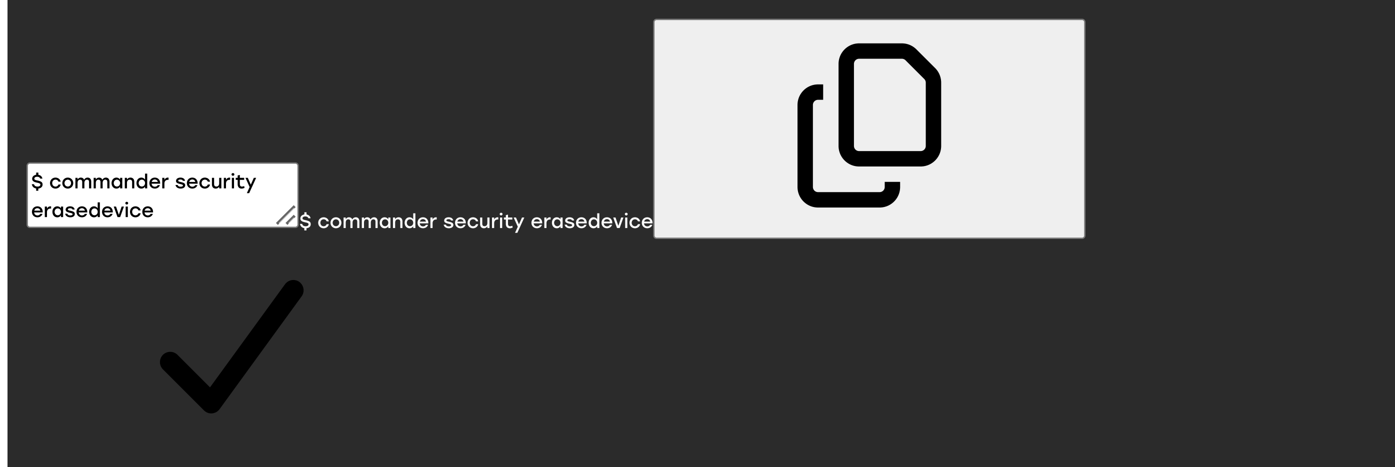
If device erase has been disabled, this command is not available.

> Note: After a device erase, the DCI interface is unavailable until the device has been reset

Command Line Syntax

```
$ commander security erasedevice
```

Command Line Input Example




Command Line Output Example

## Disable Device Erase

IMPORTANT: This is a one-time command. It cannot be run more than once.

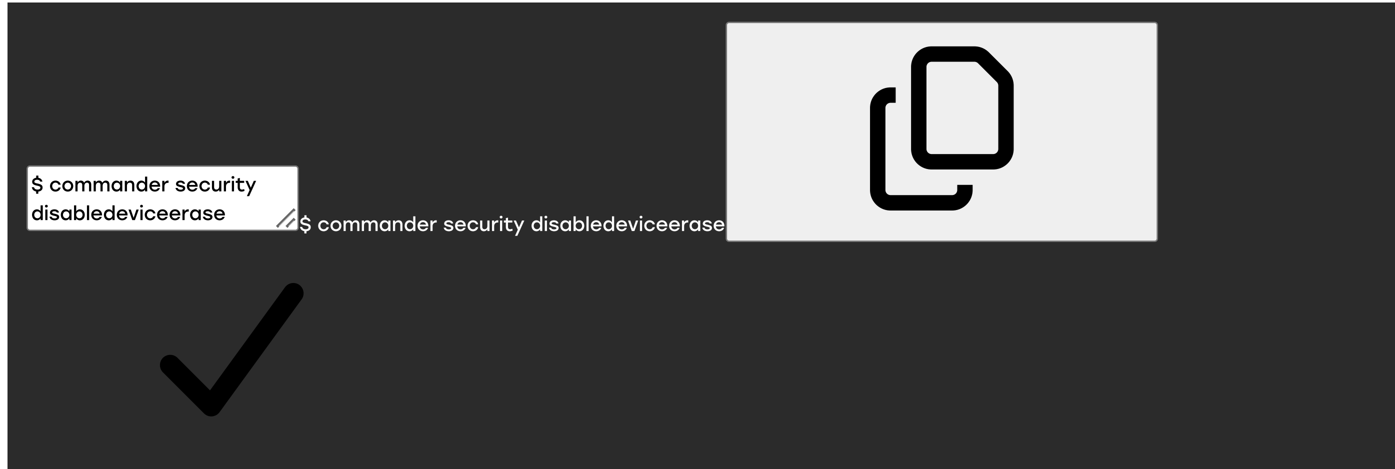
This command permanently disables device erase. When device erase is disabled, the `commander security erasedevice` command is no longer available. This means that if debug access is locked, debug access can only be opened if secure debug unlock has been enabled before the device was locked. If not, there is no way to regain debug access. This command can be run after the device has been locked.

Confirmation is required from the user to execute this command, except if the `--noprompt` option is used.

Command Line Syntax

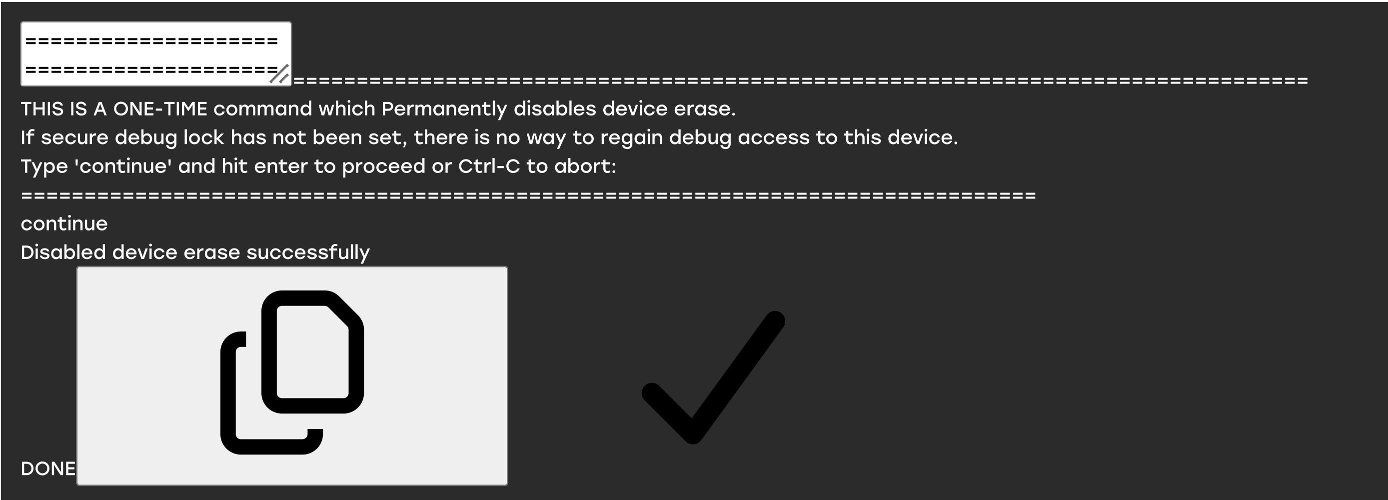
Command Line Input Example

Copyright © 2026 Silicon Laboratories. All rights reserved.



```
$ commander security disabledeviceerase
```

Command Line Output Example

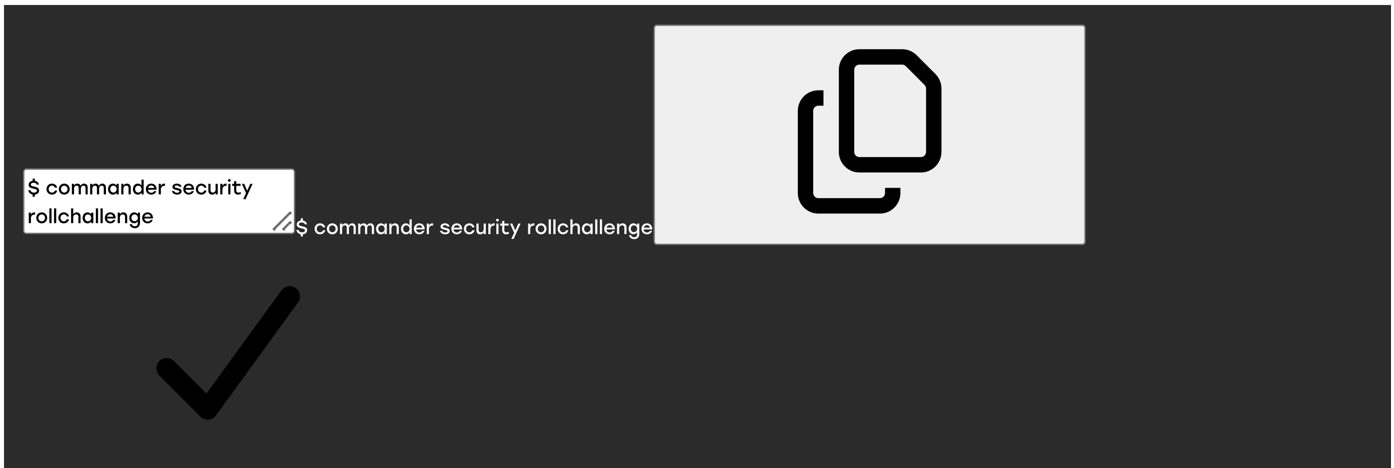


```
=====================================================================================
THIS IS A ONE-TIME command which Permanently disables device erase.
If secure debug lock has not been set, there is no way to regain debug access to this device.
Type 'continue' and hit enter to proceed or Ctrl-C to abort:
=====================================================================================
continue
Disabled device erase successfully
DONE
```

## Roll Challenge

This command makes the Secure Engine *roll* or update its challenge data. The challenge is random data that must be read from the device before an unlock command can be executed. Rolling the challenge renders existing command signatures invalid. For more information, see Challenge and Command Signing.

The challenge cannot be rolled before it has been used at least once; that is, by running the security unlock command or the disable tamper command.



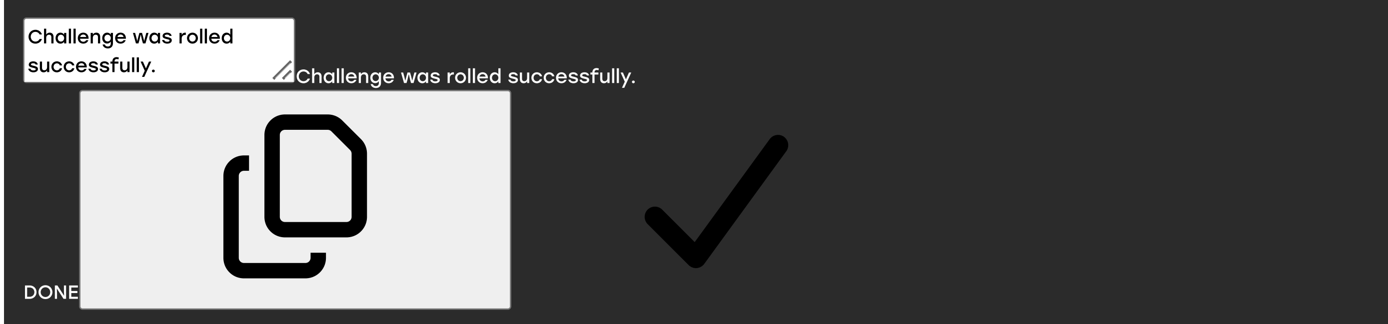
```
$ commander security rollchallenge
```

Copyright © 2026 Silicon Laboratories. All rights reserved.

Command Line Input Example

```
$ commander security rollchallenge
```

Command Line Output Example



```
Challenge was rolled successfully.
DONE
```

## Read Challenge

This command reads the current Secure Engine challenge from the device. The challenge is random data that must be known before creating signed unlock or tamper-disable commands. For more information, see Challenge and Command Signing.

Command Line Syntax

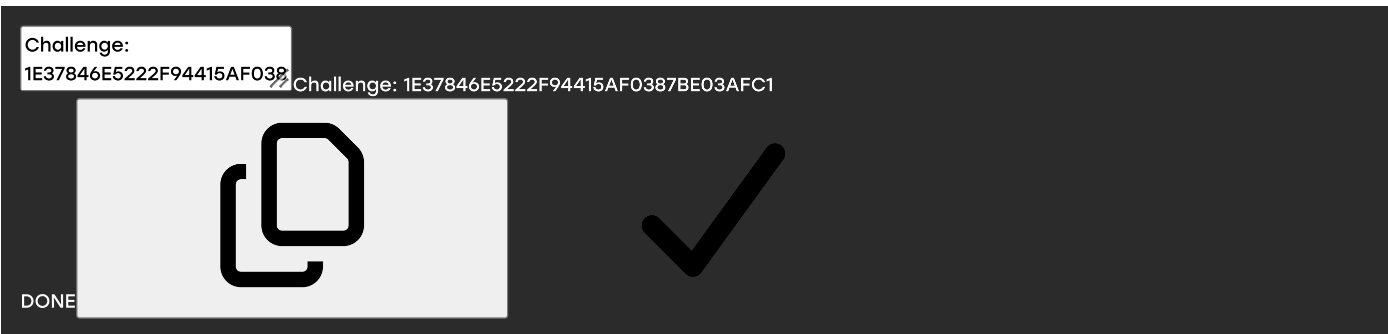
```
$ commander security readchallenge
```

Command Line Input Example

Copyright © 2026 Silicon Laboratories. All rights reserved.

```
$ commander security readchallenge
```

Command Line Output Example



```
Challenge: 1E37846E5222F94415AF0387BE03AFC1
DONE
```

## Generate Example Authorization File

This command generates a default authorization file to be used in the certificate. The authorization file will be stored in Security Store unless the `--nostore` option is used.

Default Authorization File for Devices without Secure Vault

```
{
  "debug_authorizations":{
    "ENABLE_DEBUG_PORT": true
  }
}
```

Default Authorization File for Devices with Secure Vault


Copyright © 2026 Silicon Laboratories. All rights reserved.

```
{
{
  "debug_authorizations":{
    "ENABLE_DEBUG_PORT": true
  },
  "tamper_authorizations":{
    "FILTER_COUNTER": 1,
    "WATCHDOG": 1,
    "SE_RAM_CRC": 1,
    "SE_HARDFAULT": 1,
    "SOFTWARE_ASSERTION": 1,
    "SE_CODE_AUTH": 1,
    "USER_CODE_AUTH": 1,
    "MAILBOX_AUTH": 1,
    "DCI_AUTH": 1,
    "OTP_READ": 1,
    "AUTO_CODE_AUTH": 1,
    "SELF_TEST": 1,
    "TRNG_MONITOR": 1,
    "PRS0": 1,
    "PRS1": 1,
    "PRS2": 1,
    "PRS3": 1,
    "PRS4": 1,
    "PRS5": 1,
    "PRS6": 1,
    "PRS7": 1,
    "DECOUPLE_BOD": 1,
    "TEMP_SENSOR": 1,
    "VGLITCH_FALLING": 1,
    "VGLITCH_RISING": 1,
    "SECURE_LOCK": 1,
    "SE_DEBUG": 1,
    "DGLITCH": 1,
    "SE_ICACHE": 1
  }
}
```


Debug Authorization

Enable Debug Port must be set to *true* in order to perform a secure debug unlock. For more information about secure debug unlock, see Secure Debug Unlock.

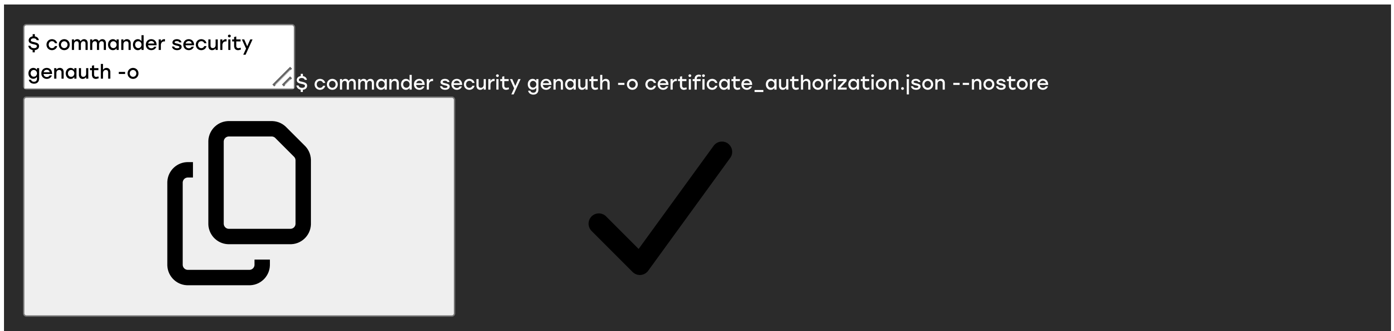
Tamper Authorizations

The Tamper Authorizations indicate which sources may be disabled. By default all sources may be disabled. For more information about disabling tamper sources, see Disable Tamper.

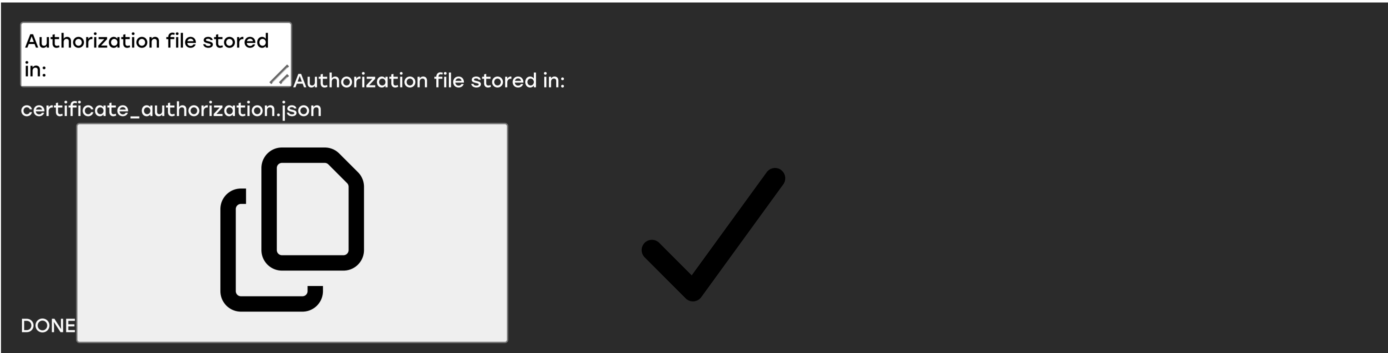
Command Line Syntax

Copyright © 2026 Silicon Laboratories. All rights reserved.

Command Line Input Example




Command Line Output Example




## Generate Access Certificate

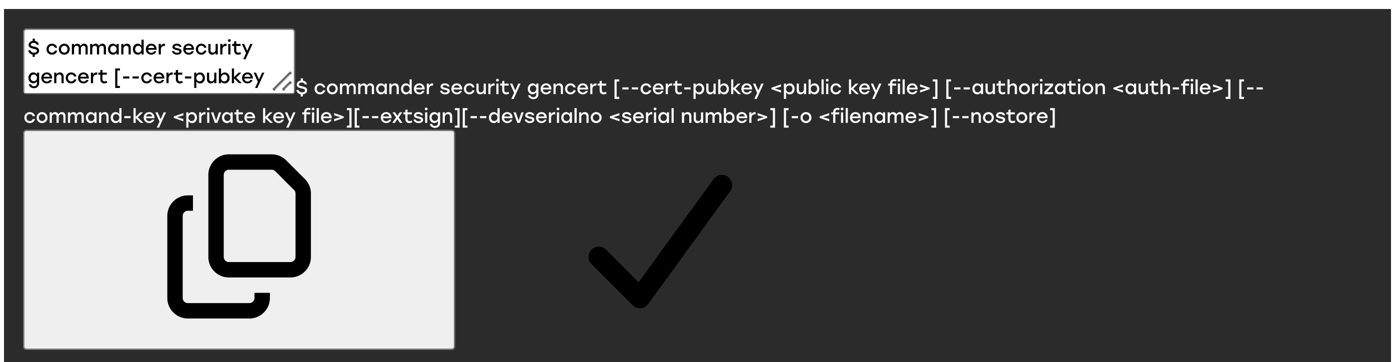
Access certificates are used to unlock debug access or disable tamper on the device. For more information, see Secure Debug Unlock or Disable Tamper. The certificate and the keys provided to or generated by Simplicity Commander are stored in Security Store unless the `--nostore` option is used. If `--cert-pubkey` or `--authorization` are not used as options on the command line, Simplicity Commander checks if the files are stored in Security Store. If the files are not in Security Store, Simplicity Commander generates a default authorization file that may be edited. If the file is edited, a new certificate must be generated. Simplicity Commander will also generate a pair of certificate keys if the `--cert-pubkey` option is not used. If the certificate keys are generated, the `--nostore` option cannot be used. If the `--command-key` option is not used on the command line and not located in Security Store, the `--extsign` option should be used for Simplicity Commander to generate an unsigned certificate. To use the certificate to unlock debug access, a certificate signature must be generated and provided. If the device for which the certificate is made is connected, Simplicity Commander retrieves the device serial number directly.

Copyright © 2026 Silicon Laboratories. All rights reserved.

> Note: Before Simplicity Commander version 1.11.2 unsigned certificates were created with all zeros in replace of the signature. This was fixed in version 1.11.2 making it compatible with external signing using tools such as OpenSSL.

| Device Serial Number |
| --- |
| Authorization |
| Certificate Public Key |
| Access Certificate Signature<br>Signed by Command private key |

Command Line Syntax



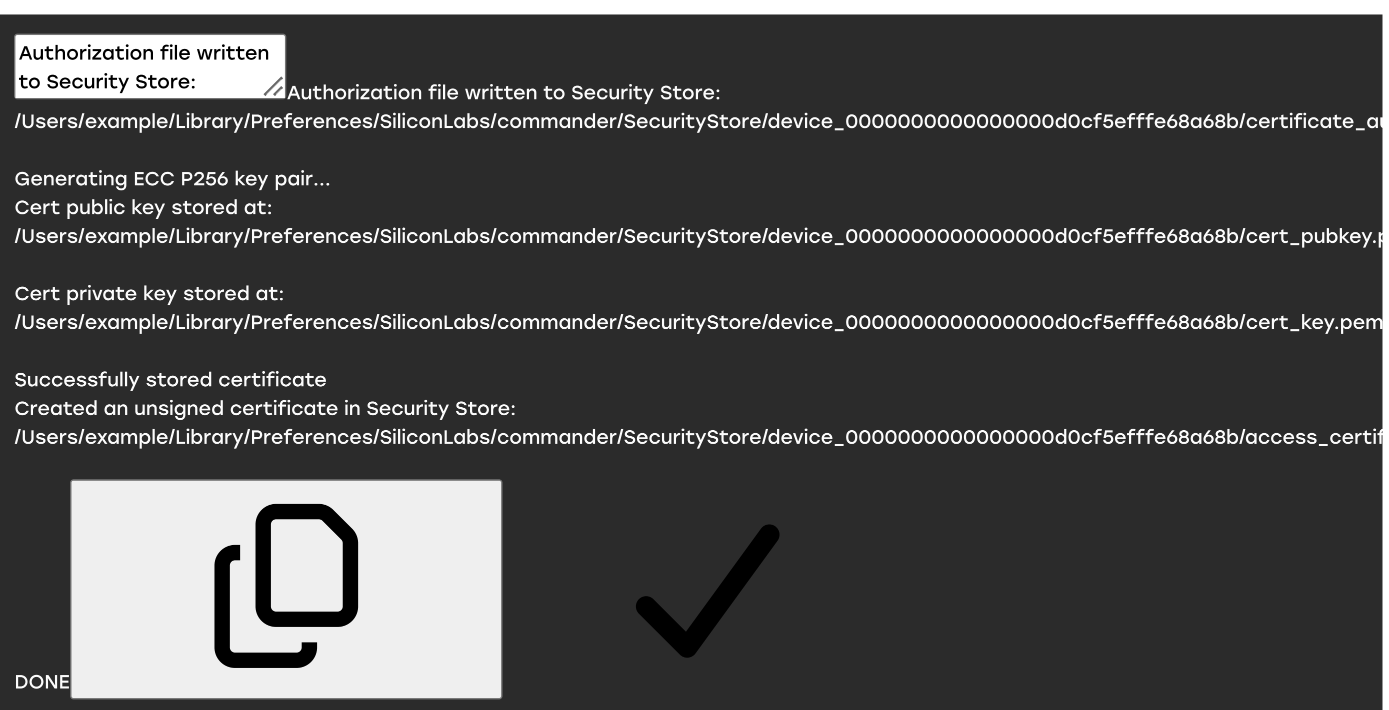

```
$ commander security gencert [--cert-pubkey <public key file>] [--authorization <auth-file>] [--command-key <private key file>][--extsign][--devserialno <serial number>] [-o <filename>] [--nostore]
```

Command Line Input Example

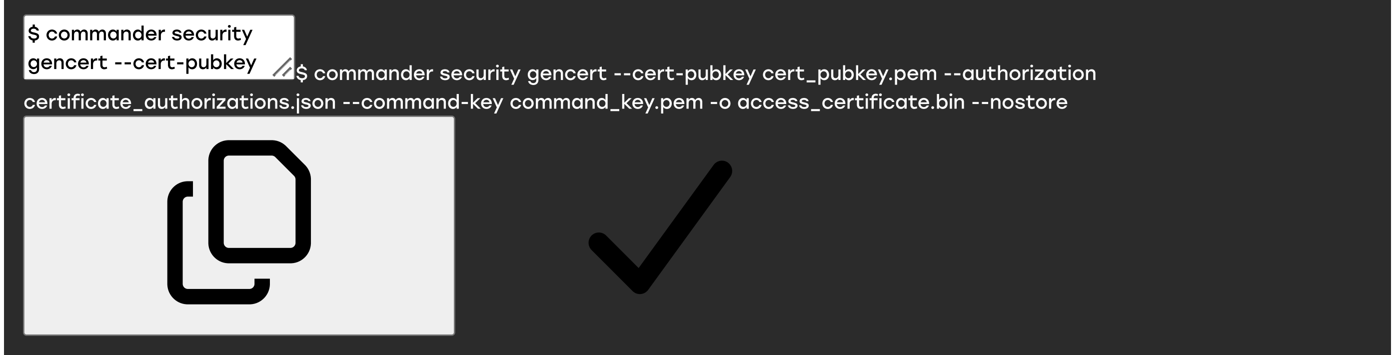
Copyright © 2026 Silicon Laboratories. All rights reserved.

This example generates an unsigned certificate, as the command private key is not provided as a command option, nor is it located in Security Store. The public certificate key is not provided either, so Simplicity commander generates a pair of certificate keys and stores them in Security Store. A default authorization file is also generated and stored in Security Store.

Command Line Output Example



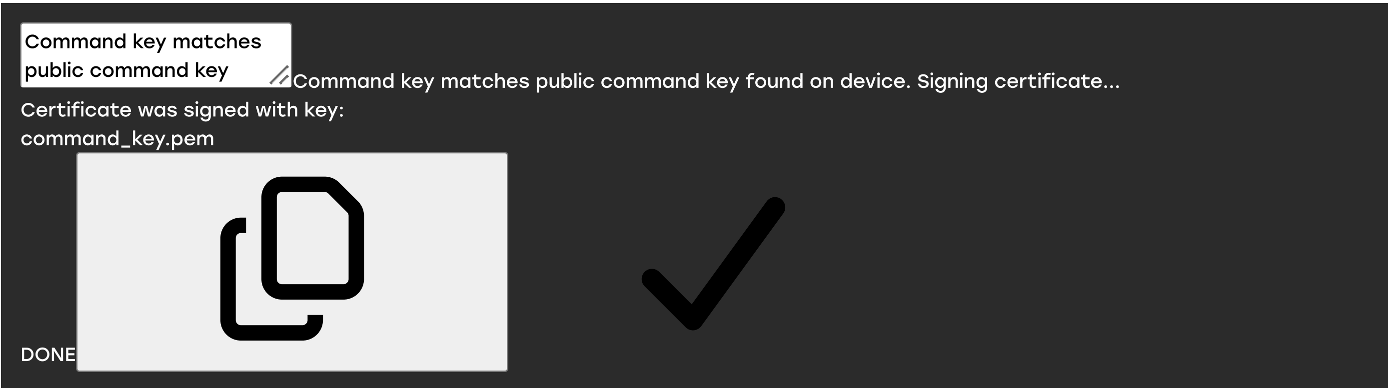

Command Line Input Example




In this example, all files needed to generate the certificate are provided as command line options. The device serial number is taken directly from the connected device. The certificate is signed with the private command

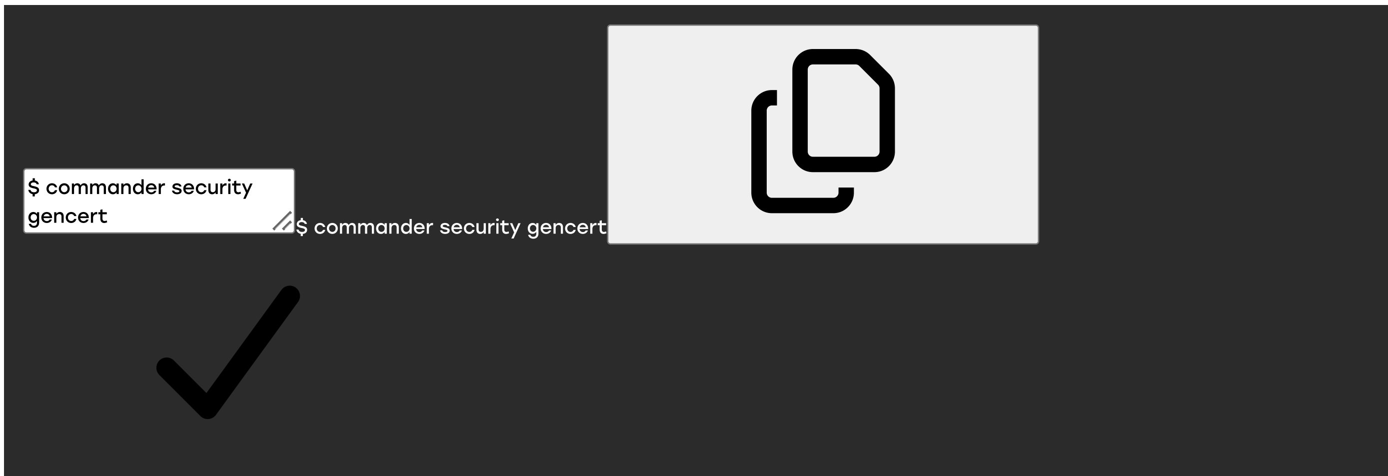
Copyright © 2026 Silicon Laboratories. All rights reserved.

Command Line Output Example



```
Command key matches public command key found on device. Signing certificate...
Certificate was signed with key:
command_key.pem
DONE
```

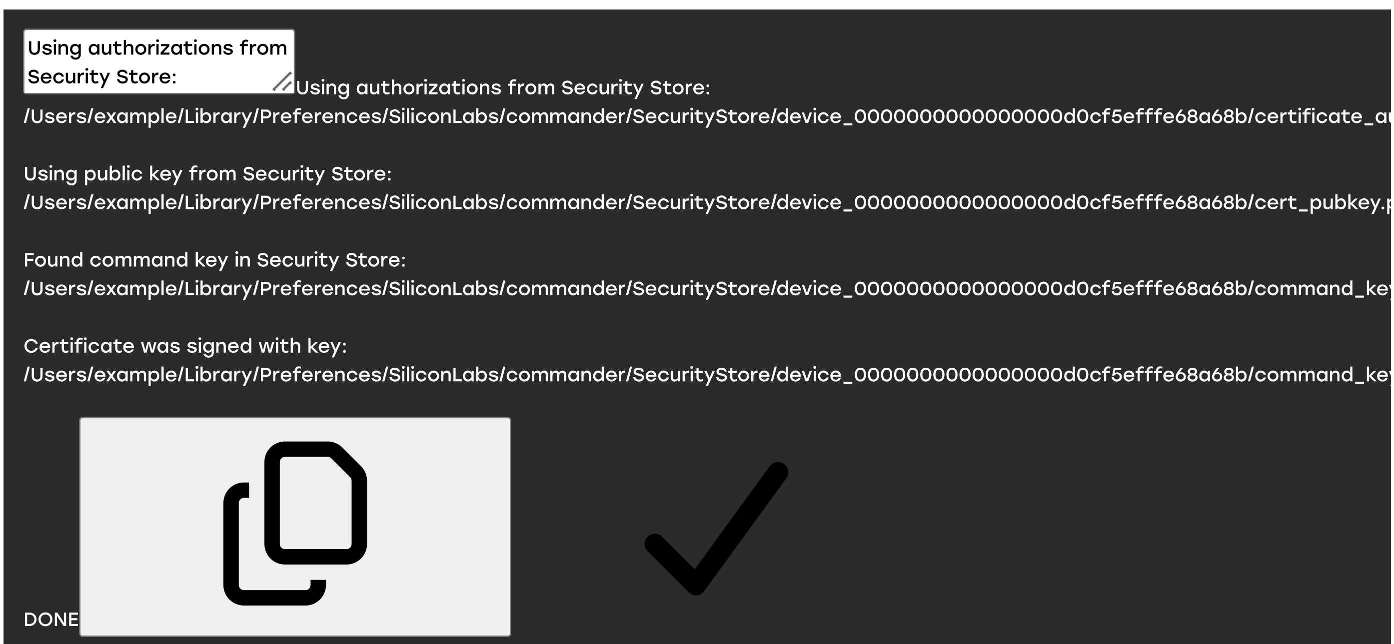
Command Line Input Example



```
$ commander security gencert
```

This example uses files already located in Security Store to generate a signed certificate. The certificate is stored in Security Store.

Command Line Output Example



```
Using authorizations from Security Store:
/Users/example/Library/Preferences/SiliconLabs/commander/SecurityStore/device_0000000000000000d0cf5efffe68a68b/certificate_a

Using public key from Security Store:
/Users/example/Library/Preferences/SiliconLabs/commander/SecurityStore/device_0000000000000000d0cf5efffe68a68b/cert_pubkey.p

Found command key in Security Store:
/Users/example/Library/Preferences/SiliconLabs/commander/SecurityStore/device_0000000000000000d0cf5efffe68a68b/command_ke

Certificate was signed with key:
/Users/example/Library/Preferences/SiliconLabs/commander/SecurityStore/device_0000000000000000d0cf5efffe68a68b/command_ke
DONE
```

## Generate Unsigned Command File

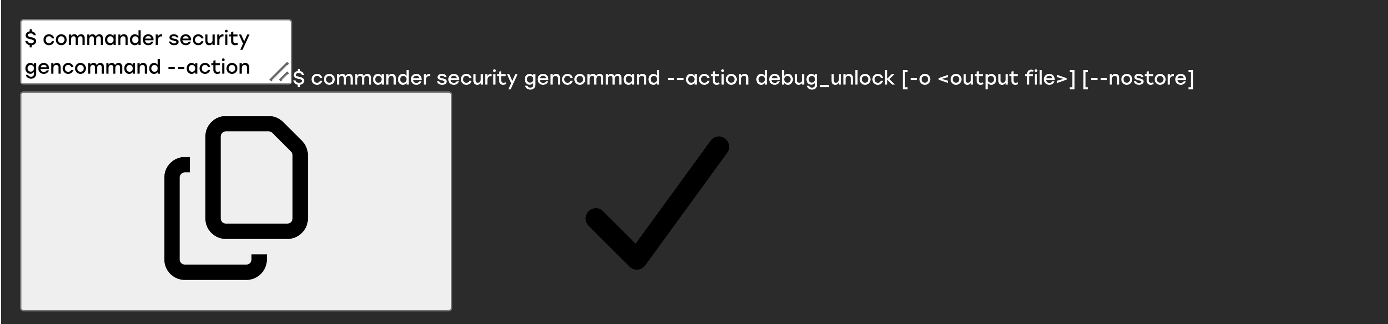
Copyright © 2026 Silicon Laboratories. All rights reserved.

The `commander security gencommand` command retrieves the security challenge from the device and stores it in a file with other data. The signature of this file using the private certificate key can be used as part of the payload to perform a secure debug unlock.

Unless the `--nostore` option is used, the unsigned command file will be stored in the Security Store.

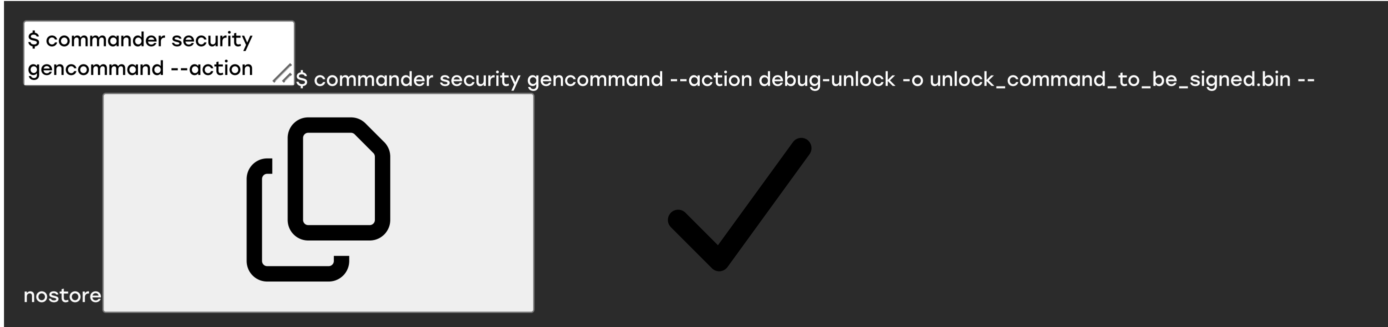
If the user has the private certificate key, Simplicity Commander automatically generates the command file and signature using the `commander security unlock` command. If the command file is signed by an external process—for example, an HSM—the command signature needs to be passed as a command line option when executing the `commander security unlock` command.

Command Line Syntax



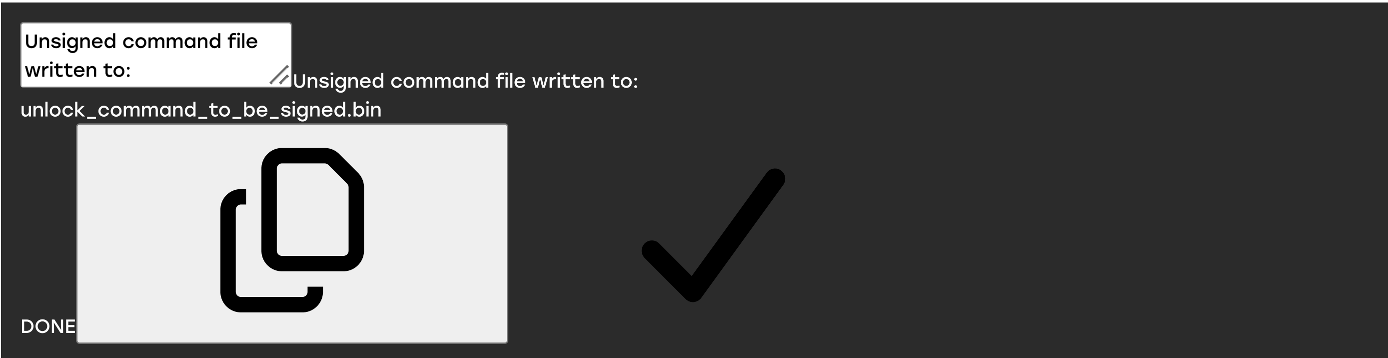
```
$ commander security gencommand --action debug_unlock [-o <output file>] [--nostore]
```

Command Line Input Example



```
$ commander security gencommand --action debug-unlock -o unlock_command_to_be_signed.bin --nostore
```

Command Line Output Example



```
Unsigned command file written to: unlock_command_to_be_signed.bin
DONE
```

## Generate Example Configuration File

This command generates a default configuration file to be used with the `security_writeconfig` command. The file is stored in Security Store unless the `--nostore` option is used.

Default Configuration File for Devices without Secure Vault

```
{
  "mcu_flags": {
    "SECURE_BOOT_ENABLE": true,
    "SECURE_BOOT_VERIFY_CERTIFICATE": false,
    "SECURE_BOOT_ANTI_ROLLBACK": true,
    "SECURE_BOOT_PAGE_LOCK_NARROW": false,
    "SECURE_BOOT_PAGE_LOCK_FULL": true
  }
}
```

Default Configuration File for Devices with Secure Vault

Copyright © 2026 Silicon Laboratories. All rights reserved.

```
{
  "mcu_flags": {
{
  "mcu_flags": {
    "SECURE_BOOT_ENABLE": true,
    "SECURE_BOOT_VERIFY_CERTIFICATE": false,
    "SECURE_BOOT_ANTI_ROLLBACK": true,
    "SECURE_BOOT_PAGE_LOCK_NARROW": false,
    "SECURE_BOOT_PAGE_LOCK_FULL": true
  },
  "tamper_levels": {
     "FILTER_COUNTER": 0,
     "WATCHDOG": 4,
     "SE_RAM_CRC": 4,
     "SE_HARDFAULT": 4,
     "SOFTWARE_ASSERTION": 4,
     "SE_CODE_AUTH": 4,
     "USER_CODE_AUTH": 4,
     "MAILBOX_AUTH": 0,
     "DCI_AUTH": 0,
     "OTP_READ": 0,
     "AUTO_CODE_AUTH": 0,
     "SELF_TEST": 4,
     "TRNG_MONITOR": 0,
     "PRS0": 0,
     "PRS1": 0,
     "PRS2": 0,
     "PRS3": 0,
     "PRS4": 0,
     "PRS5": 0,
     "PRS6": 0,
     "PRS7": 0,
     "DECOUPLE_BOD": 4,
     "TEMP_SENSOR": 1,
     "VGLITCH_FALLING": 0,
     "VGLITCH_RISING": 0,
     "SECURE_LOCK": 4,
     "SE_DEBUG": 0,
     "DGLITCH": 0,
     "SE_ICACHE": 4
  },
  "tamper_filter" : {
     "FILTER_PERIOD": 0,
     "FILTER_THRESHOLD": 0,
     "RESET_THRESHOLD": 0
  },
  "tamper_flags": {
     "DGLITCH_ALWAYS_ON": false
  }
}
```

MCU settings

- Secure Boot Enable: Enables Secure Boot on the device if set. Requires all applications running on the device to be signed.

- Secure Boot Verify Certificate: Applications running on the device must be signed using an intermediary certificate if this option is set. It is still possible to use certificates for signing even if this option is not set. For more information, see Signing an Application for Secure Boot using an Intermediary Certificate.
- Secure Boot Anti Rollback: If set, application images with a lower version than the image currently stored in flash will not run on the device.
- Secure Boot Page Lock Narrow: Flash pages validated by the Secure Boot process are locked down to prevent re-flashing by means other than through Root Code. Pages from 0 through the page where the Secure Boot signature of the application is located are locked down, not including the last page if the signature is not on a page boundary.
- Secure Boot Page Lock Full: Flash pages validated by the Secure Boot process are locked down to prevent re-flashing by means other than through Root Code. Pages from 0 through the page where the Secure Boot signature of the application is located are locked down, including the last page if the signature is not on a page boundary.

Tamper Levels

The different tamper sources are listed under tamper levels. The default configuration is an absolute minimum. The Root Code will never set tamper levels to a lower setting than the default configuration. The tamper levels are listed in the following table.

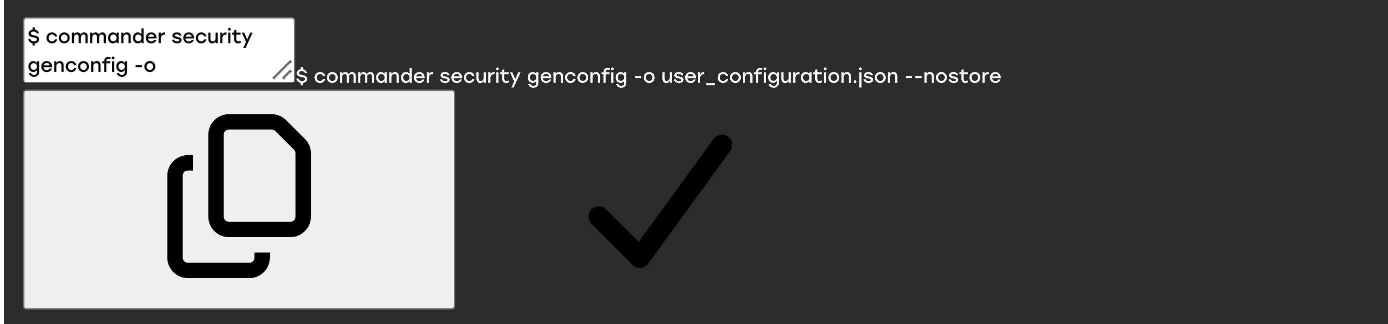
Table: Tamper Levels

| Tamper Level | Description |
|---|---|
| 0 | No action taken |
| 1 | Generate SE interrupt |
| 2 | Increment filter counter |
| 4 | System Reset |
| 5 | Reserved |
| 6 | Reserved |
| 7 | Erase OTP (Makes the device unrecoverable; it will never boot again.) |

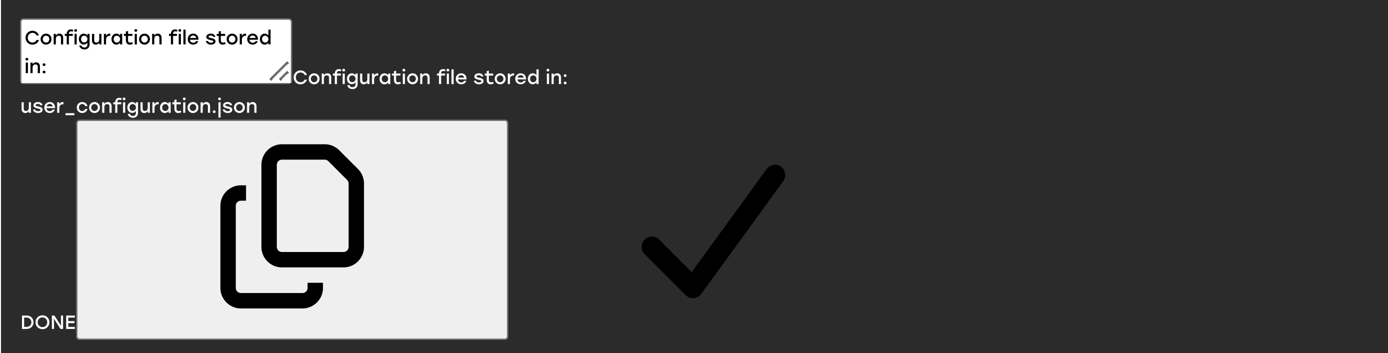
Command Line Syntax

Command Line Input Example

Copyright © 2026 Silicon Laboratories. All rights reserved.




Command Line Output Example




## Write User Configuration

IMPORTANT: This is a one-time command. It cannot be run more than once.

The `commander security writeconfig` command sets the configurations determined in the configuration file in the Root Code.

Secure Boot is enabled through this command. Before Secure Boot is enabled, you must write the public sign key to the device. For more information on writing keys to the device, see Write Public Key to Device. In addition, a configuration file must be generated and the Secure Boot Enabled flag must be set to true. If no configuration file is provided, a default configuration will be generated.

In Simplicity Commander version 1.9, tamper configuration is supported on devices with Secure Vault. The tamper configuration determines the response from the Secure Engine in the occurrence of a tamper event. For more information about the configuration file and tamper configuration, see Generate Example Configuration File.

For more information about Secure Boot, see *AN1218: Series 2 Secure Boot with RTSL*.

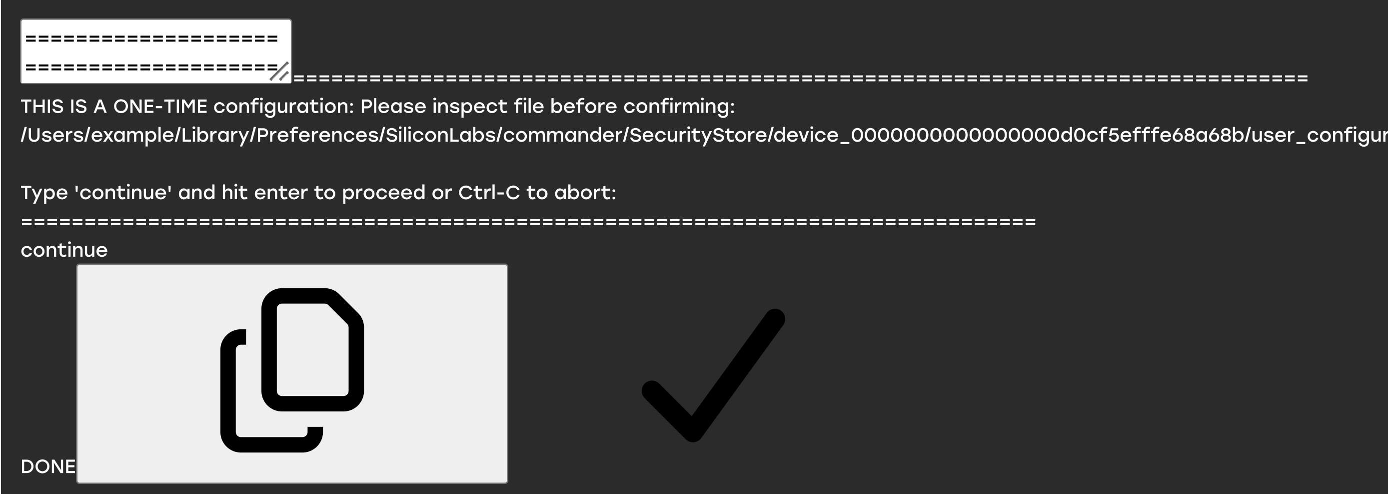
For more information about tamper events, see Disable Tamper.

Command Line Syntax

Command Line Input Example

Copyright © 2026 Silicon Laboratories. All rights reserved.

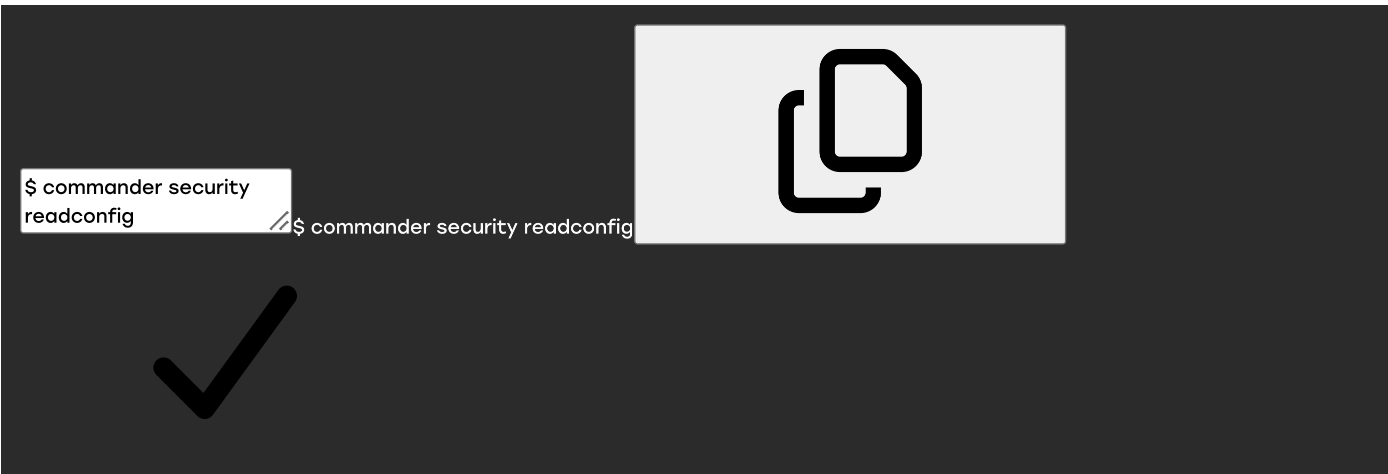
Command Line Output Example




## Read User Configuration

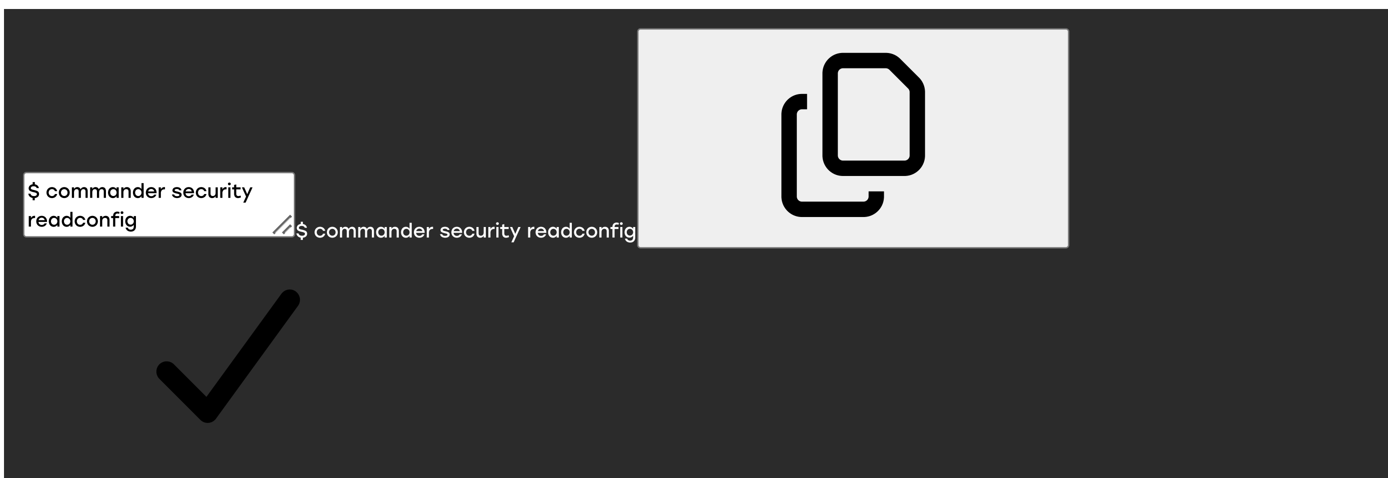
This command returns the One-Time Programmable (OTP) setting from the device. If the device has not been configured with the Write User Configuration command, no OTP settings are available to read.

Command Line Syntax




Command Line Input Example

Copyright © 2026 Silicon Laboratories. All rights reserved.




Command Line Output Example

Copyright © 2026 Silicon Laboratories. All rights reserved.

```
MCU Flags
Secure Boot                 : MCU Flags
Secure Boot                 : Enabled
Secure Boot Verify Certificate : Disabled
Secure Boot Anti Rollback      : Enabled
Secure Boot Page Lock Narrow   : Disabled
Secure Boot Page Lock Full     : Enabled
Tamper Levels
FILTER_COUNTER     : 0
WATCHDOG           : 4
SE_RAM_CRC         : 4
SE_HARDFAULT       : 4
SOFTWARE_ASSERTION : 4
SE_CODE_AUTH       : 4
USER_CODE_AUTH     : 4
MAILBOX_AUTH       : 0
DCI_AUTH           : 0
OTP_READ           : 0
AUTO_CODE_AUTH     : 0
SELF_TEST          : 4
TRNG_MONITOR       : 0
PRS0               : 0
PRS1               : 0
PRS2               : 0
PRS3               : 0
PRS4               : 0
PRS5               : 0
PRS6               : 0
PRS7               : 0
DECOUPLE_BOD       : 4
TEMP_SENSOR        : 1
VGLITCH_FALLING    : 0
VGLITCH_RISING     : 0
SECURE_LOCK        : 4
SE_DEBUG           : 0
DGLITCH            : 0
SE_ICACHE          : 4
Tamper Filter
Filter Period   : 0
Filter Treshold : 0
Reset Treshold  : 0
Tamper Flags
Digital Glitch Detector Always On: Disabled
DONE
```

## Close Code Region

This command closes (locks) a code region on the device. It is only available on Series 3 devices.

The region is selected by index in the range 0–7. You can optionally set a 32-bit code region version when locking by using `--codeversion` ; this is applied when the region is closed.

Closing a code region consumes one OTP bit. On SiMG301 devices, the SE requires that code region 0 be closed when secure boot is enabled. The bootloader requires that an application region be closed before an OTA upgrade can be applied.

Copyright © 2026 Silicon Laboratories. All rights reserved.

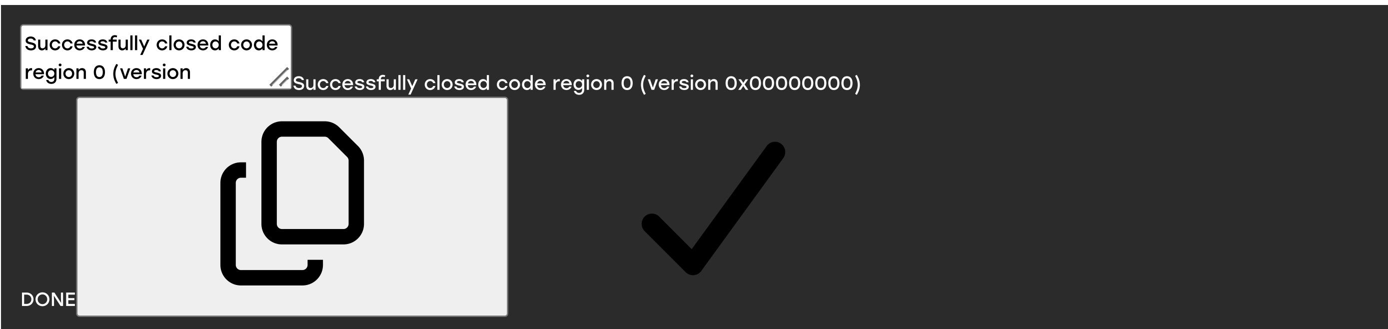
Command Line Syntax

```
$ commander security closeregion <index> [--codeversion <version (32 bits)>]
```

Command Line Input Example

```
$ commander security closeregion 0
```

Command Line Output Example



```
Successfully closed code region 0 (version 0x00000000)
DONE
```

## Read Code Region Configuration

This command reads the code region configuration from the device. It is only available on Series 3 devices.

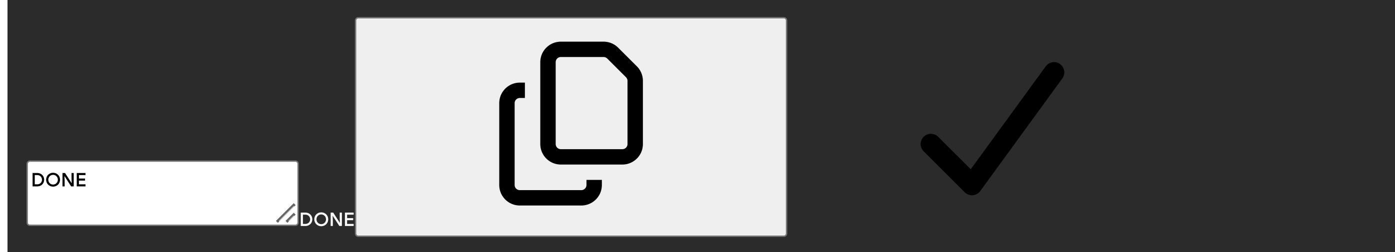
Use `--outfile` to write the configuration to a file. The output is in YAML format and can be used as a starting point for Write Code Region Configuration. Run `readregionconfig` with `--outfile`, edit the file as needed, and then pass it to `writeregionconfig`.

Command Line Syntax

Command Line Input Example

When you specify the `--outfile` option, the command writes the YAML output to the specified path. If you omit the `--outfile` option, the command prints the same YAML output to the terminal. You can edit the file and pass it to `writeregionconfig` (see Write Code Region Configuration).

Command Line Output Example




## Write Code Region Configuration

This command writes a code region configuration from a YAML file to the device. It is available only on Series 3 devices.

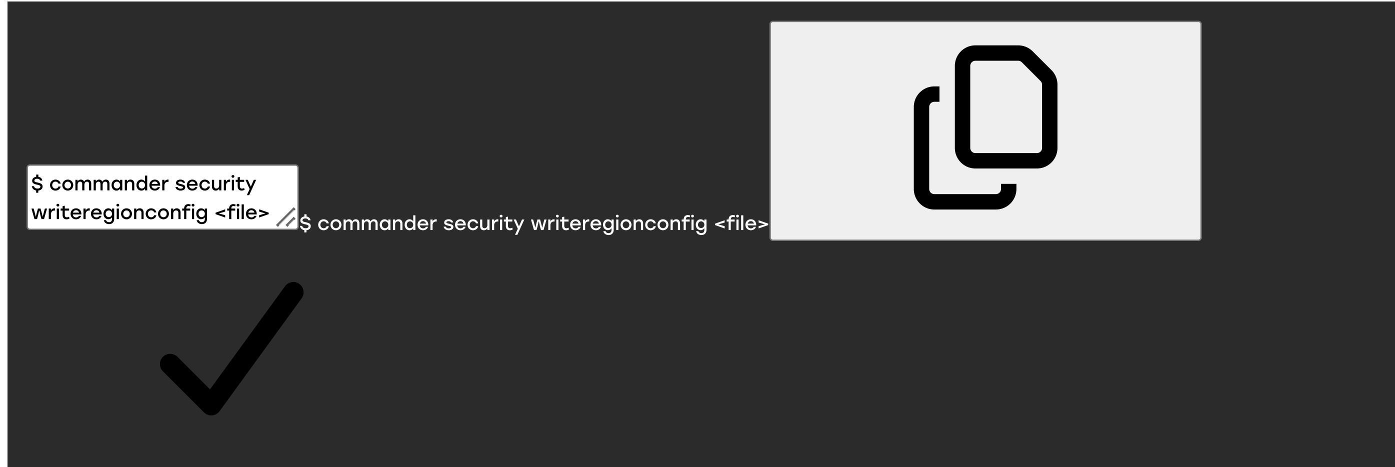
The file must contain the region configuration. To generate a template, use Read Code Region Configuration with the `--outfile` option.

If any code regions are already closed, you must erase them before changing the configuration. Otherwise, the write operation fails with an error indicating that closed regions must be erased first.

The device needs to be reset ( `commander device reset` ) for the data region to be resized accordingly.

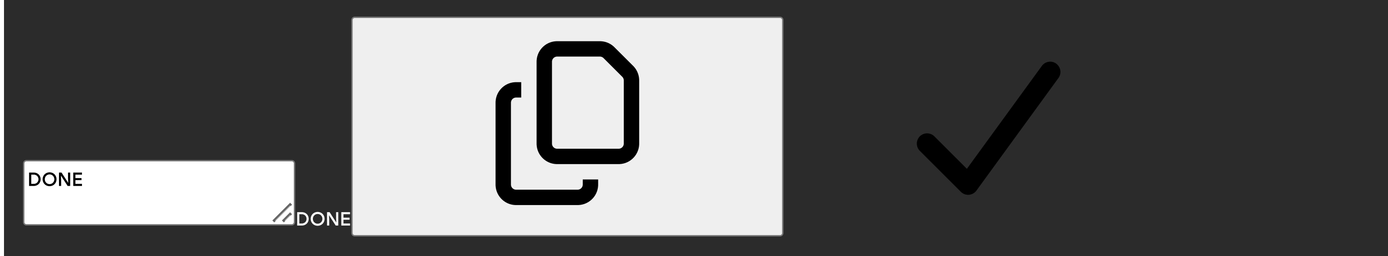
Command Line Syntax

Copyright © 2026 Silicon Laboratories. All rights reserved.




Command Line Input Example

Command Line Output Example

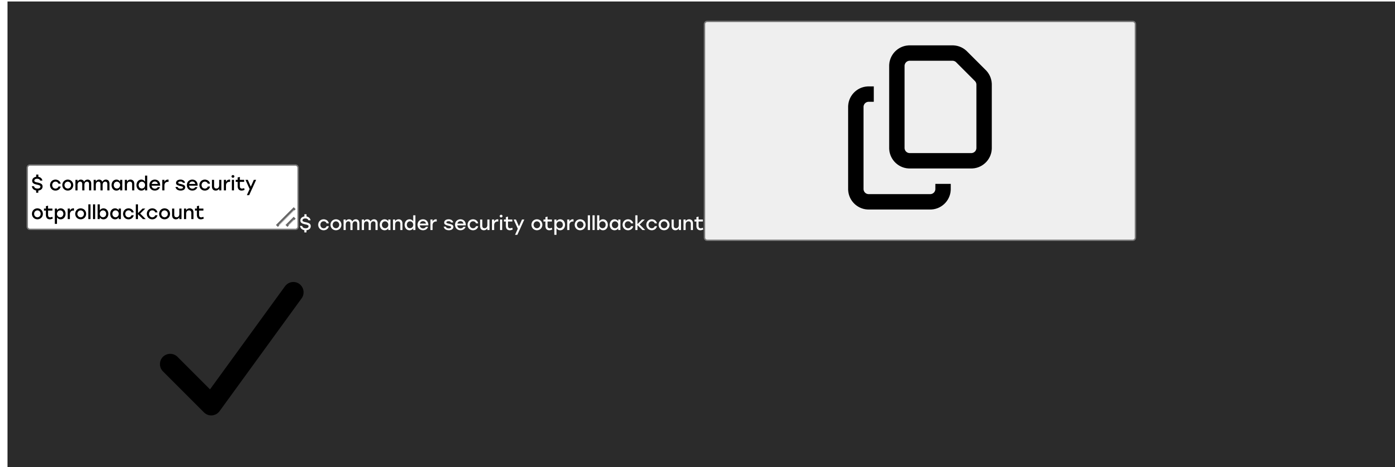



## OTP Rollback Count

If any code regions are already closed, you must erase them before changing the configuration. Otherwise, the write operation fails with an error indicating that closed regions must be erased first.

Command Line Syntax

Copyright © 2026 Silicon Laboratories. All rights reserved.

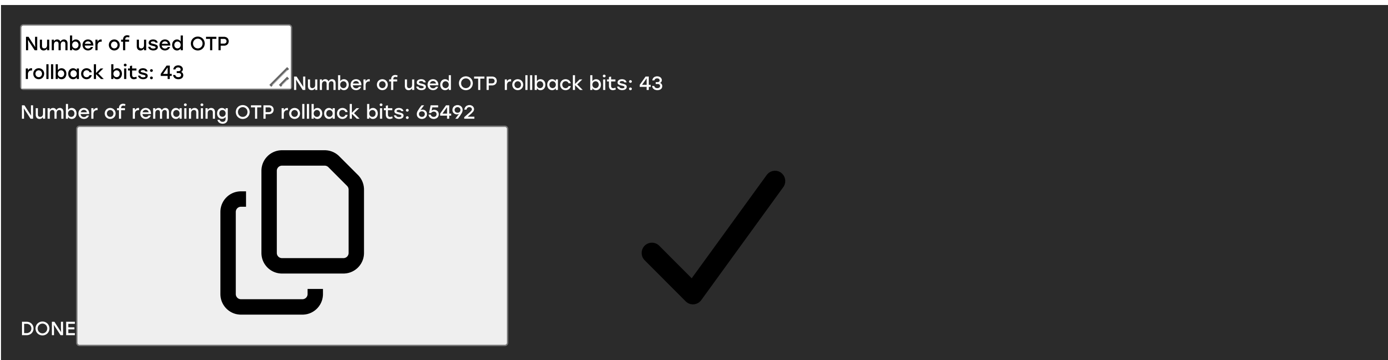


```
$ commander security otprollbackcount
```

Command Line Input Example

```
$ commander security otprollbackcount
```

Command Line Output Example



```
Number of used OTP rollback bits: 43
Number of remaining OTP rollback bits: 65492
DONE
```

If all rollback bits have been consumed, Commander reports that all OTP rollback bits are consumed instead of the counts above.

## Provision Secure Engine Firmware (External Flash)

This command provisions an external flash device with Secure Engine (SE) firmware. It is typically run during production when building boards that include an external flash device. The command can be used only once per device. After provisioning, the device is associated with the external flash device. This command applies only to Series 3 devices with external flash.

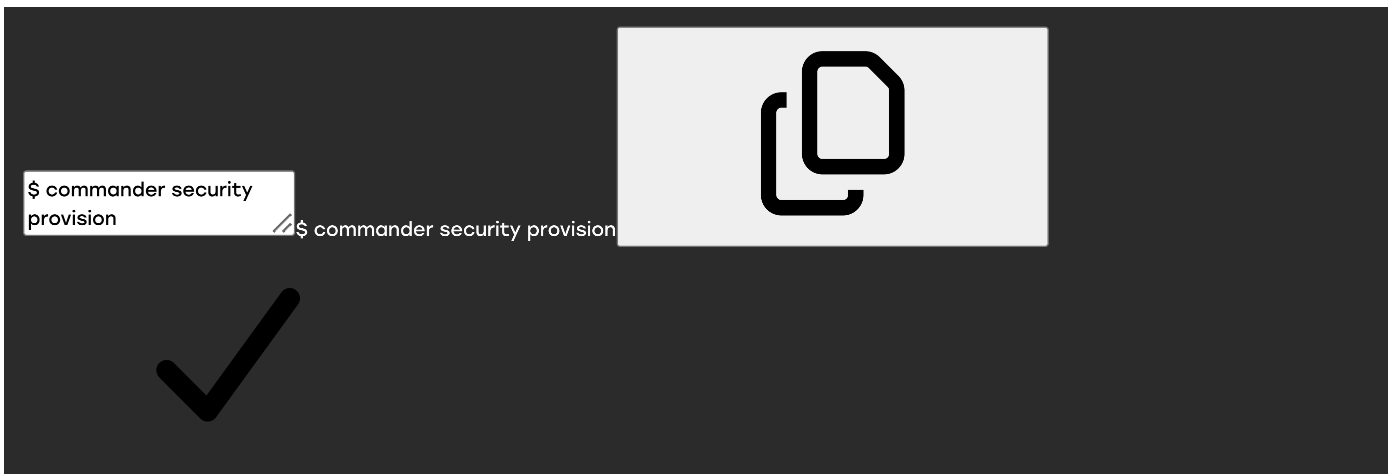
If you omit the `--sefw` option, Commander uses the latest bundled SE firmware image in seuv2 format. To install a specific image file, specify the `--sefw` option.

Copyright © 2026 Silicon Laboratories. All rights reserved.

Command Line Syntax

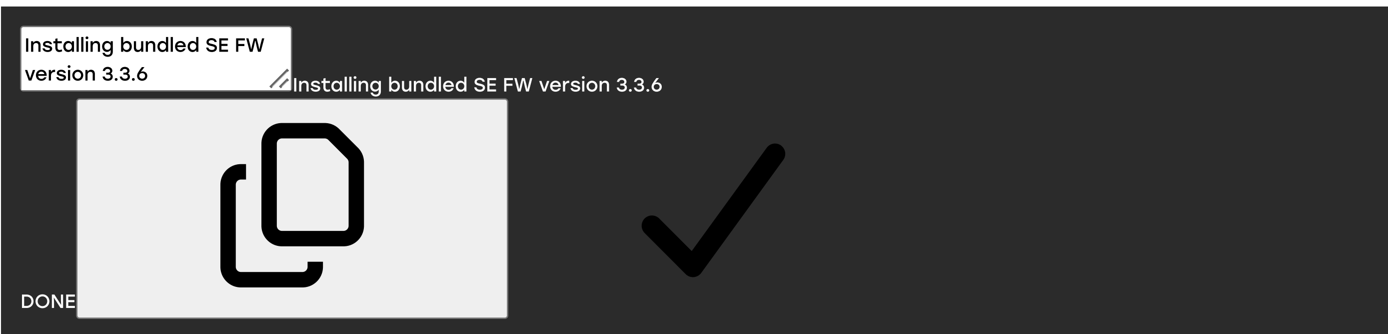
```
$ commander security provision [--sefw <seuv2 file>]
```

Command Line Input Example



```
$ commander security provision
```

Command Line Output Example



```
Installing bundled SE FW version 3.3.6
DONE
```

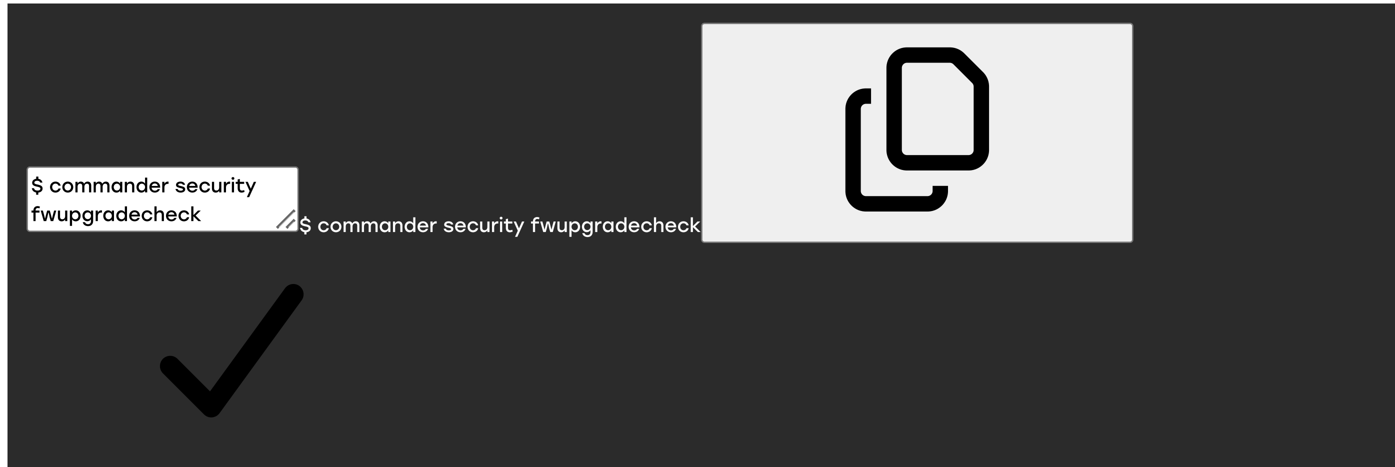
## Secure Engine Firmware Upgrade Check

This command checks whether a bundled SE firmware upgrade is available for the connected device. It is available only on Series 3 devices.

You may run this command before Secure Engine Firmware Upgrade to confirm that Commander provides a newer SE firmware image than the one currently installed on the device.

Command Line Syntax

Copyright © 2026 Silicon Laboratories. All rights reserved.



```
$ commander security fwupgradecheck
```

Command Line Input Example

```
$ commander security fwupgradecheck
```

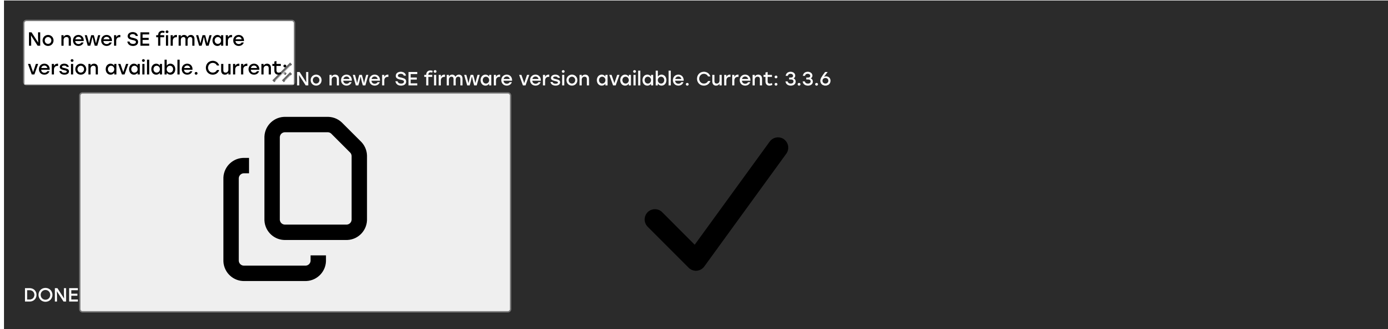
Command Line Output Example

If a newer bundled image exists:

```
New SE firmware version available: 3.3.6 (current: 3.2.0)
DONE
```

If the device already has the latest (or newer) SE firmware:

Copyright © 2026 Silicon Laboratories. All rights reserved.



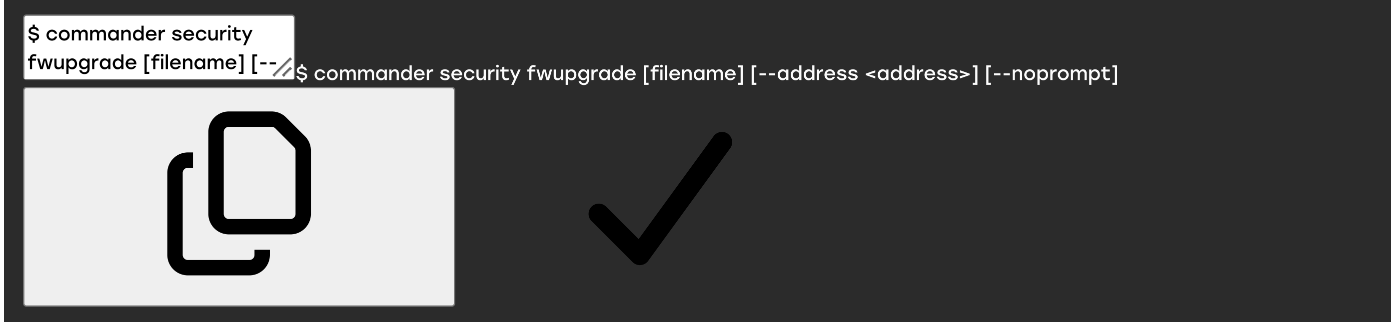

## Secure Engine Firmware Upgrade

This command upgrades the SE firmware on the device. It is available only on Series 3 devices.

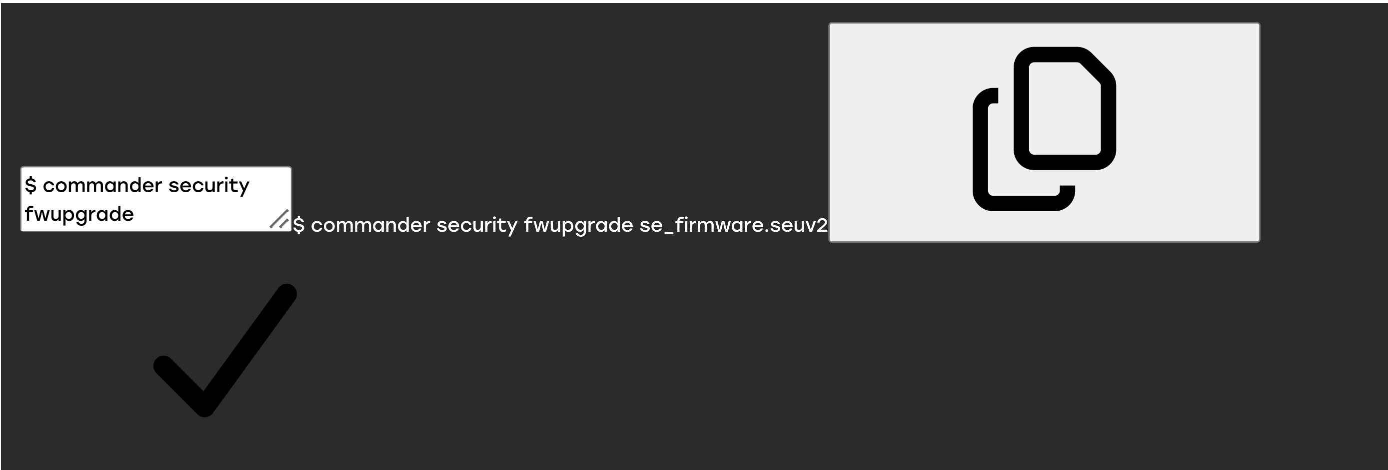
The firmware package must be in `.seuv2` format, as provided by Silicon Labs. If you omit filename, Commander applies the bundled SE firmware when it is newer than the version currently installed on the device. This behavior follows similar selection logic as Secure Engine Firmware Upgrade Check). To force installation of a specific image, pass the path as filename. SE firmware cannot be downgraded.

- `--address` specifies where the upgrade image is placed in memory before the upgrade runs. The default is the start of flash memory.
- `--noprompt` skips the interactive confirmation prompt. Use this option only when appropriate for your workflow, typically when installing from RAM and accepting the associated risks.

Command Line Syntax

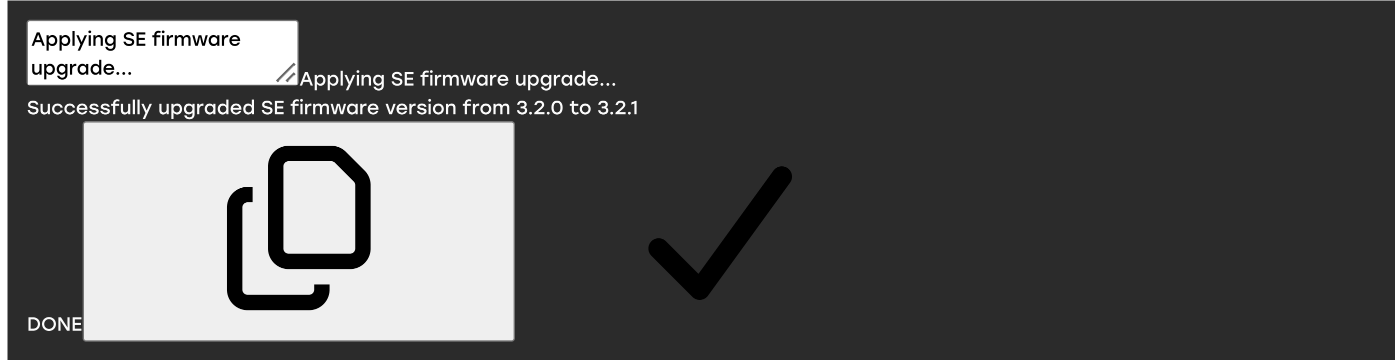



Command Line Input Example




Command Line Output Example

Copyright © 2026 Silicon Laboratories. All rights reserved.



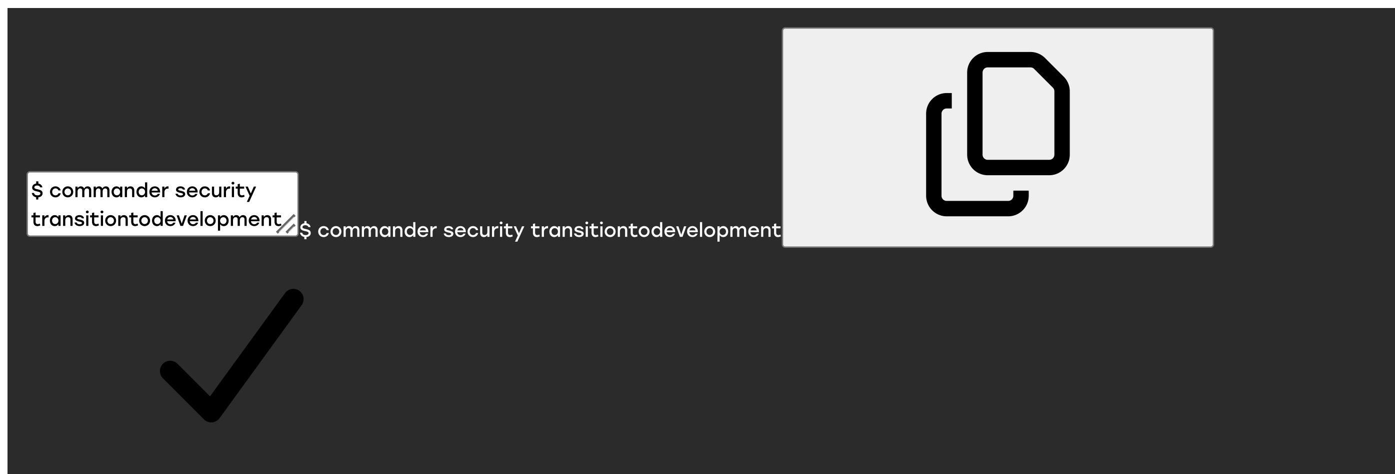

## Transition to Development Mode

This command permanently moves the device into development mode. It is available only on Series 3 devices.

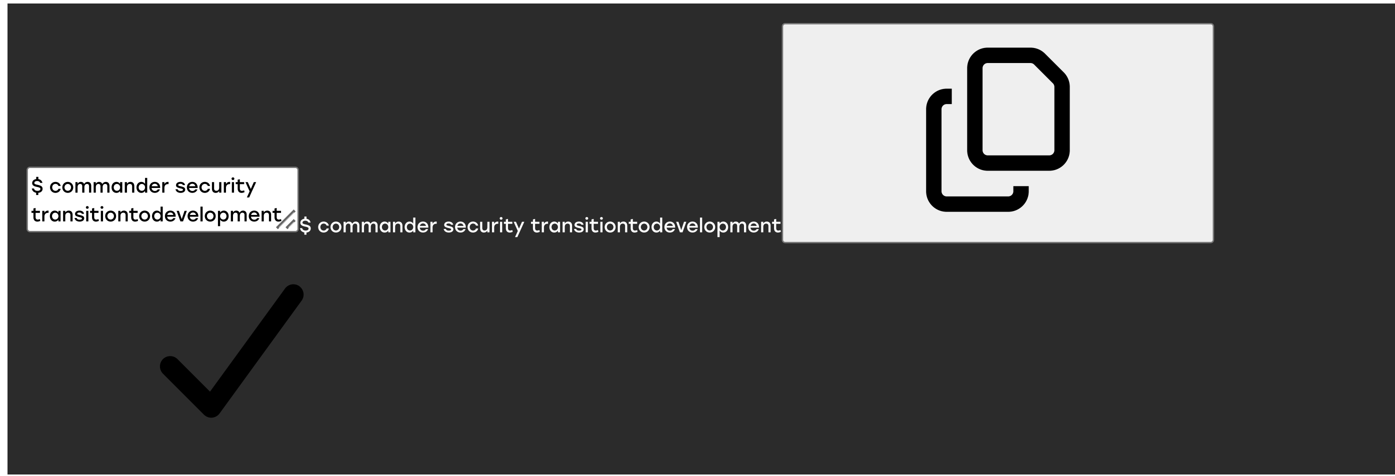
Development mode can reduce consumption of OTP rollback bits during repeated flash and erase cycles in a lab. This mode is not secure and must not be used for production devices.

After the transition, the device permanently reuses the same initialization vector (IV) for flash encryption. This behavior is not secure for products that require strong confidentiality of flash contents.

Command Line Syntax

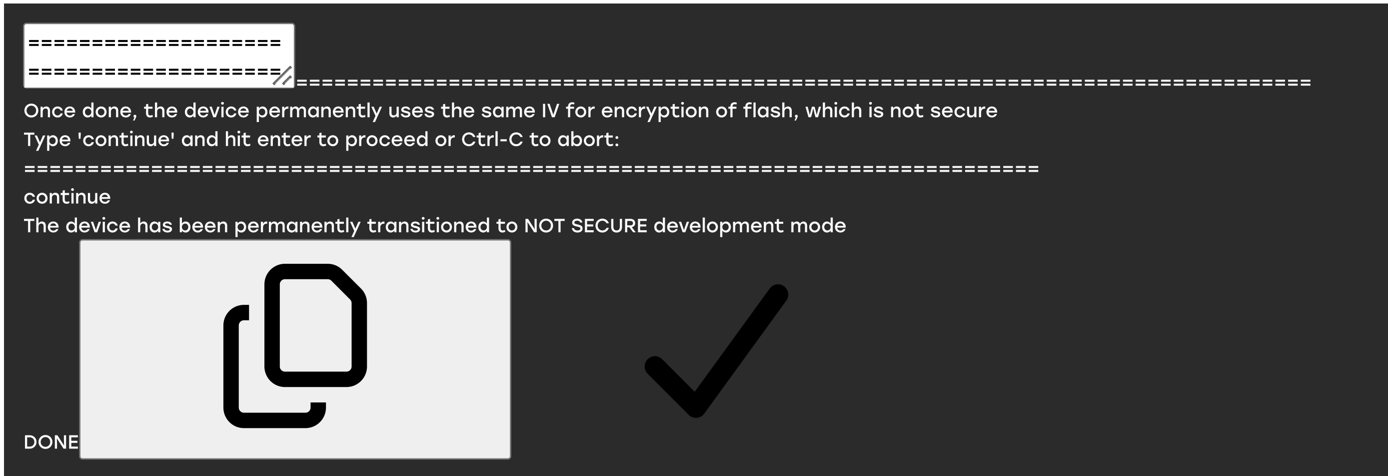



Command Line Input Example




Command Line Output Example

Copyright © 2026 Silicon Laboratories. All rights reserved.

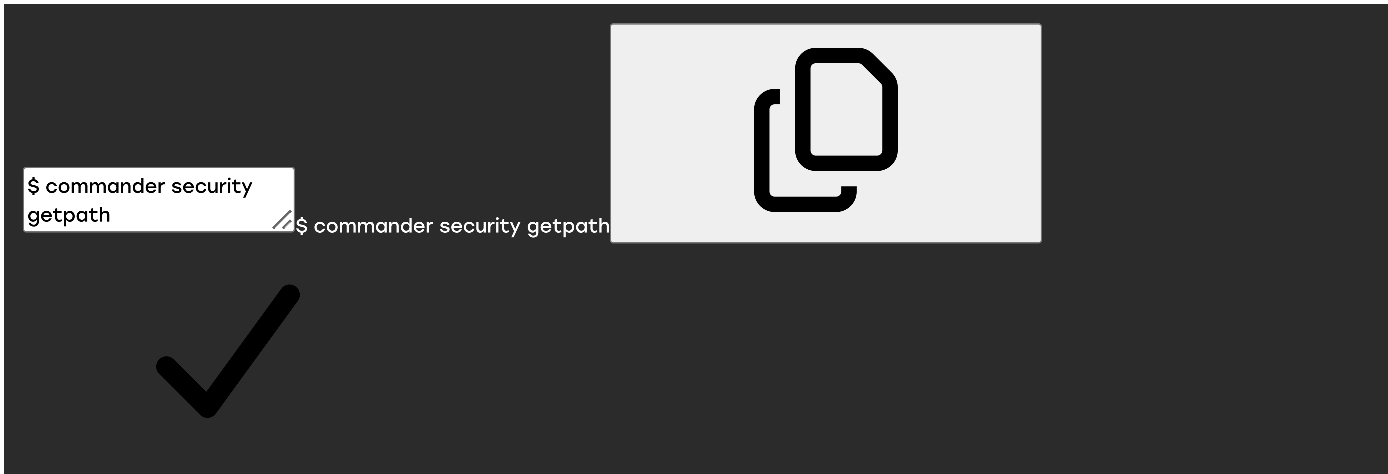



## Get Security Store Path

Get the path to the security store. If a device is connected or the `--deviceserialno` option is provided, the device specific path is returned. Otherwise, the path to Security Store is returned.

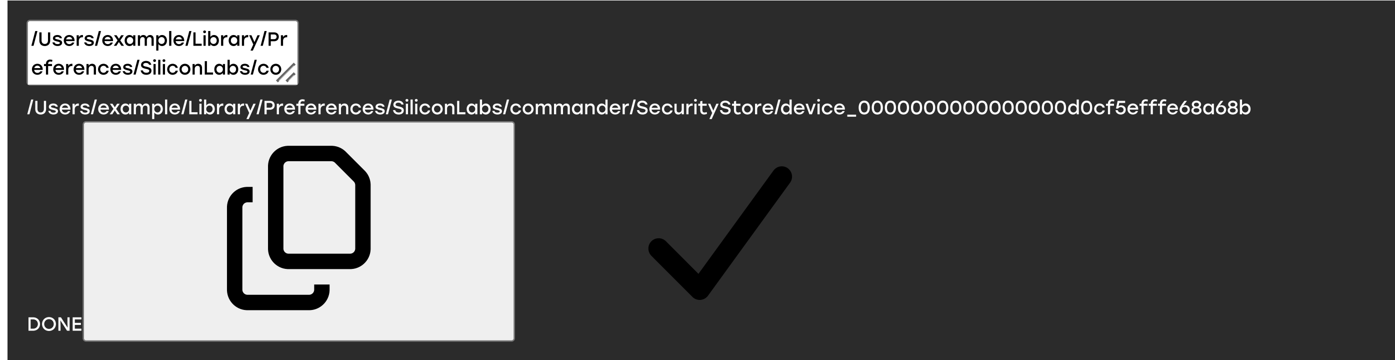
Command Line Syntax

Command Line Input Example




Command Line Output Example

Copyright © 2026 Silicon Laboratories. All rights reserved.

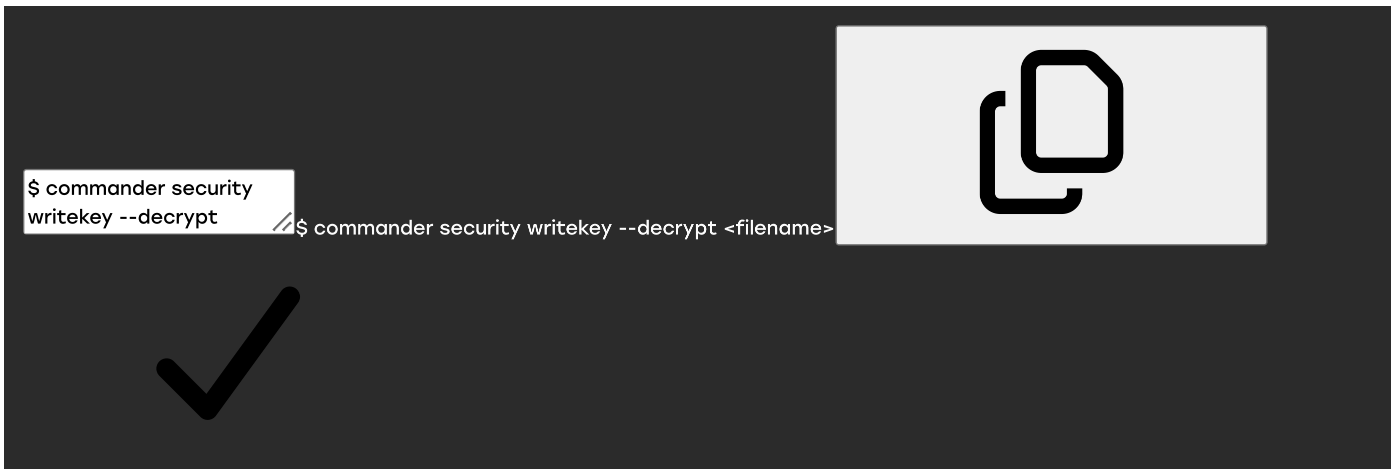



## Write AES Decryption Key

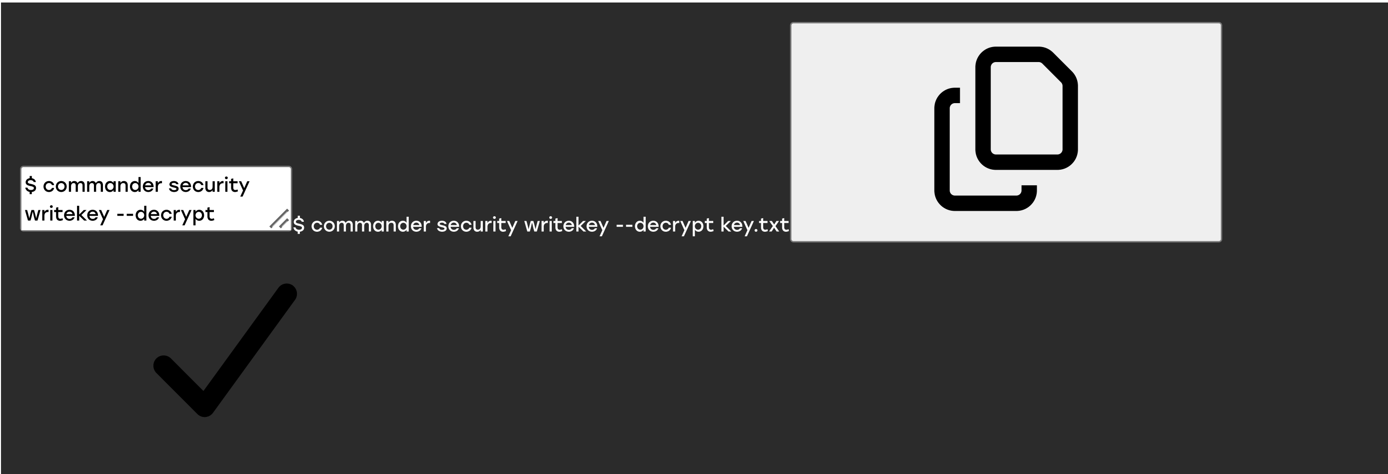
Important: This is a one-time command. It cannot be run more than once per device.

The symmetric 128-bit AES key is used to decrypt GBL files. This key is also known as the MFG_BOOTLOAD_AES_KEY. All encrypted images on this device must be encrypted with the same AES key.

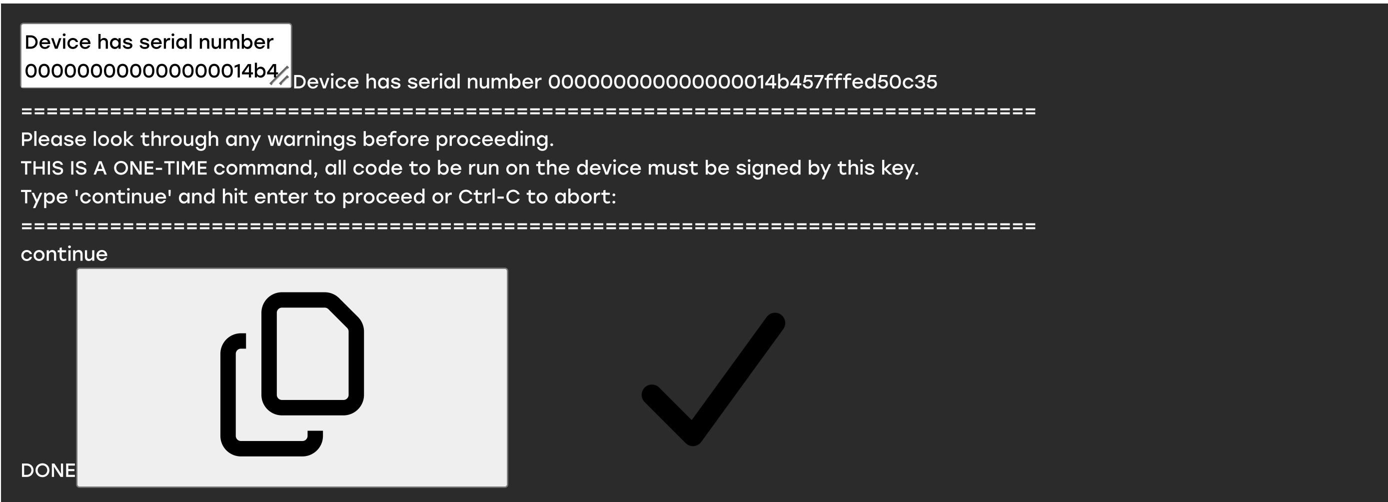
Command Line Syntax




Command Line Input Example




Command Line Output Example



```
Device has serial number 000000000000000014b457fffed50c35
================================================================================
Please look through any warnings before proceeding.
THIS IS A ONE-TIME command, all code to be run on the device must be signed by this key.
Type 'continue' and hit enter to proceed or Ctrl-C to abort:
================================================================================
continue
DONE
```

## Read Device Certificates

This command reads out a X509 certificate from the device. The available certificates are:

- batch: same for each manufacturing batch
- SE: unique per device
- MCU: unique per device

The certificates form a root-of-trust certificate chain up to the `Silicon Labs Root Certificate` issued by Silicon Labs. The `SE` and `MCU Certificates` are issued by a `Batch Certificate`. The `Batch Certificate` is issued by a `Factory Certificate`, and the `Factory Certificate` is issued by the `Silicon Labs Root Certificate`.

Key information about the certificate is printed to the command line if no outfile is given. The certificate may be read out in entirety by providing the `outfile` argument. The available encodings are `pem` and `der`.

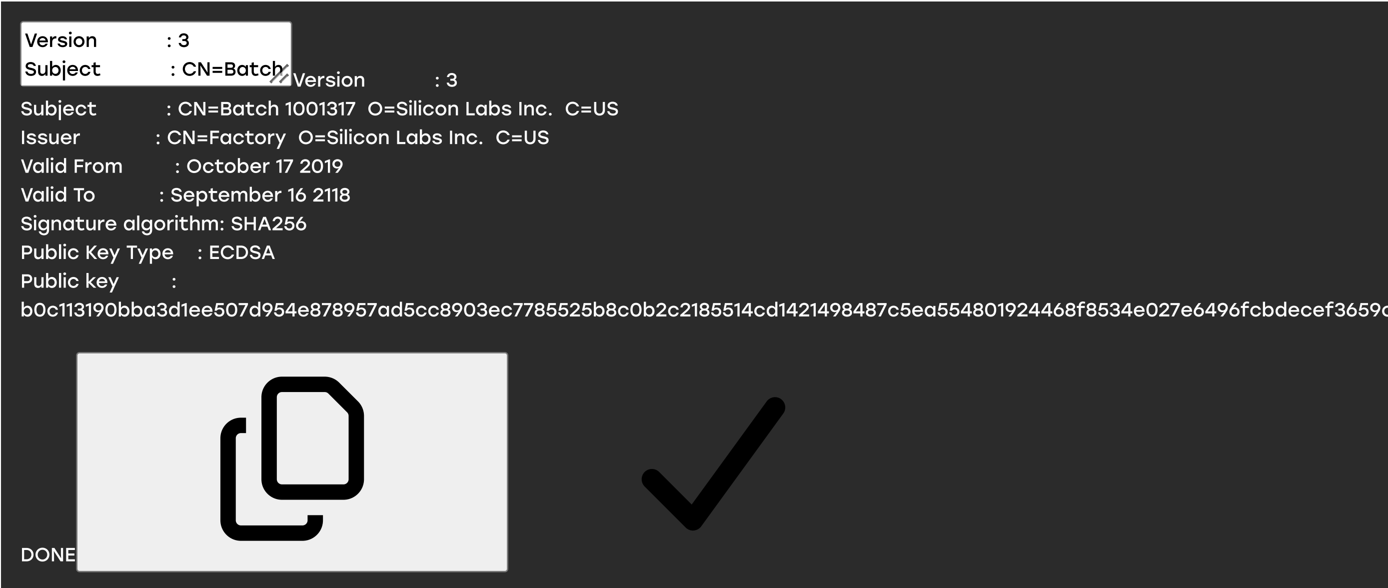
Command Line Syntax

```
$ commander security readcert <cert type> [--outfile <filename>]
```

Command Line Input Example

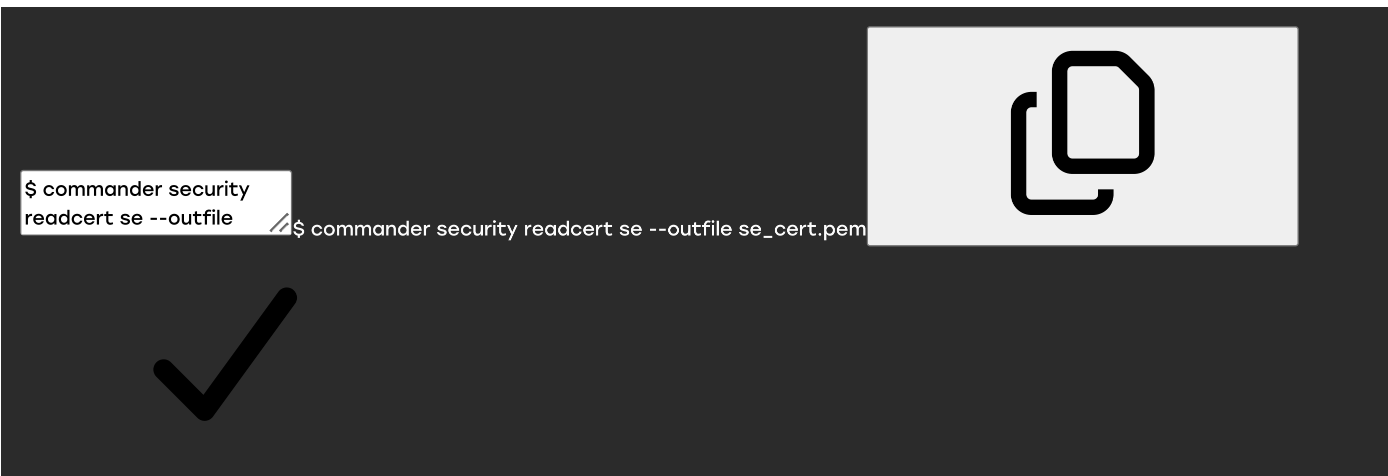
```
$ commander security readcert batch
```

Command Line Output Example



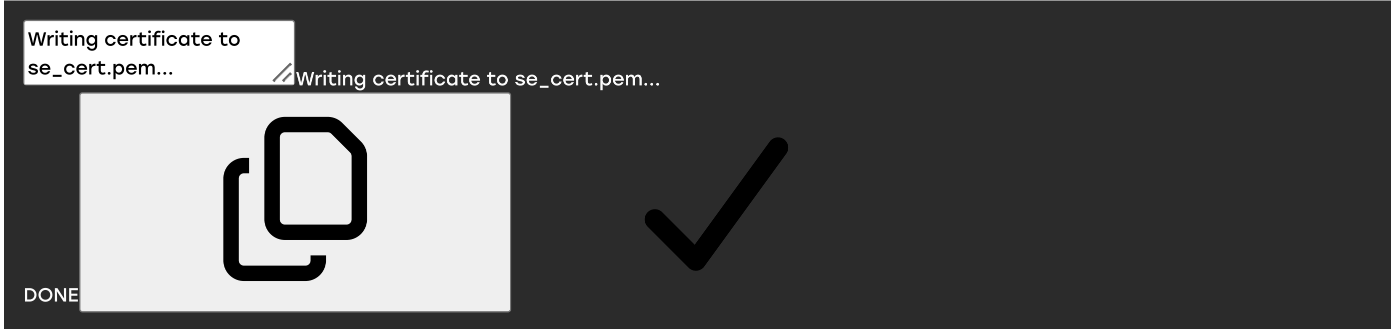
```
Version          : 3
Subject          : CN=Batch 1001317  O=Silicon Labs Inc.  C=US
Issuer           : CN=Factory  O=Silicon Labs Inc.  C=US
Valid From       : October 17 2019
Valid To         : September 16 2118
Signature algorithm: SHA256
Public Key Type    : ECDSA
Public key         :
b0c113190bba3d1ee507d954e878957ad5cc8903ec7785525b8c0b2c2185514cd1421498487c5ea554801924468f8534e027e6496fcbdecef3659c
DONE
```

Command Line Input Example



```
$ commander security readcert se --outfile se_cert.pem
```

Command Line Output Example




## Vault Device Attestation

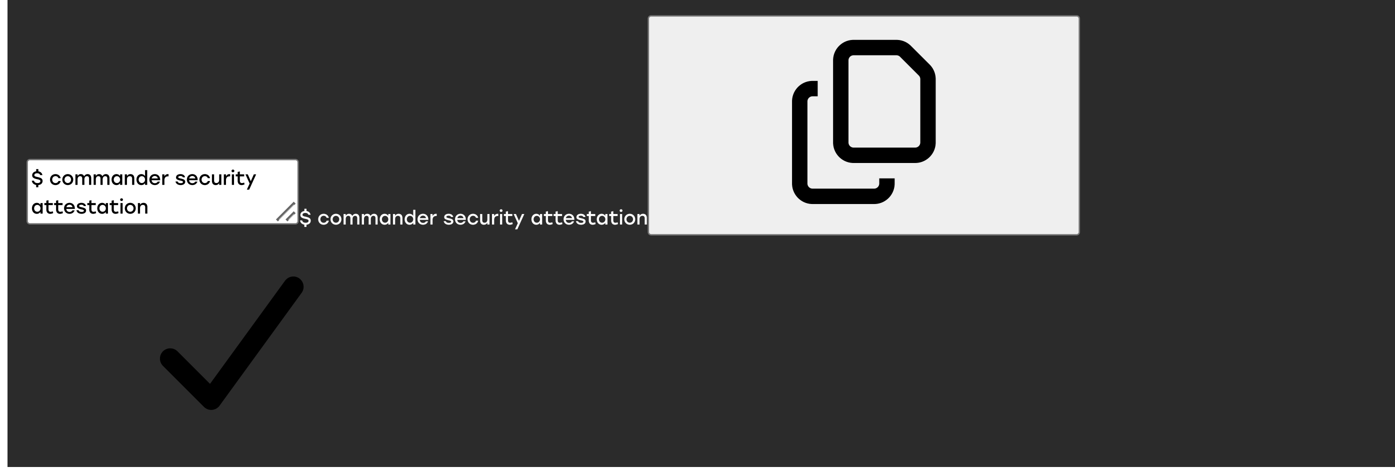
Attestation of a device is used to cryptographically prove to a remote party that they are the system they say they are, and ensure that the device they are talking to is the same device as the one that got produced in the factory.

The attestation process starts with authenticating the certificate chain up to the Silicon Labs Root certificate. For more information on certificates, see Read Device Certificates.

The attestation token is printed to the command line. The token consists of multiple claims as listed in the following table.

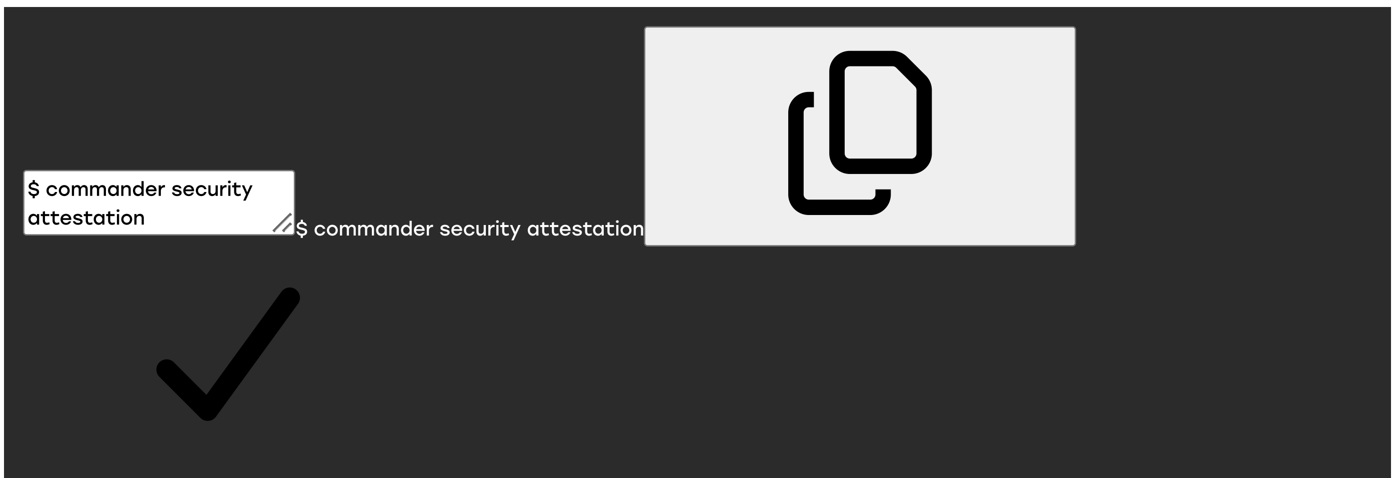
| Claim ID | Claim friendly name | Present in token | Content |
|---|---|---|---|
| -75000 | ARM PSA Profile ID | Always | ASCII 'SILABS_1' |
| -75008 | ARM PSA nonce | Always | Copy of the nonce supplied as input to the token generation command. |
| -75009 | ARM PSA/IETF EAT UEID | Always | The device's EUI-64 pre-pended with 0x06 and zeroes. |
| -76000 | SE status | Always | Current SE status |
| -76001 | OTP configuration | Always when provisioned | User configuration |
| -76002 | MCU Sign key | Always when provisioned | Public sign key |
| -76003 | MCU Command key | Always when provisioned | Public command key |
| -76004 | Current applied tamper settings | Always | Currently applied tamper level per tamper signal (one nibble per tamper signal). |

Finally, the signature of the attestation token is verified as shown in the following examples.




Command Line Input Example

Copyright © 2026 Silicon Laboratories. All rights reserved.



```
$ commander security attestation
```

Command Line Output Example

```
Certificate chain successfully validated up to Silicon Labs device root certificate.
-75008 ARM PSA nonce                 : 1799c9296ac44a854b74fe50dc6f1546a5c1e17de73584afcc478739161db7d0
-75000 ARM PSA Profile ID            : SILABS_1
-75009 ARM PSA/IETF EAT UEID         : 0614b457fffe0f7789
-76000 SE status                     : 0000000100000000000000000000000000000002000010202fffffff00000002ffffffff
-76002 MCU sign key                  :
fb2470314c0710f5a72e89a30d2af607770187568f80cffa7fc6516f61e0dc258a8606fe664a097eb94d3ea29e1b87262babdb969842da31512bdc7

-76003 MCU command key               :
a218c9615321567527e94ac1f01230604e231f1eabe699fb1d751af3e28d00feaa3dd823540a2452baa40dfb3475d3bb786b41e7880881b5a5427e7

-76004 Current applied tamper settings : 05044440040004040000000014000440
Successfully validated signature of attestation token.
```

Copyright © 2026 Silicon Laboratories. All rights reserved.

# Util Commands

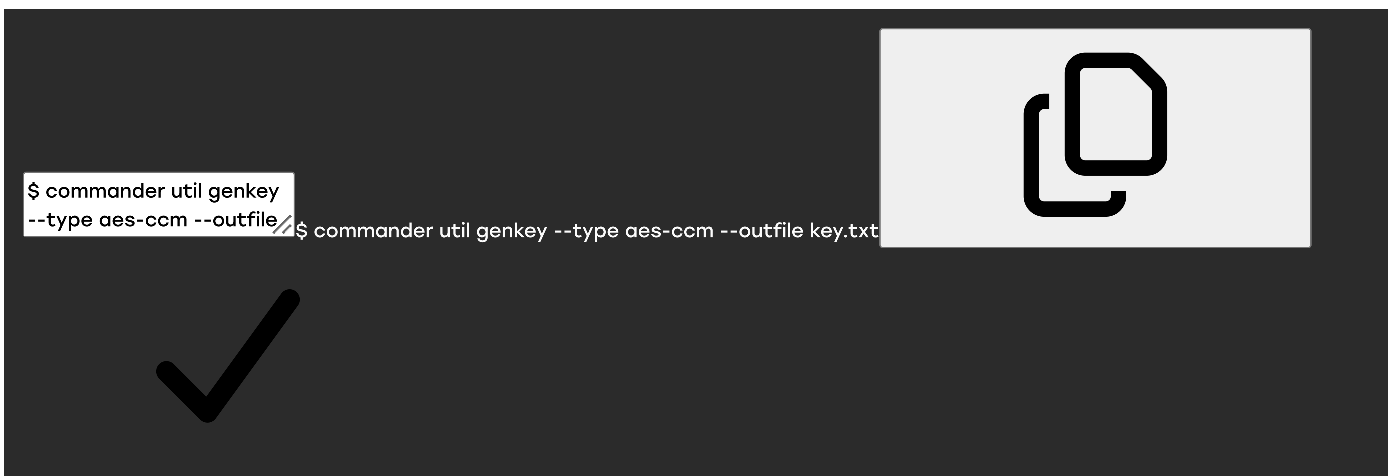
# Util Commands

## Key Generation

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

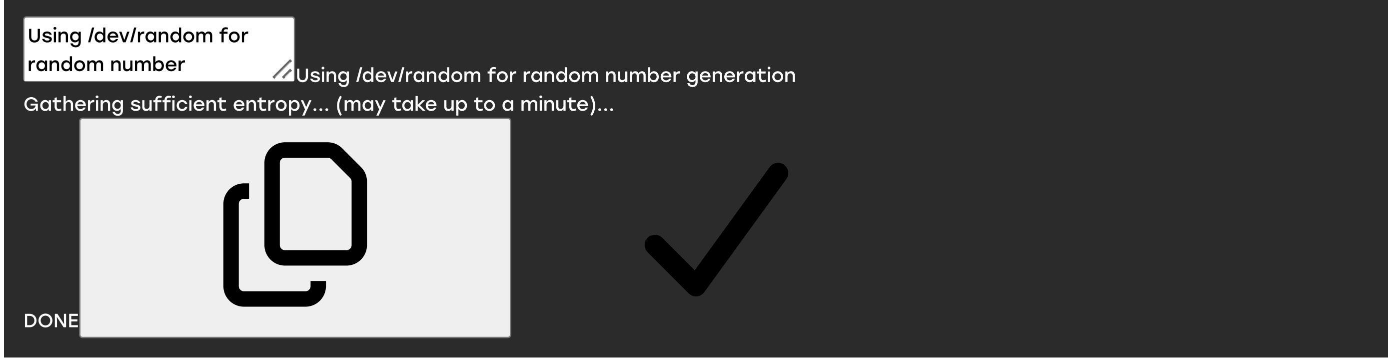
Command Line Syntax

Command Line Input Example




Command Line Output Example

Copyright © 2026 Silicon Laboratories. All rights reserved.



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

## Generating a Signing Key

Creates an EDCSA-P256 key pair and outputs the result to the specified private and public key files. For more information, see *UG266: Silicon Labs Gecko Bootloader User's Guide for GSDK 3.x and Lower* or *UG489: Silicon Labs Gecko Bootloader User's Guide for GSDK 4.0 and Higher.*

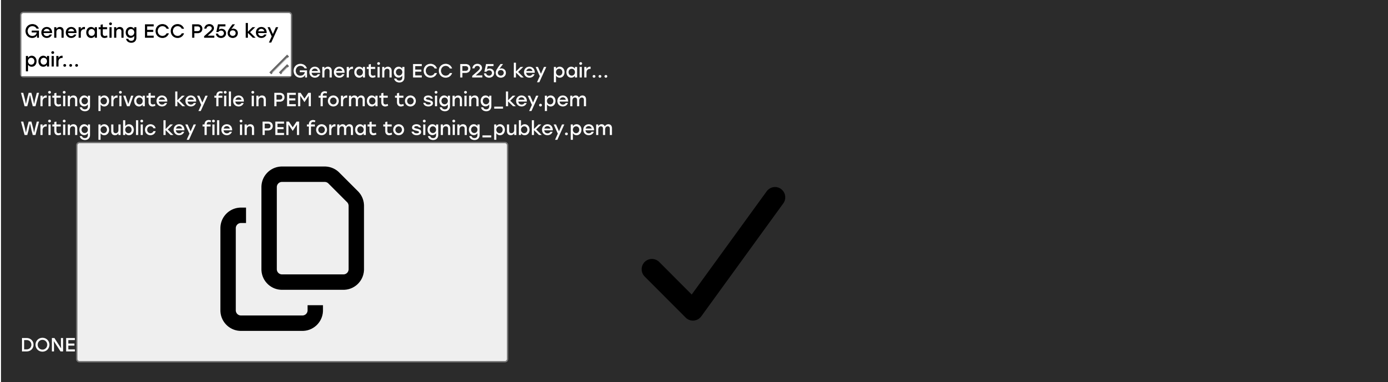
Command Line Syntax

```
$ commander util genkey --type ecc-p256 --privkey <filename> --pubkey <filename> [--tokenfile <filename>]
```

Command Line Input Example

```
$ commander util genkey --type ecc-p256 --privkey signing_key.pem --pubkey signing_pubkey.pem
```

Command Line Output Example



```
Generating ECC P256 key pair...
Writing private key file in PEM format to signing_key.pem
Writing public key file in PEM format to signing_pubkey.pem
DONE
```

Copyright © 2026 Silicon Laboratories. All rights reserved.

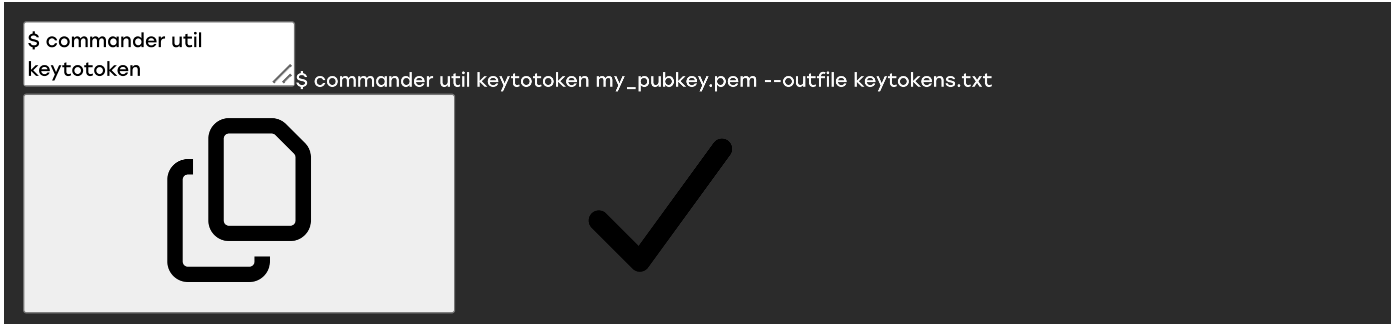
## Key to Token

Creates a token text file containing an Elliptic Curve Cryptography (ECC) public key suitable for flashing to a device. For more information, see *UG266: Silicon Labs Gecko Bootloader User's Guide for GSDK 3.x and Lower or UG489: Silicon Labs Gecko Bootloader User's Guide for GSDK 4.0 and Higher.*

Command Line Syntax

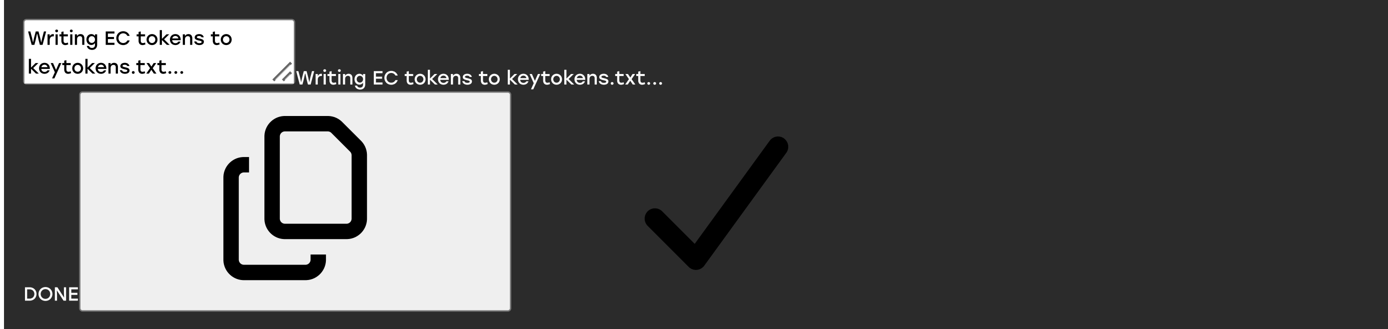
```
$ commander util keytotoken <input file> --outfile <filename>
```

Command Line Input Example



```
$ commander util keytotoken my_pubkey.pem --outfile keytokens.txt
```

Command Line Output Example



```
Writing EC tokens to keytokens.txt...
DONE
```

## Key Config Generation

Generates a key configuration file to the specified file name. This command is only available for SiWx91x devices, so the device options is required. The output file is used as input to the Provision Security Keys to the Device and Provision OTP Security Keys to the Device command, among others. The file contains the following keys:

- ATTESTATION_PRIVATE_KEY

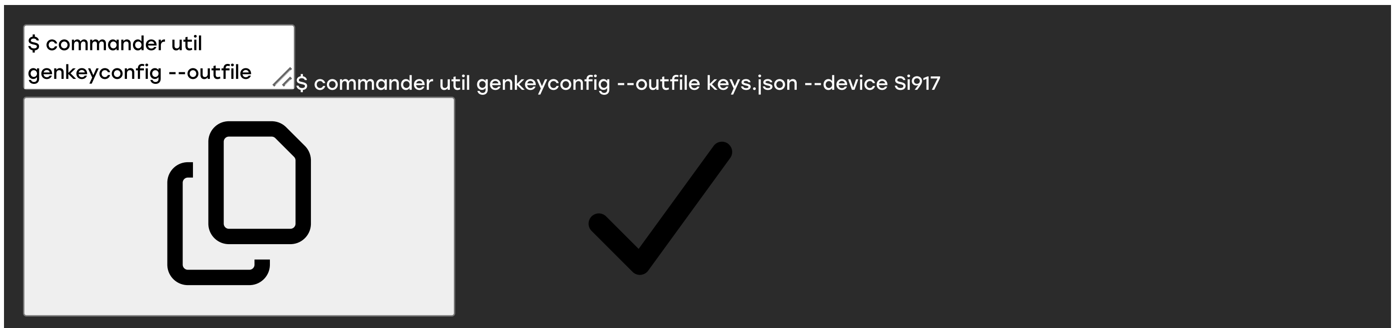
Copyright © 2026 Silicon Laboratories. All rights reserved.

- ATTESTATION_PUBLIC_KEY
- M4_OTA_KEY
- M4_PRIVATE_KEY
- M4_PUBLIC_KEY
- OTA_KEY
- NWP_PRIVATE_KEY
- NWP_PUBLIC_KEY
- OTP_AES_KEY
- OTP_PRIVATE_KEY
- OTP_PUBLIC_KEY

Command Line Syntax

```
$ commander util genkeyconfig --outfile <filename> --device <device>
```

Command Line Input Example

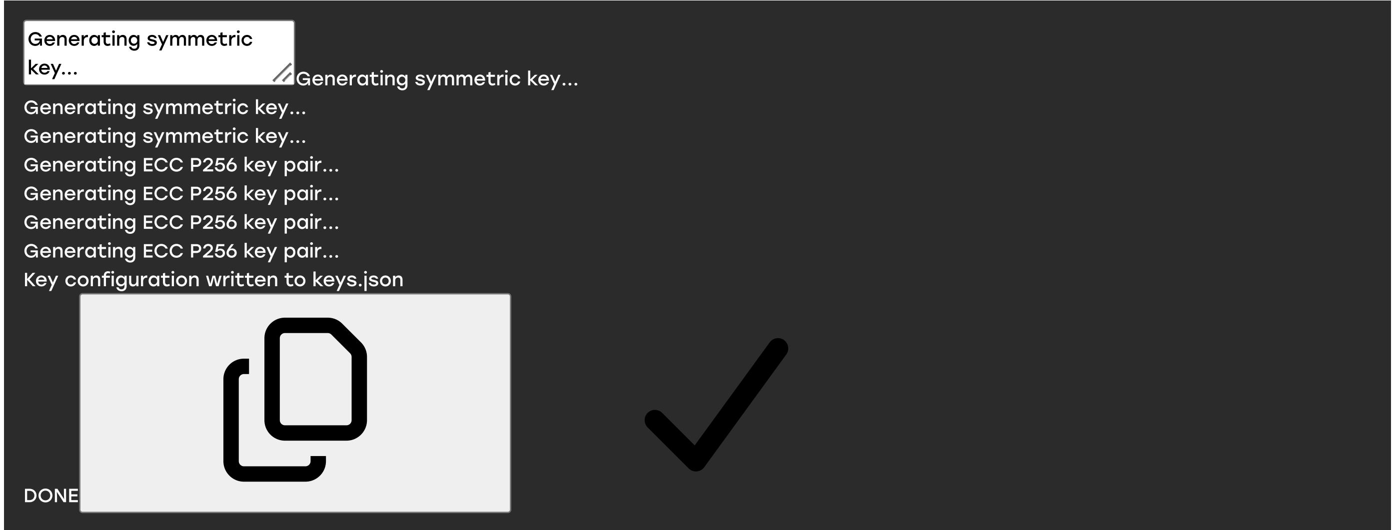


```
$ commander util genkeyconfig --outfile keys.json --device Si917
```

This example generates a file, `keys.json`, containing the key configuration for a Si917 device.

Command Line Output Example

Copyright © 2026 Silicon Laboratories. All rights reserved.

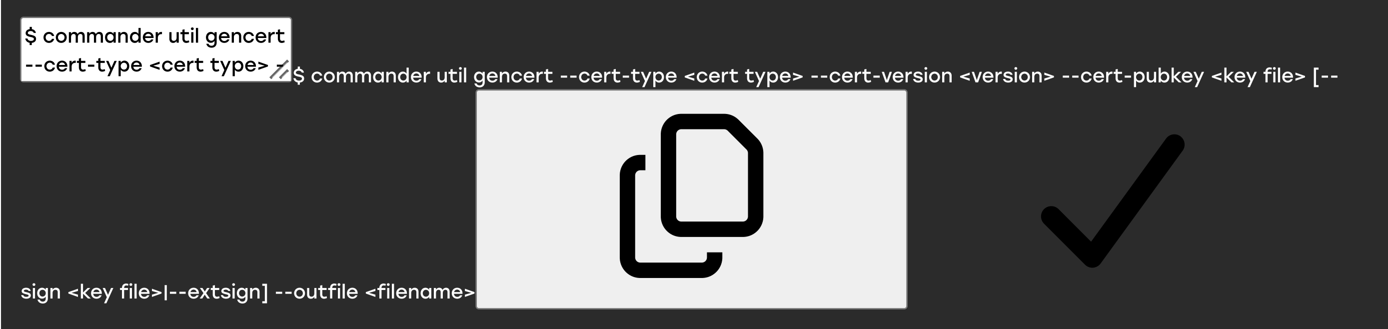


```
Generating symmetric key...
Generating symmetric key...
Generating symmetric key...
Generating ECC P256 key pair...
Generating ECC P256 key pair...
Generating ECC P256 key pair...
Generating ECC P256 key pair...
Key configuration written to keys.json
DONE
```

## Generate Certificate

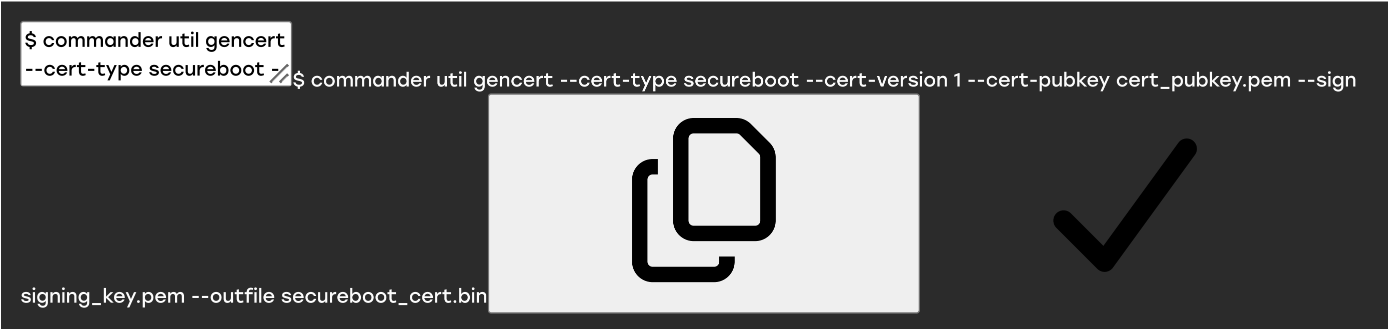
The process of signing files can be done using an intermediate certificate. These certificates can be generated with the `util gencert` command. There are currently two available certificate types: GBL certificates and Secure Boot certificates. If rollback prevention is enabled, the device will not boot if it has seen a certificate with a higher version number. This is set by the `--cert-version` option. The private key corresponding to the `--cert-pubkey` is used to sign the image. The certificate may either be signed directly by providing a signing key with the `--sign` option or unsigned by providing the `--extsign` option.

Command Line Syntax



```
$ commander util gencert --cert-type <cert type> --cert-version <version> --cert-pubkey <key file> [--sign <key file>|--extsign] --outfile <filename>
```

Command Line Input Example



```
$ commander util gencert --cert-type secureboot --cert-version 1 --cert-pubkey cert_pubkey.pem --sign signing_key.pem --outfile secureboot_cert.bin
```

In this example, the signing key is provided and the certificate is signed directly.

Command Line Output Example

Copyright © 2026 Silicon Laboratories. All rights reserved.

Command Line Input Example

In this example, an unsigned certificate is created. The signature for the certificate can be created, for example, by a Hardware Security Module (HSM). The certificate can be signed by passing the unsigned certificate and the HSM generated signature to the `util signcert` command.

Command Line Output Example

## Sign Certificate

Sign a certificate with an externally created signature. You can use the optional `--verify` option to verify the signature by providing the public key corresponding to the private key used to create the signature.

Command Line Syntax

Command Line Input Example

Copyright © 2026 Silicon Laboratories. All rights reserved.

Command Line Output Example

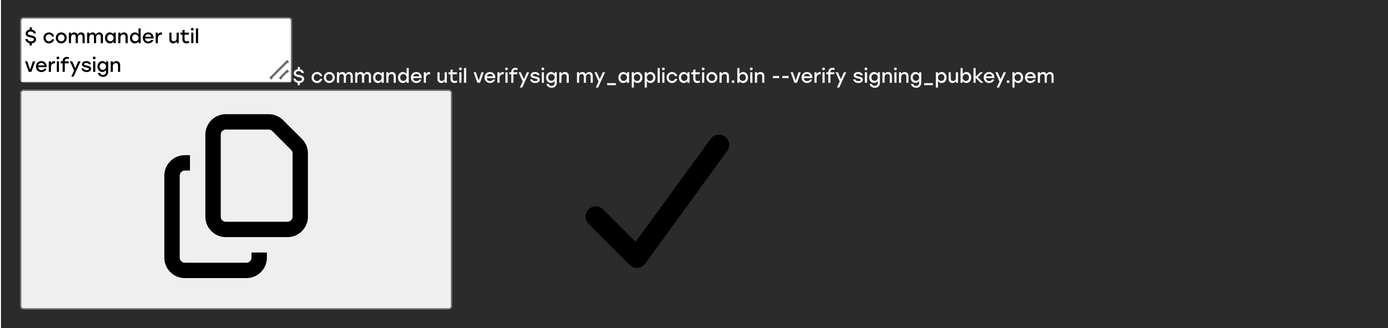
## Verify Signature

When secure boot is enabled, all code running on the device must be signed. This command can be used as a check to verify that the file was correctly signed, which may help in debugging in case secure boot fails, or as a verification before flashing the image. If the file is signed using an intermediate certificate, the certificate key is used to check the signature of the file. The key given by the `--verify` option is used to verify the signature of the certificate.

Command Line Syntax

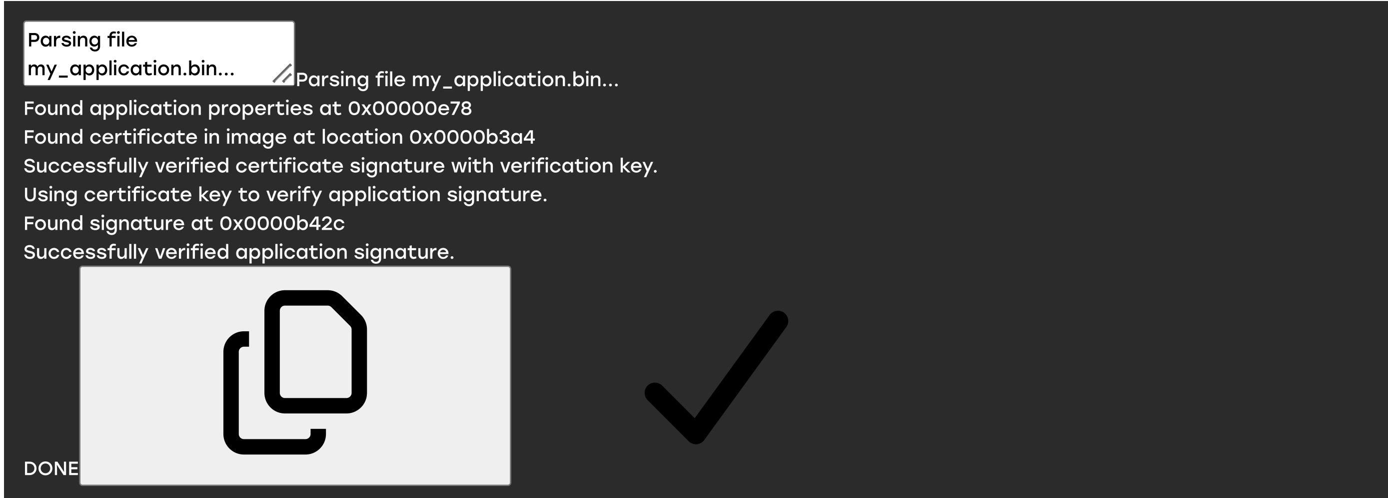
Command Line Input Example

Copyright © 2026 Silicon Laboratories. All rights reserved.



```
$ commander util verifysign my_application.bin --verify signing_pubkey.pem
```

Command Line Output Example



```
Parsing file my_application.bin...
Found application properties at 0x00000e78
Found certificate in image at location 0x0000b3a4
Successfully verified certificate signature with verification key.
Using certificate key to verify application signature.
Found signature at 0x0000b42c
Successfully verified application signature.
DONE
```

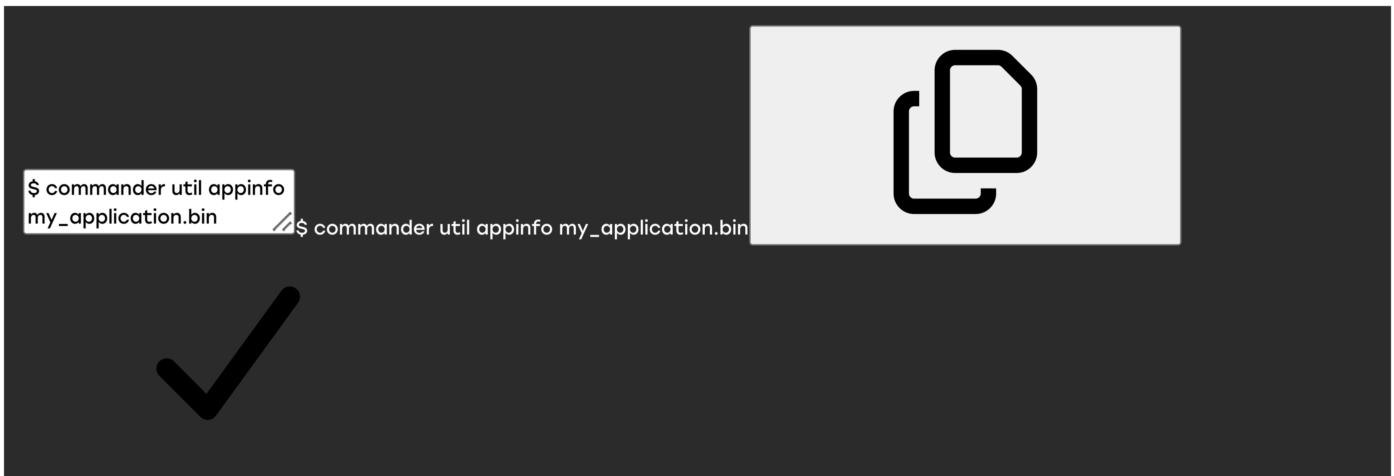
## Application Information

Get all available information about an application by parsing the `ApplicationProperties_t` struct in the image. If the file does not have application properties, no information can be extracted from the file.

Command Line Syntax

```
$ commander util appinfo <filename>
```

Command Line Input Example



```
$ commander util appinfo my_application.bin
```

Command Line Output Example

```
Parsing file my_application.bin...
Found application properties in image.
Application properties info:
Signature location         : 0x0000b42c
Signature type             : ECDSA-P256
Long token section address  : Not set (0x00000000)
Application data info:
If rollback prevention is enabled, the device will not boot if the device has seen an application with a higher version number.
App type                   : The application is an MCU application
App version                : 0x00000000
Product ID                 : 0x53455f555047524144455f4150500000
Application certificate info:
If rollback prevention is enabled, the device will not boot if the device has seen a certificate with a higher version number.
Certificate located at     : 0x0000b3a4
Certificate version        : 0x00000001
Certificate key            :
0x249919c28b28156f19d2e03379b968c8a931aa9b195258e2741da28b686983dd71d0140e9a7b0d7e39de43f592163b8aa38d4e0871f5d2d88b57

Certificate signature      :
0x013f2adc310f10f1426db74b503f3612a46ab85c7ce86c967eb965b10f7d24267101192513d9481c49c0eb0b61c1f73392cc6f6d1cd1209a9d58e76

DONE
```

## Print Section Header Information from an ELF File

Parse and print the section header information from an Executable and Linkable Format (ELF) file.

Command Line Syntax

Copyright © 2026 Silicon Laboratories. All rights reserved.

Command Line Input Example

Displays section header information of ELF file *my_bootloader.out*.

Command Line Output Example

Copyright © 2026 Silicon Laboratories. All rights reserved.

```
Index  Name
Size  Address Type
Index   Name              Size  Address Type
  1 .shstrtab         0x00000111  0x00000000 STRTAB
  2 .strtab           0x0001e169  0x00000000 STRTAB
  3 .symtab           0x000243a0  0x00000000 SYMTAB
  4 HEADERS            0x000000ac  0x00000000 PROGBITS
  5 APP ro            0x0002ddf4  0x00000200  PROGBITS
  6 SIMEE&LOCKBITS      0x00009000  0x000f7000   NOBITS
  7 ResetHeap          0x00001490  0x20000000  NOBITS
  8 Guard             0x00000030  0x20001490   NOBITS
  9 APP rw            0x00002148  0x2003dce0  NOBITS
 10 .debug_abbrev       0x00006325  0x00000000 PROGBITS
 11 .debug_aranges      0x000037ac  0x00000000 PROGBITS
 12 .debug_frame        0x0003a2f5  0x00000000 PROGBITS
 13 .debug_info        0x00063435  0x00000000 PROGBITS
 14 .debug_line        0x00064f5c  0x00000000 PROGBITS
 15 .debug_loc         0x00010fe3  0x00000000 PROGBITS
 16 .debug_macinfo      0x00009941  0x00000000 PROGBITS
 17 .debug_pubnames       0x00007132  0x00000000 PROGBITS
 18 .debug_ranges       0x00003778  0x00000000 PROGBITS
 19 .iar.debug_frame    0x00015349  0x00000000 PROGBITS
 20 .iar.debug_line    0x00020199  0x00000000 PROGBITS
 21 .comment           0x001d394a  0x00000000 PROGBITS
 22 .iar.rtmodel       0x00000032  0x00000000 PROGBITS
 23 .ARM.attributes     0x0000002e  0x00000000

DONE
```

## Get RAM and Flash Usage of an ELF Application

Calculate the static RAM usage and the flash storage usage of an application from an Executable and Linkable Format (ELF) file, and print usage details of the RAM sections.

If the `--map` option is provided with the path to the .map file created when building the application (only GCC map files are supported), the available RAM and flash storage will also be displayed.

If no map file is available, the `--device` option can be provided to let Commander infer the RAM and flash sizes of the device from its part number.

> Note: Any changes you might have introduced to the memory regions on your specific device will not be reflected if you are using the `--device` option.

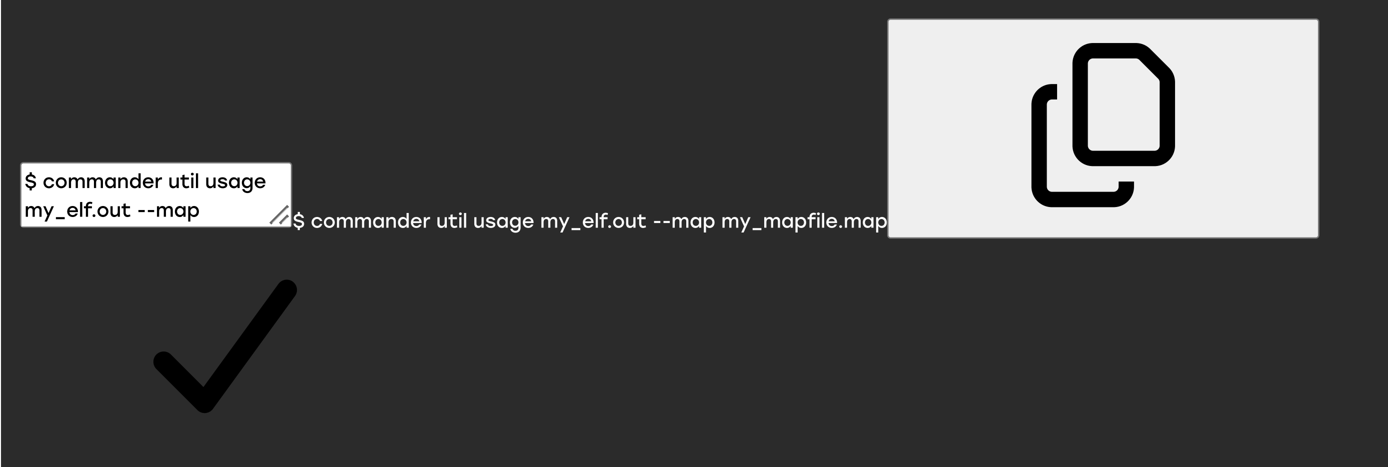
Command Line Syntax



Copyright © 2026 Silicon Laboratories. All rights reserved.

```
$ commander util usage <filename> [--map <filename>|--device <device part no.>] [--include-section <ELF section> --exclude-section <ELF section>]
```

Command Line Input Example



```
$ commander util usage my_elf.out --map my_mapfile.map
```

Command Line Output Example

```
Ram usage     :   262144 /   262144 B (100.00 %)
  .bss        :   3344 B           (  1.28 %)
  .data       :    152 B           (  0.06 %)
  .heap       :  254552 B          ( 97.10 %)
  .stack      :   4096 B           (  1.56 %)
Flash usage   :    23884 /  1564672 B (  1.53 %)
DONE
```

## Print Header Information of an RPS File

Parse and print the information contained in the header of an RPS file. The printed information includes (but is not limited to) security settings, signature data, bootloader instructions, flash address, image type, and image size. If the provided RPS file is a combined RPS image, the data for all constituent images is printed sequentially. If the image is encrypted, the bootloader instructions will be unavailable.

RPS files for on-device key upgrades are also supported by this command.

Command Line Syntax

Copyright © 2026 Silicon Laboratories. All rights reserved.

Command Line Input Example

This command line prints the information contained in the header of 'image.rps'.

Command Line Output Example

Copyright © 2026 Silicon Laboratories. All rights reserved.

```
RPS application image
Application info:
RPS application image
Application info:
Combined image bit set  : No
Image type             : TA application
Image size             : 0x001986A0 (1672864 B)
Flash address          : 0x00011000
Firmware version       : 0x020101BF
Firmware version ext.  : 0x1610ABFF
Counter                : 0x00000000 (0)
PSRAM                  : No
Security settings:
Integrity check        : CRC
CRC                    : 0x844D33FA (2219652090)
Encrypted              : No
Signed                 : No
Boot descriptor info:
Boot desc. offset      :    0x0080
IVT offset             : 0x00000000
3 boot descriptor entries found:
Length      :  0x000140 (320)
Destination : 0x00000000
Length      :  0x000CFC (3324)
Destination : 0x00000B04
Length      :  0x01A984 (108932)
Destination : 0x0000E948
DONE
```

Copyright © 2026 Silicon Laboratories. All rights reserved.

OTA Commands

# OTA Commands

## Create an OTA Bootloader File

Creates a Zigbee Over-the-air (OTA) bootloader file from one or more Gecko Bootloader (GBL) files and writes the output to the specified OTA file.

Command Line Syntax

```
$ commander ota create --upgrade-image <filename> --manufacturer-id <ID> --image-type <image type> --firmware-version <version> --string <text> -o <outfile> [--manufacture-tag <tag ID:filename> -stack-version <version> --credentials <credentials> --destinations <EUI64> --min-hw <version> --max-hw <version>]
```

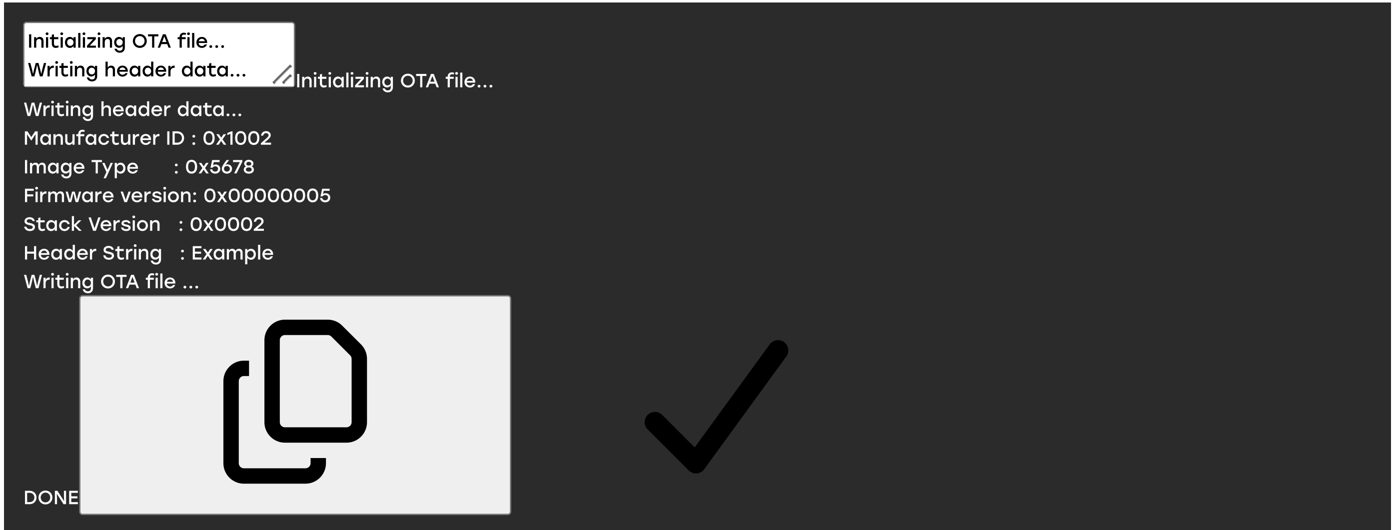
Command Line Input Example

```
$ commander ota create --upgrade-image example.gbl --manufacturer-id 0x1002 --image-type 0x5678 --firmware-version 0x00000005 --string "Example" -o example.ota
```

Creates an OTA file *example.ota* from the GBL upgrade image *example.gbl*.

Command Line Output Example

Copyright © 2026 Silicon Laboratories. All rights reserved.



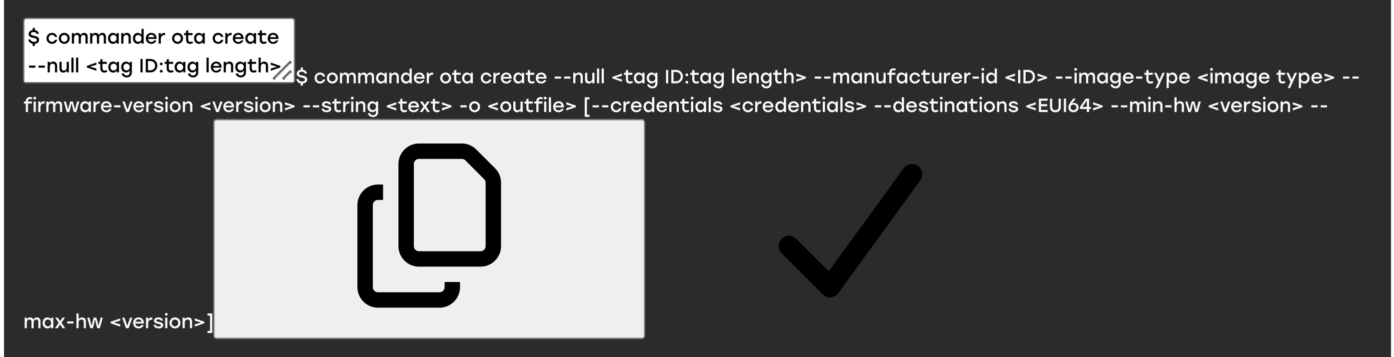
```
Initializing OTA file...
Writing header data...
Manufacturer ID : 0x1002
Image Type      : 0x5678
Firmware version: 0x00000005
Stack Version   : 0x0002
Header String   : Example
Writing OTA file ...
DONE
```

## Create a Null OTA File

The certification process for the Zigbee Over-the-Air (OTA) Bootload cluster client requires that the manufacturer provides a NULL upgrade file to the test house for testing. A NULL OTA upgrade file does not contain an actual upgrade image inside it (such as a Gecko Bootloader (GBL) file). It is much smaller than a full upgrade image, but otherwise the same as a normal Zigbee OTA file.

You create NULL files by using the `--null` option instead of the `--upgrade-image` option. The `--null` option consists of a tag ID and a tag length. The tag ID should be something other than 0x0000, which Zigbee has defined as "Upgrade Image". The tag length is a number of bytes, usually something small, such as 10. This option generates a sequence of bytes, starting at 0 and incrementing based on the tag length passed in.

Command Line Syntax



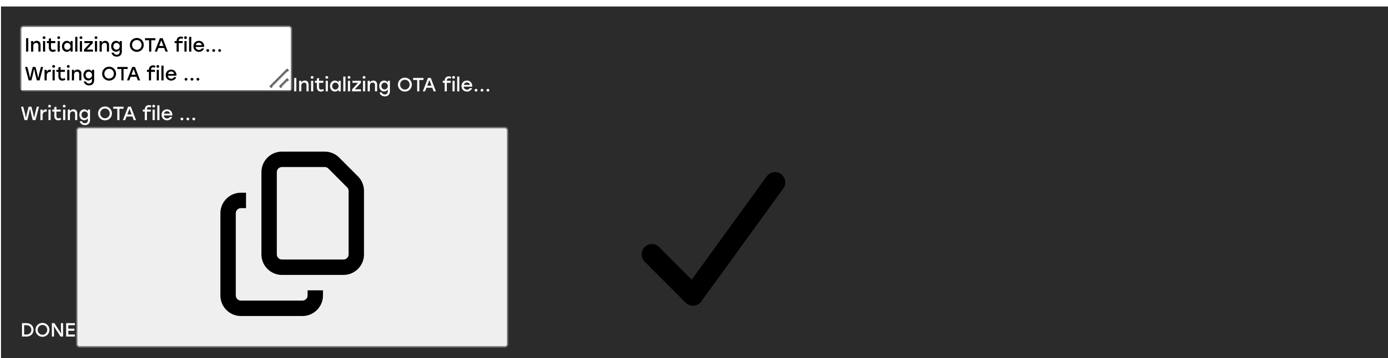
```
$ commander ota create --null <tag ID:tag length> --manufacturer-id <ID> --image-type <image type> --firmware-version <version> --string <text> -o <outfile> [--credentials <credentials> --destinations <EUI64> --min-hw <version> --max-hw <version>]
```

Command Line Input Example

```
$ commander ota create --null 0xffff:10 --manufacturer-id 0x110c --image-type 0x5678 --firmware-version 0x0102 --string "NULL OTA file"  -o ~/projects/Binaries/null.ota
```

Creates a NULL OTA file with a tag ID `0xffff` and a tag length of 10 bytes.

Command Line Output Example



```
Initializing OTA file...
Writing OTA file ...
DONE
```

## Print OTA File Information

Parses and prints the contents of an Over-the-air (OTA) file.

Command Line Syntax

```
$ commander ota parse <ota file>
```

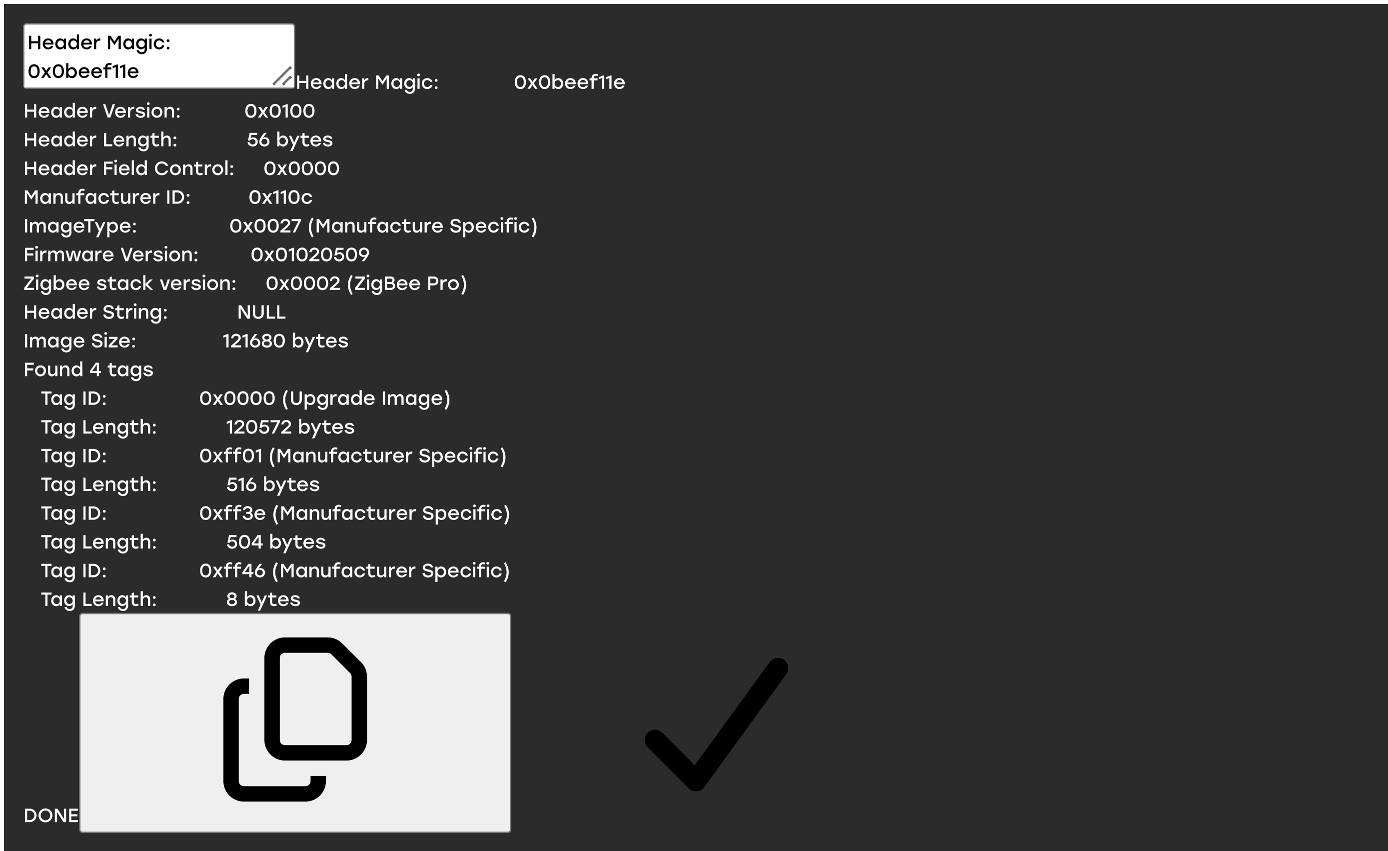
Command Line Input Example

Copyright © 2026 Silicon Laboratories. All rights reserved.

```
$ commander ota parse example.ota
```

Displays content of the OTA file *example.ota*.

Command Line Output Example



```
Header Magic:           0x0beef11e
Header Version:         0x0100
Header Length:          56 bytes
Header Field Control:   0x0000
Manufacturer ID:        0x110c
ImageType:              0x0027 (Manufacture Specific)
Firmware Version:       0x01020509
Zigbee stack version:   0x0002 (ZigBee Pro)
Header String:          NULL
Image Size:             121680 bytes
Found 4 tags
  Tag ID:               0x0000 (Upgrade Image)
  Tag Length:           120572 bytes
  Tag ID:               0xff01 (Manufacturer Specific)
  Tag Length:           516 bytes
  Tag ID:               0xff3e (Manufacturer Specific)
  Tag Length:           504 bytes
  Tag ID:               0xff46 (Manufacturer Specific)
  Tag Length:           8 bytes
DONE
```

## Sign an OTA File

The Zigbee Smart Energy Profile requires that the manufacturer signs Over-the-Air (OTA) files. The OTA client must validate the downloaded files prior to installation. Images are signed using certificates issued by Certicom. After the images are signed, the signer's certificate is included automatically as a tag in the OTA file, and a signature tag is added as the last tag in the OTA file. For more information, see *AN714: Smart Energy ECC-Enabled Device Setup Process*.

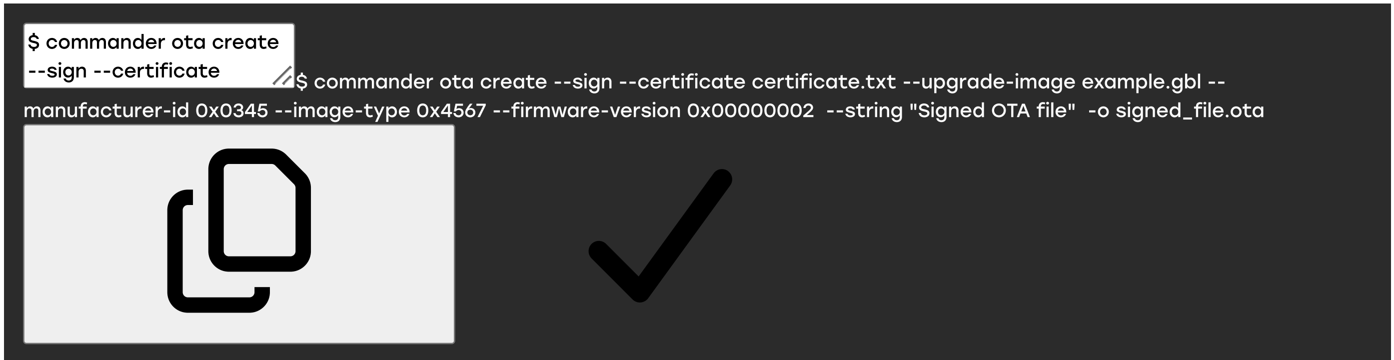
> Note: MacOS does not support OTA signing.

Copyright © 2026 Silicon Laboratories. All rights reserved.

Command Line Syntax

```
$ commander ota create --sign --certificate <certificate>  --upgrade-image <filename> --manufacturer-id <ID> --image-type <image type> --firmware-version <version> --string <text> -o <outfile> [--credentials <credentials> --destinations <EUI64> --min-hw <version> --max-hw <version>]
```

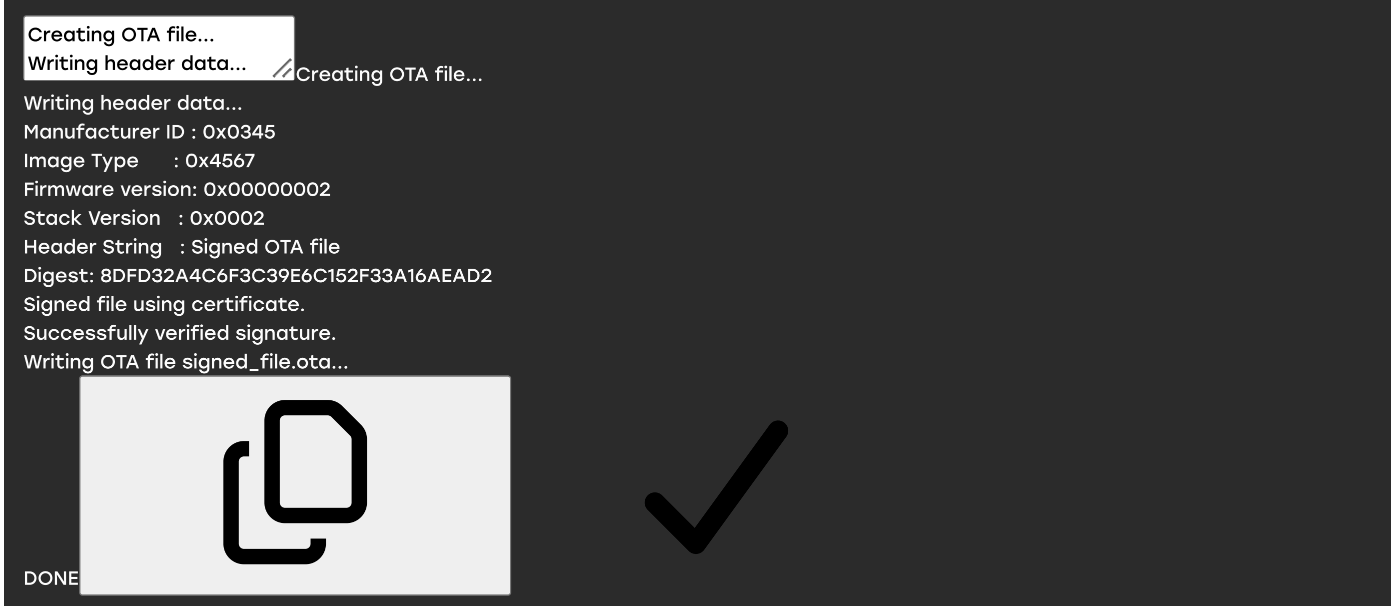
Command Line Input Example



```
$ commander ota create --sign --certificate certificate.txt --upgrade-image example.gbl --manufacturer-id 0x0345 --image-type 0x4567 --firmware-version 0x00000002  --string "Signed OTA file"  -o signed_file.ota
```

Creates a signed OTA file using certificate `certificate.txt.`

Command Line Output Example



```
Creating OTA file...
Writing header data...
Manufacturer ID : 0x0345
Image Type      : 0x4567
Firmware version: 0x00000002
Stack Version   : 0x0002
Header String   : Signed OTA file
Digest: 8DFD32A4C6F3C39E6C152F33A16AEAD2
Signed file using certificate.
Successfully verified signature.
Writing OTA file signed_file.ota...
DONE
```

Copyright © 2026 Silicon Laboratories. All rights reserved.

## Create an OTA File for External Signing

Create an Over-the-air (OTA) image to be signed externally. The external certificate is added to the image. The signature can be added to the image using the sign command in the following section.

Command Line Syntax

$ commander ota create --extsign --certificate

```
$ commander ota create --extsign --certificate <certificate>  --upgrade-image <filename> --manufacturer-id <ID> --image-type <image type> --firmware-version <version> --string <text> -o <outfile> [--credentials <credentials> --destinations <EUI64> --min-hw <version> --max-hw <version>]
```

Command Line Input Example

$ commander ota create --extsign --certificate

```
$ commander ota create --extsign --certificate certificate.txt --upgrade-image example.gbl --manufacturer-id 0x0345 --image-type 0x4567 --firmware-version 0x00000002  --string "Silicon Labs Ota Support"  -o example_file.ota
```

Command Line Output Example

Creating OTA file...
Writing header data...

```
Creating OTA file...
Writing header data...
Manufacturer ID : 0x0345
Image Type      : 0x4567
Firmware version: 0x00000002
Stack Version   : 0x0002
Header String   : Silicon Labs Ota Support
Writing OTA file example_file.ota.extsign...
DONE
```

Copyright © 2026 Silicon Laboratories. All rights reserved.

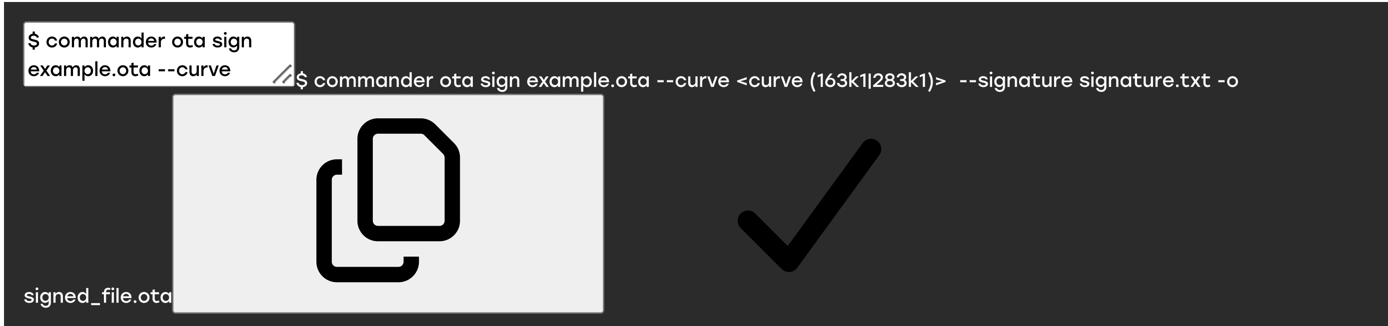
## Externally Sign an OTA File

Use the Simplicity Commander `sign` command to append an externally created signature to an Over-the-Air (OTA) file. You must specify the curve used to create the signature using the `--curve` option. Available curves are 163k1 or 283k1.

Command Line Syntax

```
$ commander ota sign <filename> --curve <curve (163k1|283k1)>  --signature <filename> -o <outfile>
```

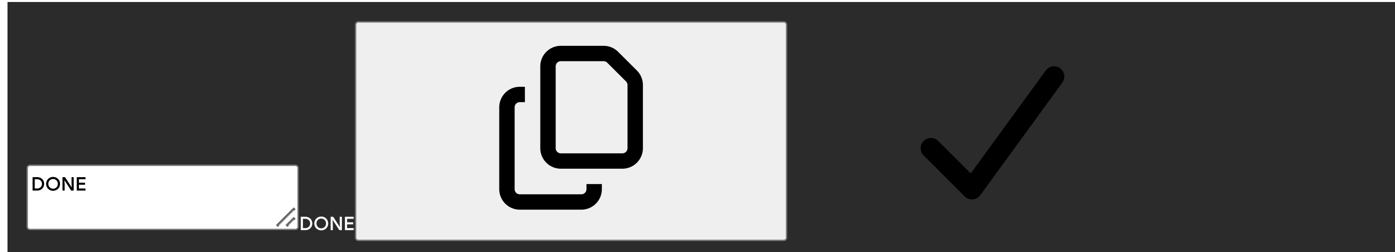
Command Line Input Example



```
$ commander ota sign example.ota --curve <curve (163k1|283k1)>  --signature signature.txt -o signed_file.ota
```

Appends the externally created signature to the OTA file.

Command Line Output Example



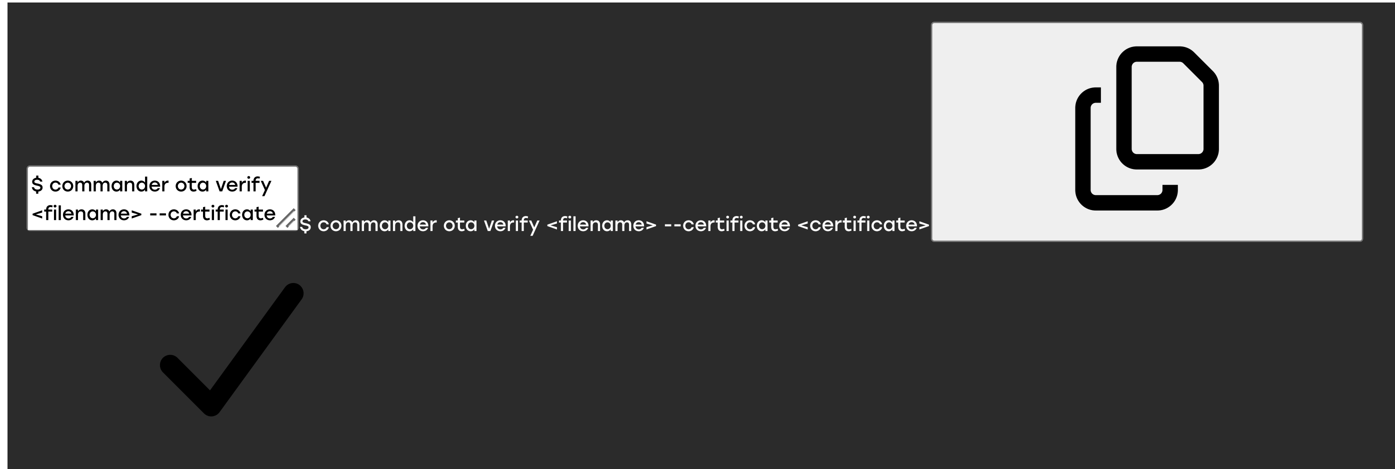
```
DONE
```

## Verify Signature of an OTA File

Use the Simplicity Commander `verify` command to verify the signature of an Over-the-Air (OTA) file. You must provide the certificate used to sign the file.

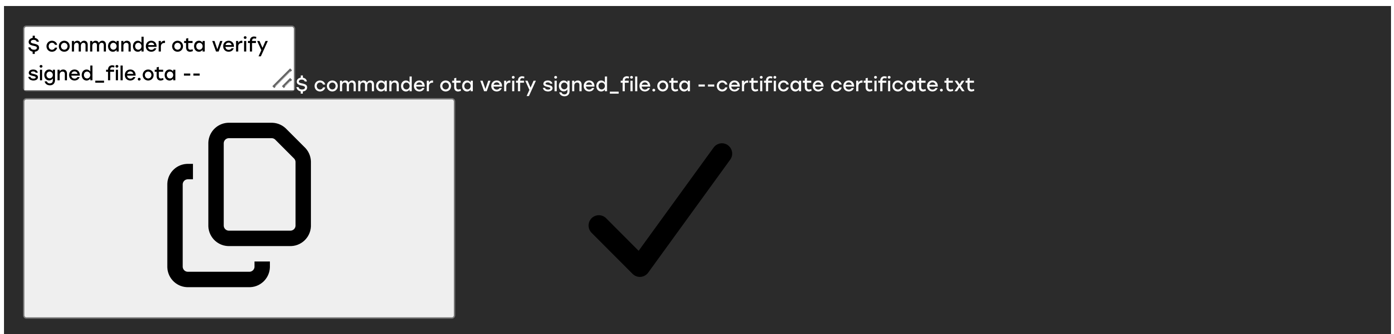
> Note: MacOS does not support OTA signature verification.

Command Line Syntax



```
$ commander ota verify <filename> --certificate <certificate>
```

Command Line Input Example



```
$ commander ota verify signed_file.ota --certificate certificate.txt
```

Verifies the signature of the OTA file.

Command Line Output Example

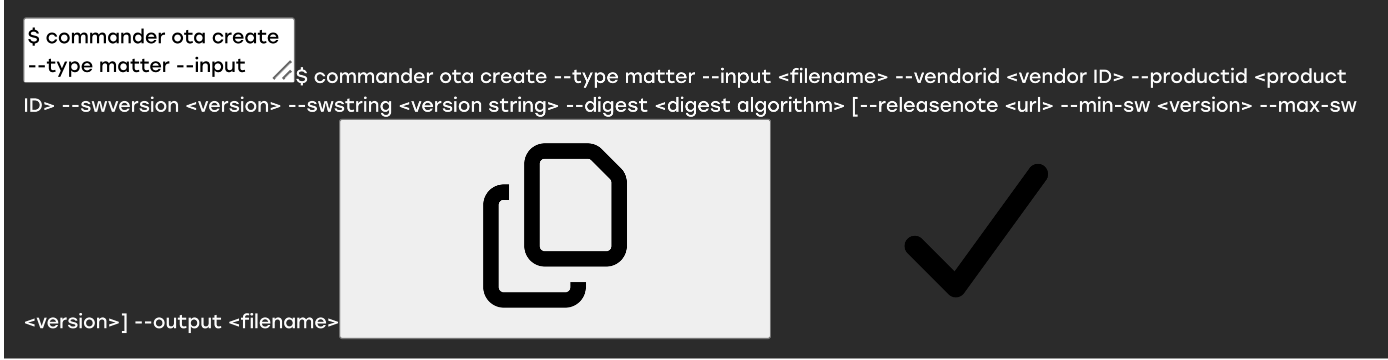
```
Digest: 8ABB04618622595401AD45FA33C7D670
Successfully verified signature
DONE
```

## Create an OTA Matter File

Create a Matter Over-the-air (OTA) software update file from an application and write the output to the specified OTA file.

Command Line Syntax

Copyright © 2026 Silicon Laboratories. All rights reserved.



```
$ commander ota create --type matter --input <filename> --vendorid <vendor ID> --productid <product ID> --swversion <version> --swstring <version string> --digest <digest algorithm> [--releasenote <url> --min-sw <version> --max-sw <version>] --output <filename>
```

Supported digest algorithms are

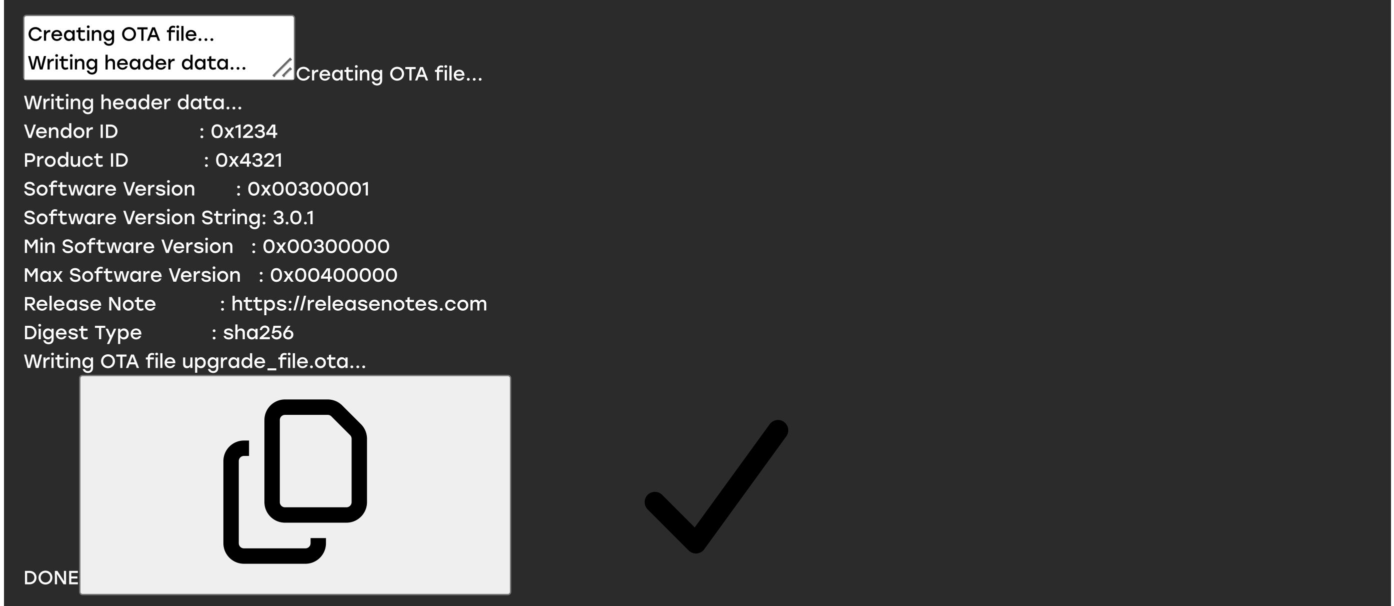
- sha256
- sha384
- sha512
- sha3_224
- sha3_256
- sha3_384
- sha3_512

Command Line Input Example

```
$ commander ota create --type matter --vendorid 0x1234 --productid 0x4321 --swversion 0x300001 --swstring "3.0.1" --input application.bin --digest sha256 --releasenote "https://releasenotes.com" --min-sw 0x300000 --max-sw 0x400000 --output upgrade_file.ota
```

Creates an OTA file *upgrade_file.ota* from the application image *application.bin*.

Command Line Output Example



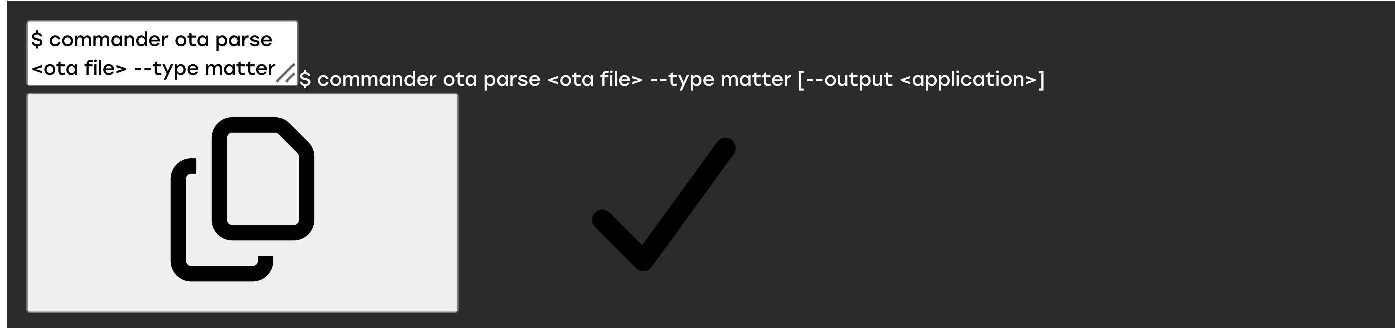
```
Creating OTA file...
Writing header data...
Vendor ID              : 0x1234
Product ID             : 0x4321
Software Version       : 0x00300001
Software Version String: 3.0.1
Min Software Version   : 0x00300000
Max Software Version   : 0x00400000
Release Note           : https://releasenotes.com
Digest Type            : sha256
Writing OTA file upgrade_file.ota...
DONE
```

Copyright © 2026 Silicon Laboratories. All rights reserved.

## Parse a Matter OTA File

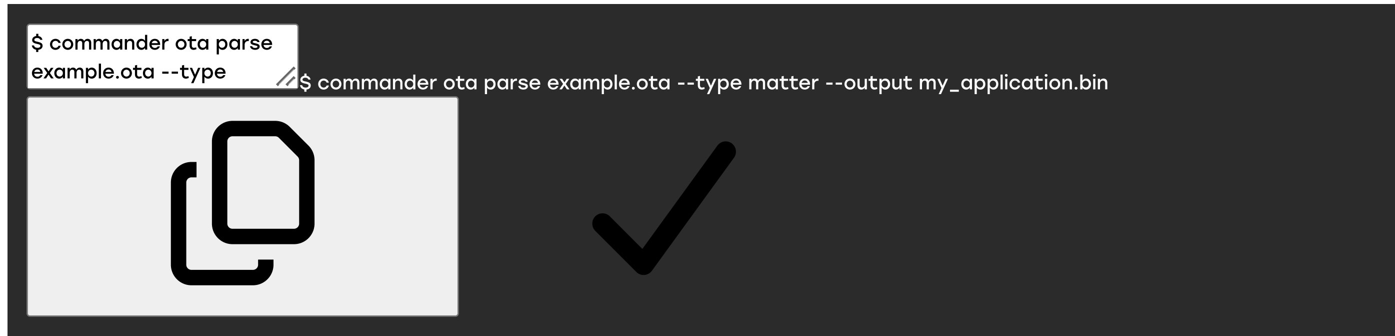
Parse and print the contents of a Matter Over-the-air (OTA) software update file. The optional `--output` option extracts the application from the OTA file and writes it to the file specified by the `--output` option.

Command Line Syntax



```
$ commander ota parse <ota file> --type matter [--output <application>]
```

Command Line Input Example



```
$ commander ota parse example.ota --type matter --output my_application.bin
```

Displays content of the OTA file *example.ota* and extracts the application from the OTA file and writes it to *my_application.bin*.

Command Line Output Example

```
Magic: 1beef11e
Total Size  : 977 bytes
Header Size : 105 bytes
Header TLV:
[0] Vendor ID     : 4660     (0x1234)
[1] Product ID    : 17185    (0x4321)
[2] Version       : 3145729  (0x300001)
[3] Version String: 3.0.1
[4] Payload Size  : 856      (0x358)
[5] Min Version   : 3145728  (0x300000)
[6] Max Version   : 4194304  (0x400000)
[7] Release Notes : https://releasenotes.com
[8] Digest Type   : 1        (0x1)
[9] Digest        : 8f259c4727adbe755ec1a49e8bfdbedb3486f53721cae5434efe5d9971eb5d55
Writing application to my_application.bin...
```

Copyright © 2026 Silicon Laboratories. All rights reserved.

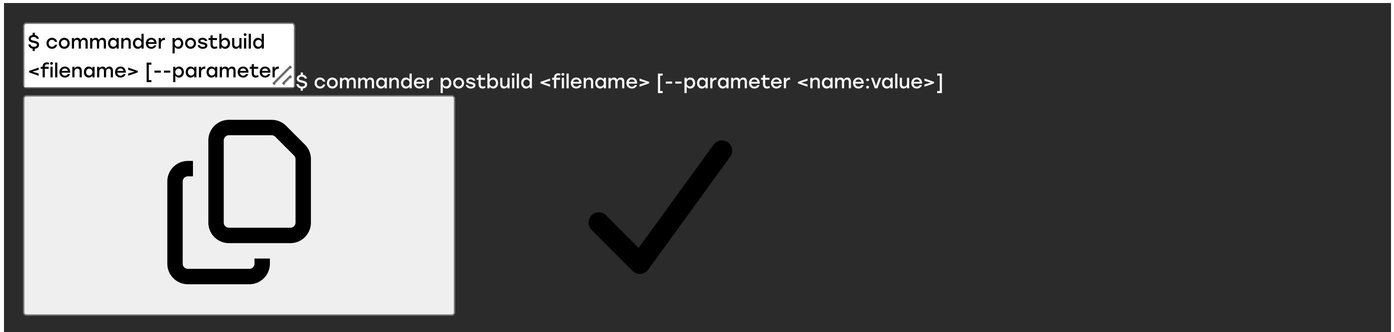
# Post-Build Command

# Post-Build Command

## Execute a Project Post-Build File

Simplicity Commander takes a project post-build description file in Yaml Ain't Markup Language (YAML) format, produced by Simplicity Studio, and executes sequentially the specified tasks in the file.

Command Line Syntax




Command Line Input Example

Executes the steps in the post-build pipeline defined in *project_name.slpb*.

Command Line Output Example

Copyright © 2026 Silicon Laboratories. All rights reserved.

```
Parsing file project_name.slbp...
Running task copy...
Running task convert...
Running task GBL create...
Running task OTA create...
DONE
```

The post-build pipeline consists of three sections:

- Parameters: named variables whose value is taken from the command line upon pipeline invocation.
- Constants: named variables whose value is taken from the post-build file itself or a path to another post-build file from which constants are inherited.
- Steps: list of tasks to be invoked, making use of the above declared variables.

See below for an example post-build file:

```
parameters:
  - name: artifact
  - name: build_dir
constants:
  - name: project_name
    value: my_project
steps:
  - task: copy
    input: "{{build_dir}}/{{project_name}}.s37"
    output: "{{artifact}}/{{project_name}}.s37"
  - task: convert
    input: "{{build_dir}}/{{project_name}}.out"
    include-section: P2 ro
    output: "{{artifact}}/{{project_name}}.bin"
```

## Tasks

Seven different types of tasks are supported. The tasks are identified with the following names:

- *copy*
- *convert*
- *convert_rps*
- *create_gbl*
- *create_gbl4*
- *create_ota*
- *create_rps*
- *usage*

The tables below summarize the required options and optional options for each task.

Copyright © 2026 Silicon Laboratories. All rights reserved.

Table: *copy*

| Required Options |
| --- |
| input: <filename> |
| output: <filename> |
| Optional Options |
| export: <constant value> |

Table: *convert*

| Required Options |
| --- |
| input: <filename> |
| output: <filename> |
| Optional Options |
| export: <constant value> |
| keyfile: <key file> |
| crc: <true> |
| certificate: <certificate file> |
| include-section: <ELF section> |
| exclude-section: <ELF section> |
| signature: <signature file> |
| verify: <key file> |

Table: *convert_rps*

| Required Options |
| --- |
| output: <RPS filename> |
| Optional Options |
| app: <M4 RPS filename> |
| taapp: <TA RPS filename> |
| app-version: <version number> |
| fw-info: <firmware info> |
| sign: <key filename> |
| sha-type: <SHA-XXX> |
| encrypt: <key filename> |
| mic: <key filename> |
| combinedimage: <true> |

Table: *create_gbl*

| Required Options |
| --- |
| output: <filename> |
| Optional Options |
| export: <constant value> |
| app: <app image> |
| bootloader: <bootloader image> |

Copyright © 2026 Silicon Laboratories. All rights reserved.

| Required Options |
| --- |
| seupgrade: <SE upgrade image> |
| metadata: <metadata bin file> |
| compress: <app compression algorithm> |
| certificate: <certificate file> |
| sign: <key file> |
| encrypt: <AES key file> |
| extsign: <true> |
| include-section: <section> |
| exclude-section: <section> |

Table: *create_gbl4*

| Required Options |
| --- |
| output: <filename> |
| Optional Options |
| export: <constant value> |
| data: <app or bootloader image> |
| seupgrade: <SE upgrade image> |
| compress: <app compression algorithm> |
| certificate: <certificate file> |
| sign: <key file> |
| encrypt: <AES key file> |
| productid: <product id> |
| bundleversion: <version> |
| minversion: <version> |
| hash: <hash function> |

Table: *create_ota*

| Required Options |
| --- |
| input: <filename> (same as upgrade-image) |
| output: <filename> |
| manufacturer-id: <ID> |
| firmware-version: <version> |
| image-type: <image type> |
| string <text> |
| Optional Options |
| export: <constant value> |
| upgrade-image: <filename> |
| manufacturer-tag: <tag ID> |
| stack-version: <version> |
| credentials: <credentials> |
| destination: <EUI64> |
| min-hw: <version> |

Copyright © 2026 Silicon Laboratories. All rights reserved.

| Required Options |
| --- |
| max-hw: <version> |
| certificate: <filename> |
| sign: <true> |

Table: *create_rps*

| Required Options |
| --- |
| input: <application filename> |
| output: <RPS filename> |
| Optional Options |
| address: <address> |
| app-version: <version number> |
| include-section: <section> |
| exclude-section: <section> |
| fw-info: <firmware info> |
| sign: <key filename> |
| sha-type: <SHA-XXX> |
| encrypt: <key filename> |
| mic: <key filename> |
| combinedimage: <true> |

Table: *usage*

| Required Options |
| --- |
| input: <application ELF filename> |
| Optional Options |
| map: <filename> |
| device: <device part number> |
| include-section: <ELF section> |
| exclude-section: <ELF section> |

Copyright © 2026 Silicon Laboratories. All rights reserved.

RPS Commands

# RPS Commands

SiWx917 devices require that application binaries are converted to RPS images before flashing. Simplicity Commander can be used to convert M4 application binaries to RPS images, apply security features, and to combine multiple RPS images into a single RPS file.

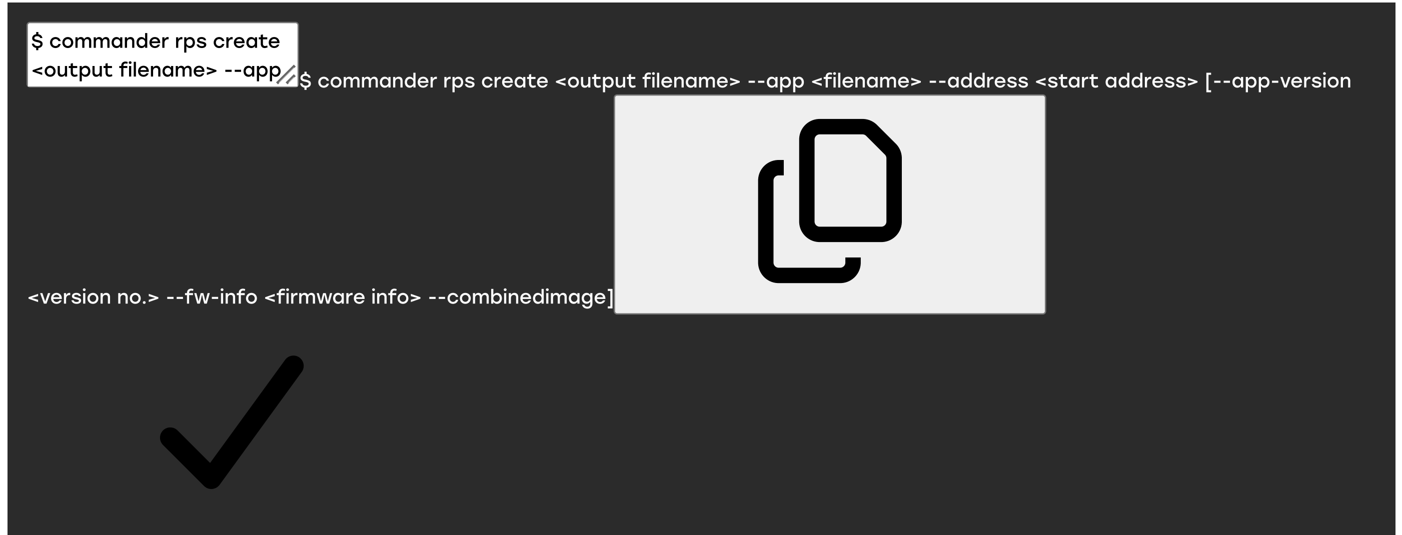
Simplicity Commander's RPS image creation supports bin, hex, SRec and ELF image formats. Commander will prepend an RPS style header to the provided application image, containing information used by the device's bootloader. An application version number may be provided using the `--app-version` option, and additional firmware/device information can be provided using the `--fw-info` option. If you intend on combining one or multiple RPS application images, the `--combinedimage` flag can be provided to prepare the image for combining with other eligible RPS images.

Simplicity Commander also supports creating RPS key images for upgrading on-device M4 keys.

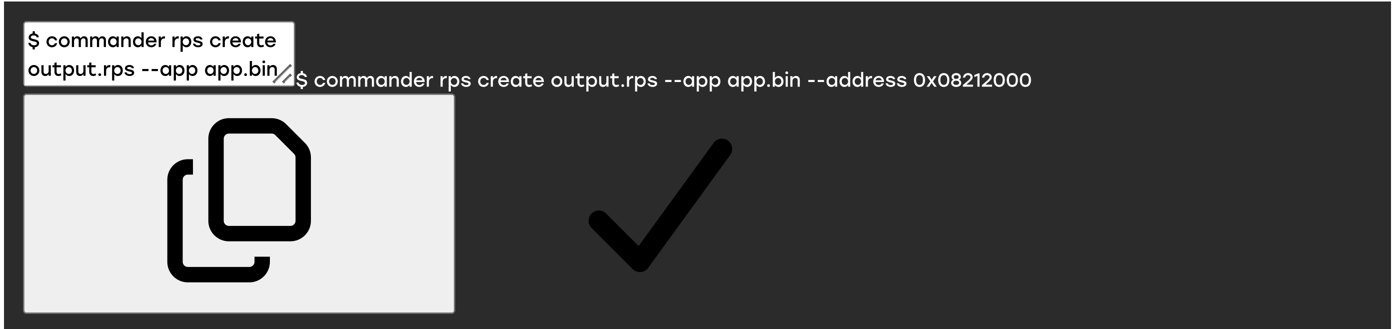
## Create an RPS File From a Binary Image

To create an RPS file from a binary image, you must provide an application start address using the `--address` flag.

Command Line Syntax



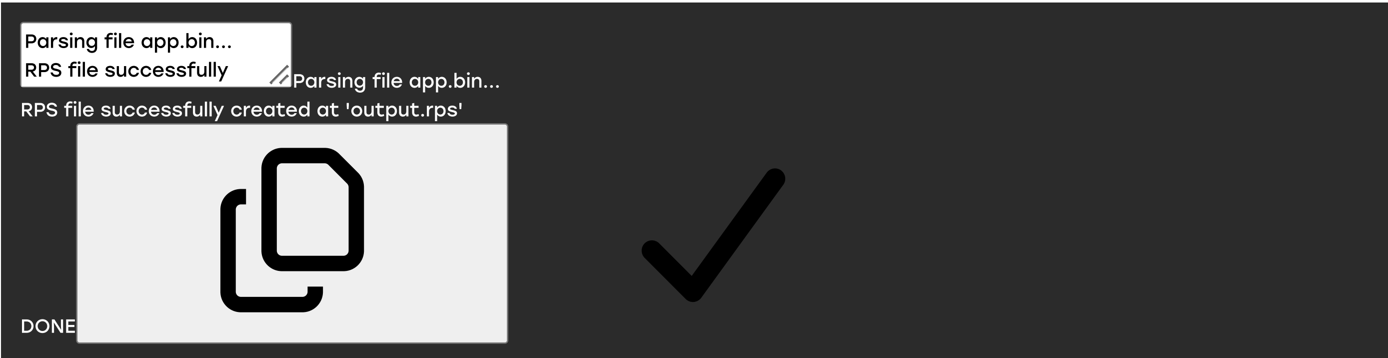

Command Line Input Example




Copyright © 2026 Silicon Laboratories. All rights reserved.

This command line creates an RPS file from a binary image with flash address '0x08212000' and saves it to the file named 'output.rps'.

Command Line Output Example

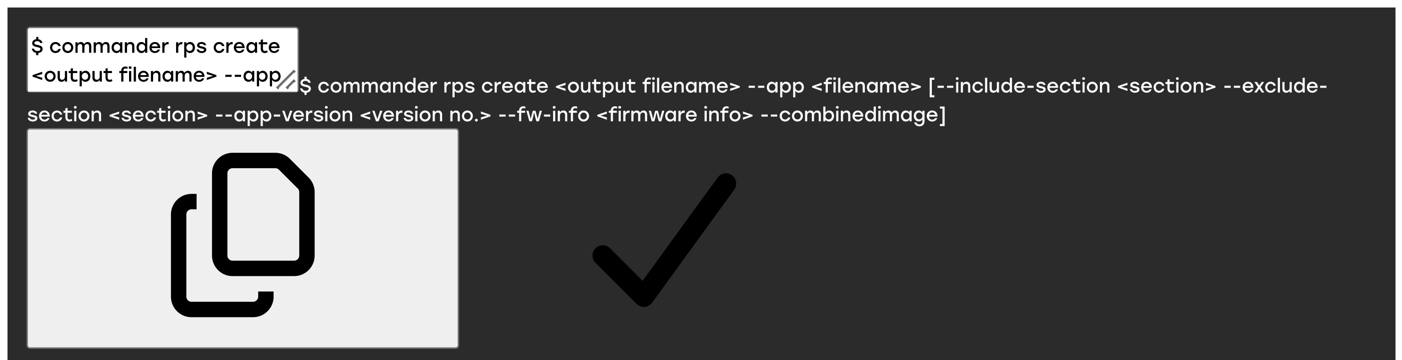



## Create an RPS File From an ELF Image

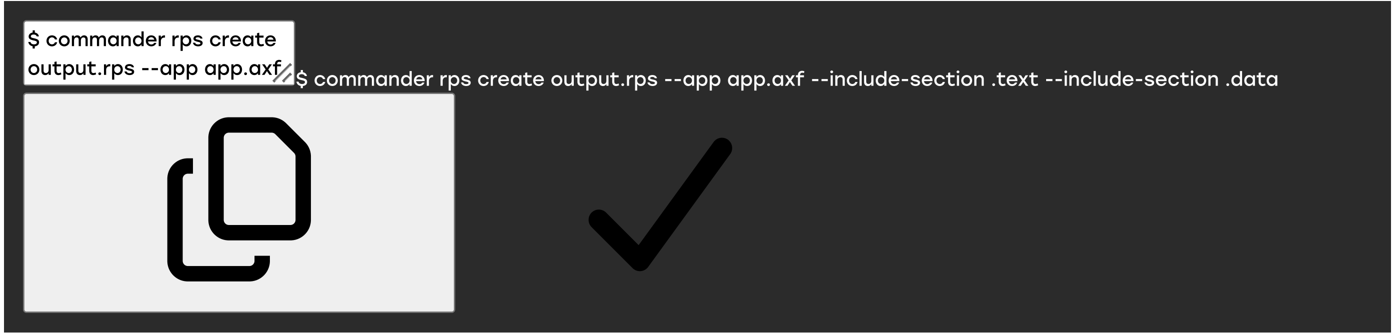
When generating an RPS file from an Execution and Linkable Format (ELF) image, you can use the `--include-section` and `--exclude-section` options to either include or exclude certain ELF sections from the application image of the output RPS file. If neither of these options is provided, Simplicity Commander will include all sections that appear to be part of the application.

You can include or exclude multiple sections by providing the respective options repeatedly.

Command Line Syntax




Command Line Input Example




This command line creates an RPS file from the sections '.text' and '.data' of an ELF application file and saves it to the file named 'output.rps'.

Command Line Output Example

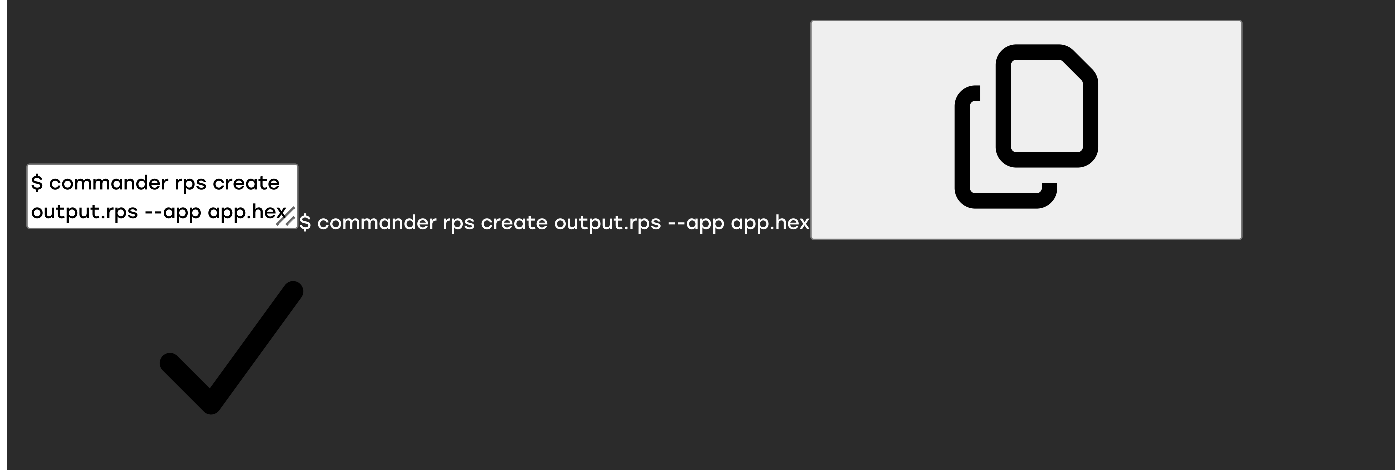
Copyright © 2026 Silicon Laboratories. All rights reserved.

## Create an RPS File from a Hex/s37 Image

You can create an RPS file from an Intel Hex (hex) image or from a Motorola S-record (s37) image.

Command Line Syntax

Command Line Input Example




This command line creates an RPS file from a hex image and saves it to the file named 'output.rps'.

Command Line Output Example

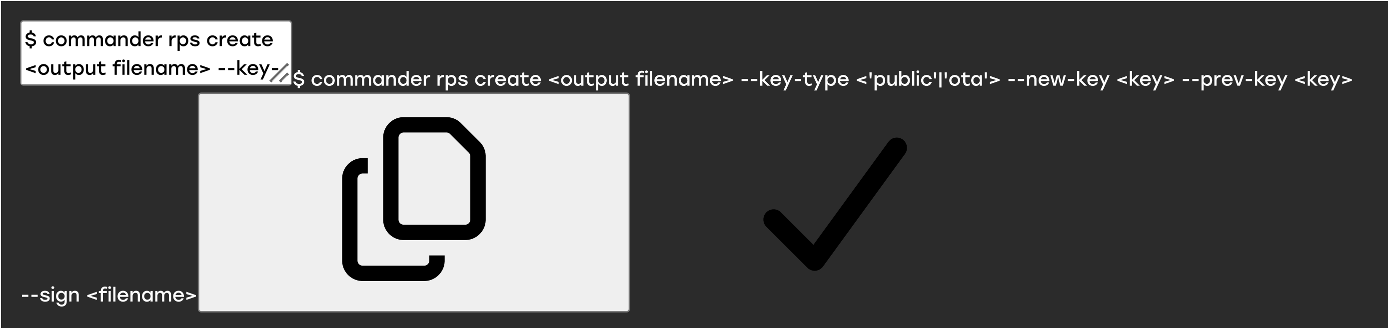
Copyright © 2026 Silicon Laboratories. All rights reserved.

## Create an RPS File For Upgrading On-Device Key

Creating an RPS key file requires a new key to store on the device, the previous (current) key stored on the device, as well as a private ECDSA key (.pem) for signing the RPS file. Only the device's M4 public key and the M4 OTA key can be upgraded, being denoted by the key types `public` and `OTA`, respectively.

Options `--new-key` and `--prev-key` support keys as plain hex-strings (e.g. '0123456789ABCDEF'), or as .h-files containing comma-separated hexadecimal values (each prefixed with '0x'). If the provided key type is `public`, the new and previous keys can also be provided as .pem-files. Alternatively, an eligible key configuration JSON file can be provided to let Commander collect the required keys automatically.

Command Line Syntax

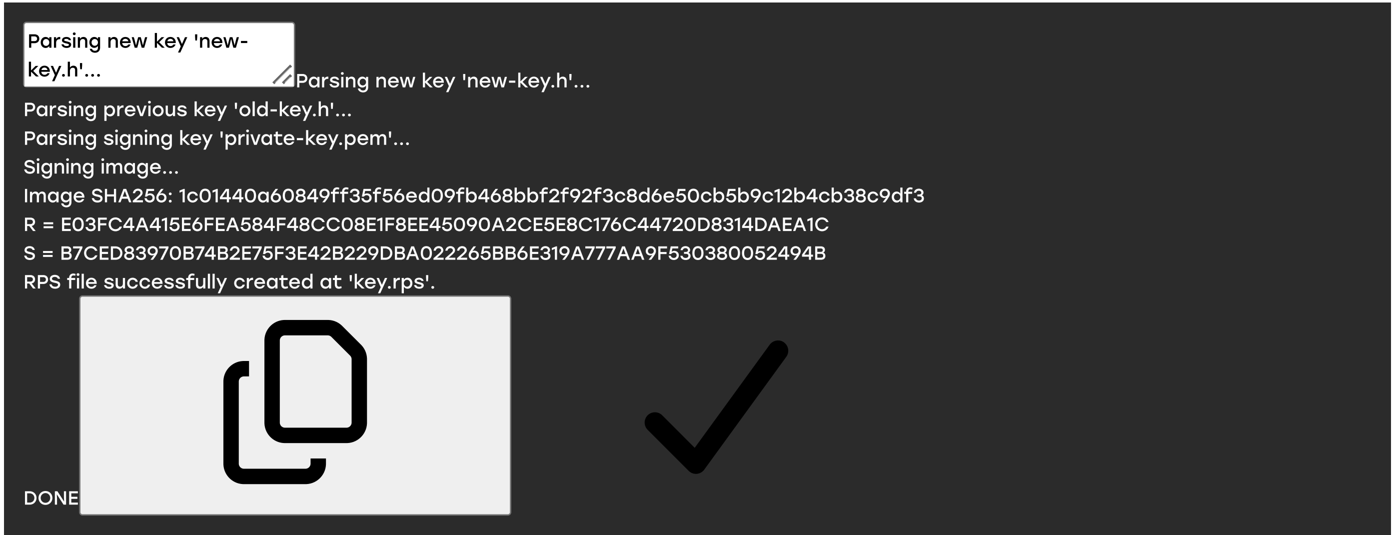



Command Line Input Example

This command line creates an RPS key file for updating the on-device M4 public key, and saves it to the file named 'key.rps'.

Command Line Output Example

Copyright © 2026 Silicon Laboratories. All rights reserved.



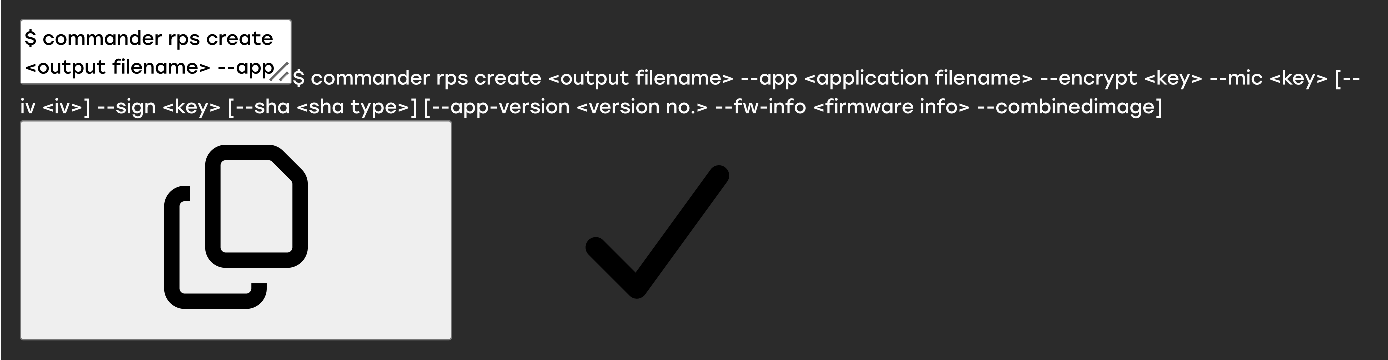
```
Parsing new key 'new-key.h'...
Parsing previous key 'old-key.h'...
Parsing signing key 'private-key.pem'...
Signing image...
Image SHA256: 1c01440a60849ff35f56ed09fb468bbf2f92f3c8d6e50cb5b9c12b4cb38c9df3
R = E03FC4A415E6FEA584F48CC08E1F8EE45090A2CE5E8C176C44720D8314DAEA1C
S = B7CED83970B74B2E75F3E42B229DBA022265BB6E319A777AA9F530380052494B
RPS file successfully created at 'key.rps'.
DONE
```

## Create a Secure RPS Application Image

RPS application images support multiple security-related features: AES-ECB-based encryption, AES-CBC MIC integrity check, and ECDSA signatures (SHA-256, SHA-384, and SHA-512). By default, these features are disabled, and a CRC-based integrity check is used on the RPS file contents.

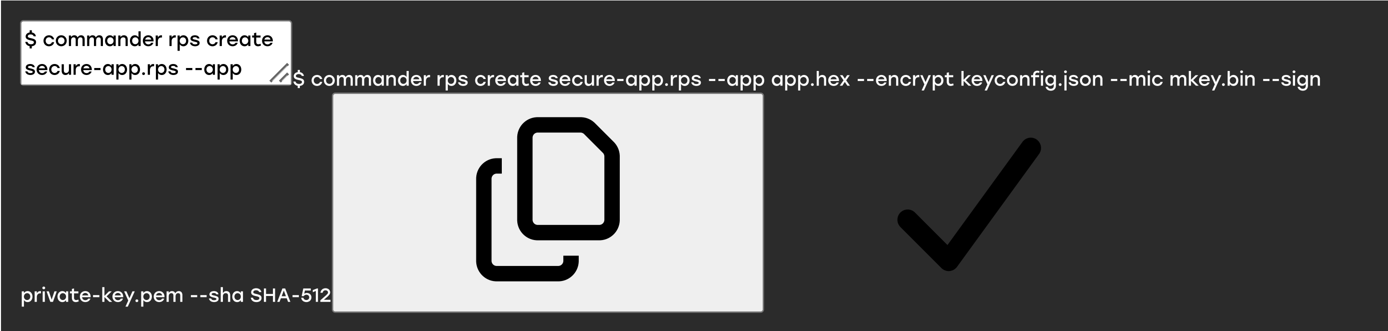
The keys for encryption and MIC are symmetric keys (32 bytes in length), and can be provided as hex strings, .bin files or as .h-files containing comma-separated hexadecimal values (each prefixed with '0x'). Alternatively, an eligible key configuration JSON file can be provided to let Commander collect the required keys automatically. If MIC integrity check is used, a custom initialization vector (IV) for the MIC algorithm may be provided as a binary file containing a 16 byte IV, using the `--iv` option.

Command Line Syntax



```
$ commander rps create <output filename> --app <application filename> --encrypt <key> --mic <key> [--iv <iv>] --sign <key> [--sha <sha type>] [--app-version <version no.> --fw-info <firmware info> --combinedimage]
```

Command Line Input Example

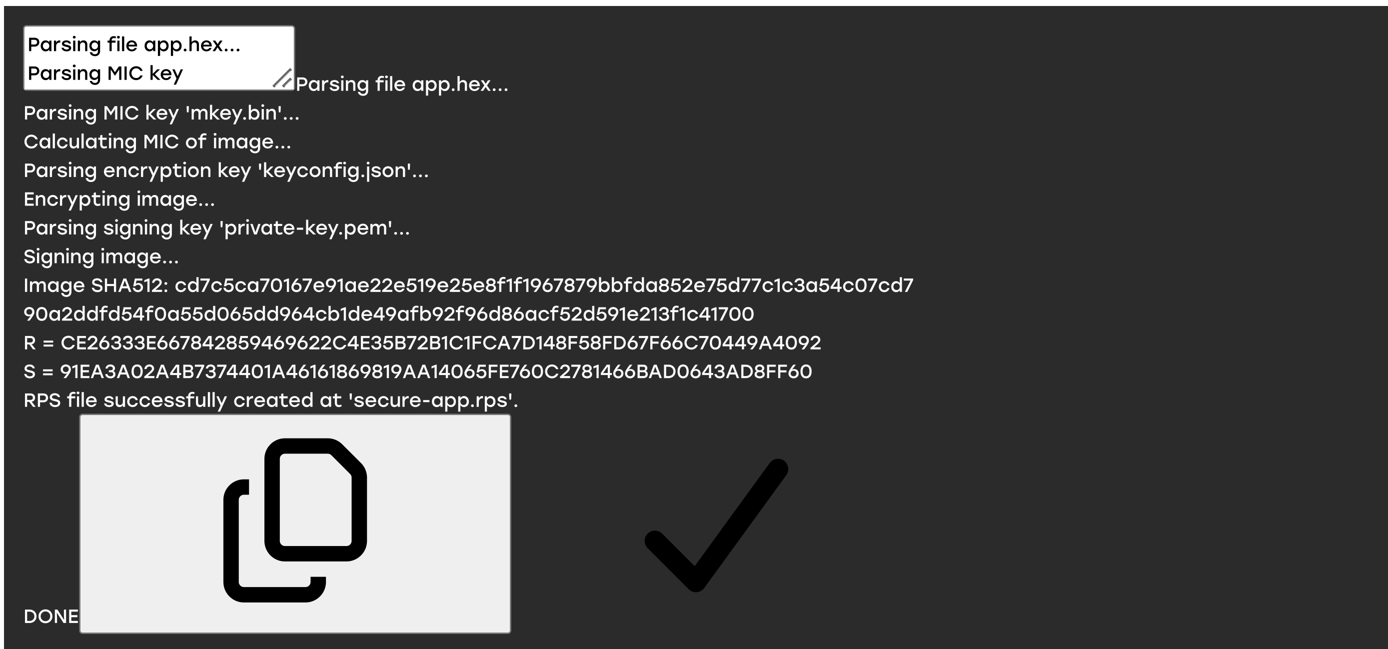


```
$ commander rps create secure-app.rps --app app.hex --encrypt keyconfig.json --mic mkey.bin --sign private-key.pem --sha SHA-512
```

Copyright © 2026 Silicon Laboratories. All rights reserved.

This command line creates a secure RPS file 'secure-app.rps' from the binary image 'app.hex', encrypted using the symmetric key in 'keyconfig.json', MIC protected using the key 'mkey.bin', and signed (SHA-512) using the private key 'private-key.pem'.

Command Line Output Example



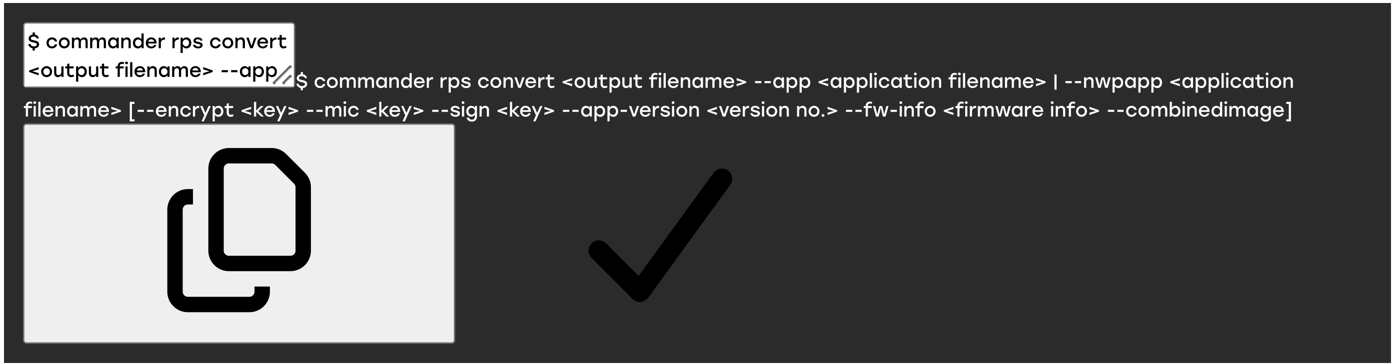
```
Parsing file app.hex...
Parsing MIC key 'mkey.bin'...
Calculating MIC of image...
Parsing encryption key 'keyconfig.json'...
Encrypting image...
Parsing signing key 'private-key.pem'...
Signing image...
Image SHA512: cd7c5ca70167e91ae22e519e25e8f1f1967879bbfda852e75d77c1c3a54c07cd7
90a2ddfd54f0a55d065dd964cb1de49afb92f96d86acf52d591e213f1c41700
R = CE26333E667842859469622C4E35B72B1C1FCA7D148F58FD67F66C70449A4092
S = 91EA3A02A4B7374401A46161869819AA14065FE760C2781466BAD0643AD8FF60
RPS file successfully created at 'secure-app.rps'.
DONE
```

## Convert an Existing RPS Application Image

Simplicity Commander can be used to convert already existing non-secure (no encryption, MIC, or signature) RPS images (both NWP and M4 images) into secure images by applying AES-ECB encryption, AES-CBC MIC integrity check, and ECDSA signatures. Non-secure images can also be modified to support combining with other RPS images by providing the `--combinedimage` flag, which sets the `COMBINED_IMAGE` bit in the RPS header.

M4 RPS images are provided using the `--app` option, whereas TA RPS images are provided using the `--nwpapp` option.

Command Line Syntax



```
$ commander rps convert <output filename> --app <application filename> | --nwpapp <application filename> [--encrypt <key> --mic <key> --sign <key> --app-version <version no.> --fw-info <firmware info> --combinedimage]
```

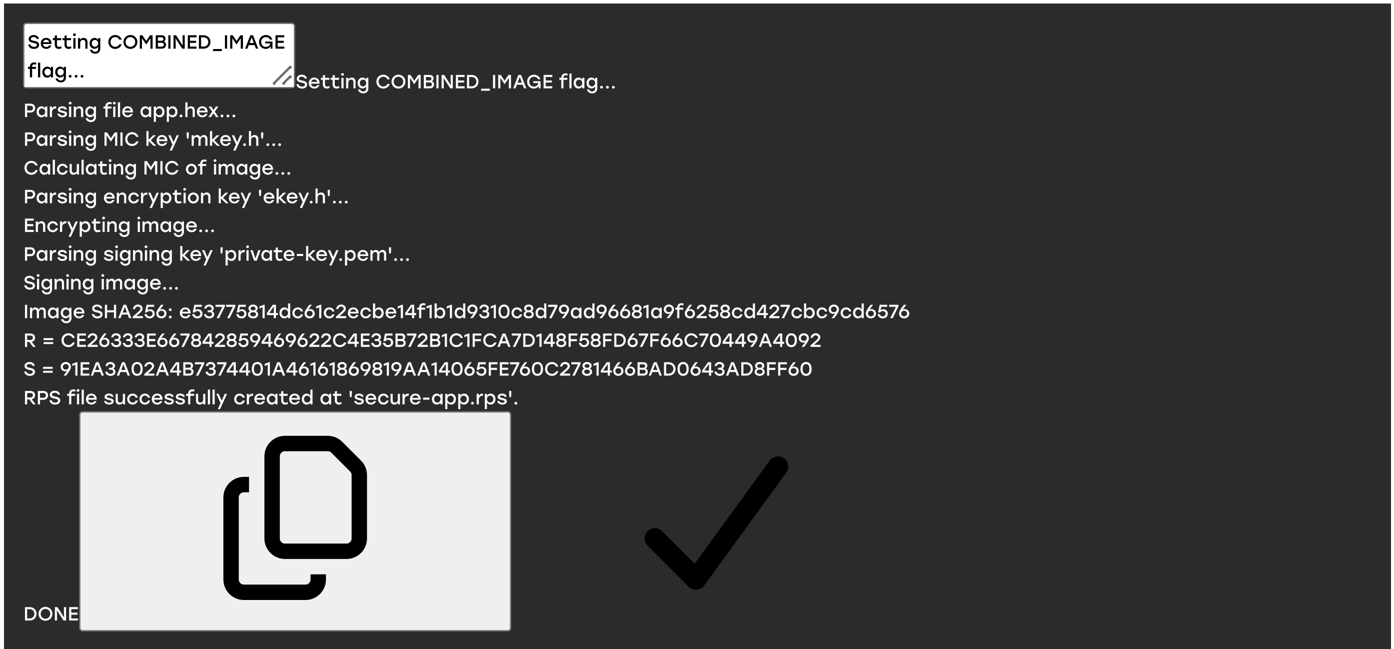
Command Line Input Example

Copyright © 2026 Silicon Laboratories. All rights reserved.

```
$ commander rps convert secure-app.rps --app app.rps --encrypt ekey.h --mic mkey.h --sign private-key.pem --app-version 0x00010209 --combinedimage
```

This command line takes the non-secure M4 RPS 'app.rps' and creates a secure RPS application image with encryption, MIC integrity check, and SHA-512 based signature, and saves it to the file named 'secure-app.rps' The command also sets a new application version number in the RPS header, and it prepares the image for combining.

Command Line Output Example



```
Setting COMBINED_IMAGE flag...
Parsing file app.hex...
Parsing MIC key 'mkey.h'...
Calculating MIC of image...
Parsing encryption key 'ekey.h'...
Encrypting image...
Parsing signing key 'private-key.pem'...
Signing image...
Image SHA256: e53775814dc61c2ecbe14f1b1d9310c8d79ad96681a9f6258cd427cbc9cd6576
R = CE26333E667842859469622C4E35B72B1C1FCA7D148F58FD67F66C70449A4092
S = 91EA3A02A4B7374401A46161869819AA14065FE760C2781466BAD0643AD8FF60
RPS file successfully created at 'secure-app.rps'.
DONE
```

## Combine Multiple RPS Images Into a Single RPS File

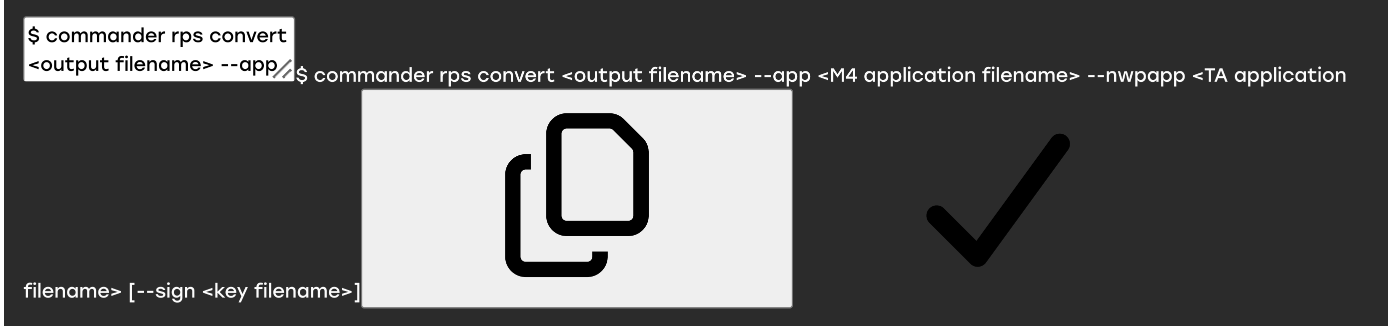
Using Simplicity Commander, you can combine an M4 RPS application image with an NWP RPS application image into a single RPS file. For an RPS image to be eligible for combining, the `COMBINED_IMAGE` bit must be set in the header of the image, either during the image's creation, or by converting an already existing non-secure RPS image.

The M4 image is provided via the `--app` option, and is always placed first within the combined image. The TA image is provided using the `--nwpapp` option.

The combined image can be signed with a private ECDSA key, provided in .pem format.

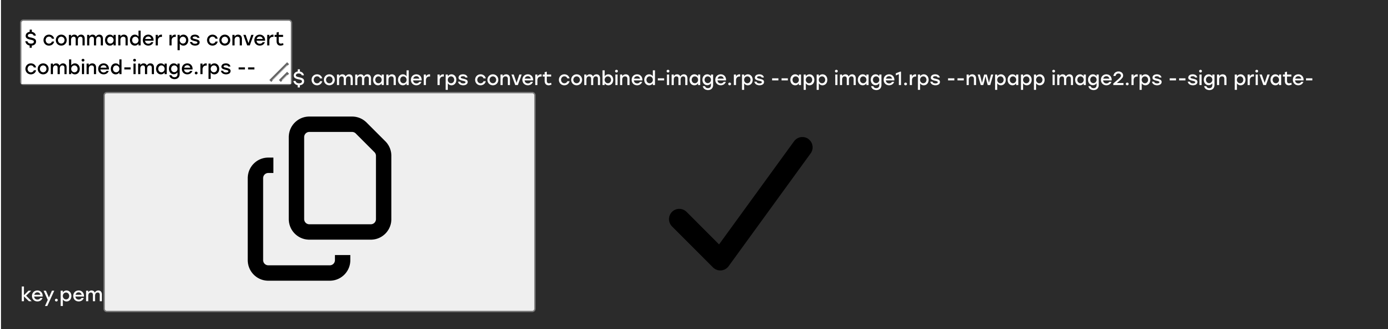
Command Line Syntax

Copyright © 2026 Silicon Laboratories. All rights reserved.



```
$ commander rps convert <output filename> --app <M4 application filename> --nwpapp <TA application filename> [--sign <key filename>]
```

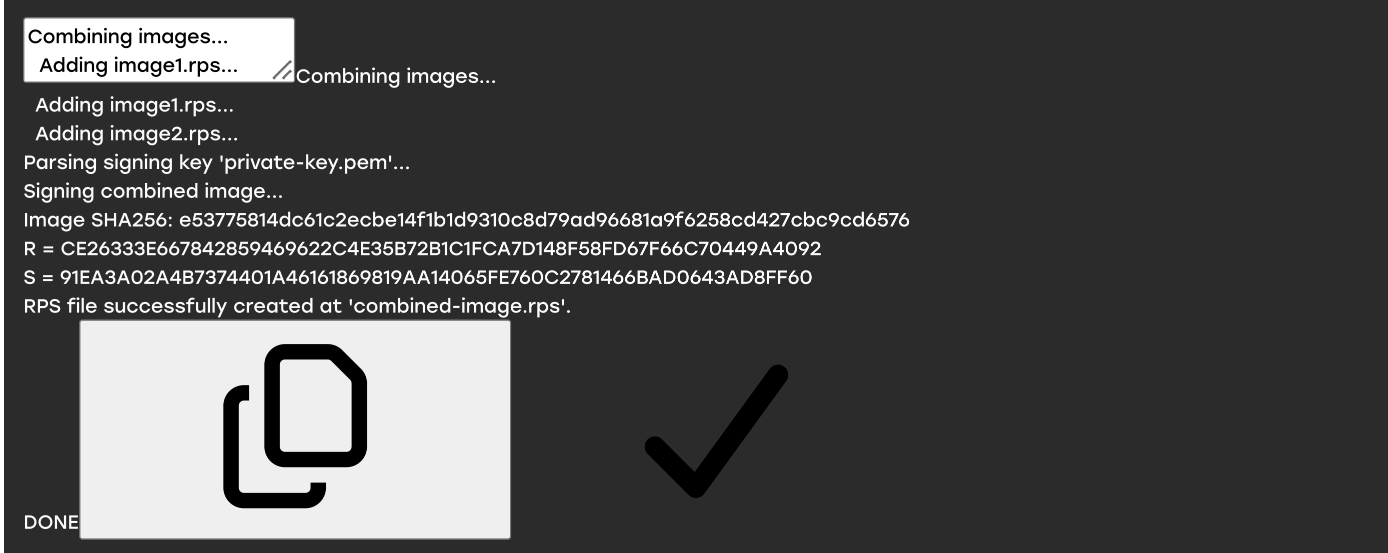
Command Line Input Example



```
$ commander rps convert combined-image.rps --app image1.rps --nwpapp image2.rps --sign private-key.pem
```

This command line takes the M4 RPS image 'image1.rps' and combines it with the TA RPS image 'image2.rps' into a single RPS image with signature.

Command Line Output Example



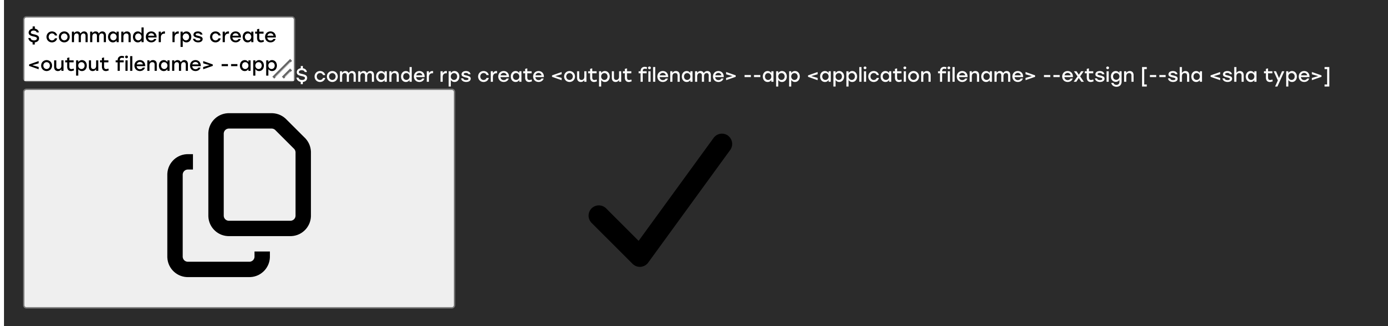
```
Combining images...
  Adding image1.rps...
  Adding image2.rps...
Parsing signing key 'private-key.pem'...
Signing combined image...
Image SHA256: e53775814dc61c2ecbe14f1b1d9310c8d79ad96681a9f6258cd427cbc9cd6576
R = CE26333E667842859469622C4E35B72B1C1FCA7D148F58FD67F66C70449A4092
S = 91EA3A02A4B7374401A46161869819AA14065FE760C2781466BAD0643AD8FF60
RPS file successfully created at 'combined-image.rps'.
DONE
```

## Create an RPS File for External Signing

Create an RPS file to be signed externally, for instance by a hardware security module (HSM), using the the `rps create` command with the `--extsign` option. The signature can be added using the `sign` command in the following section.

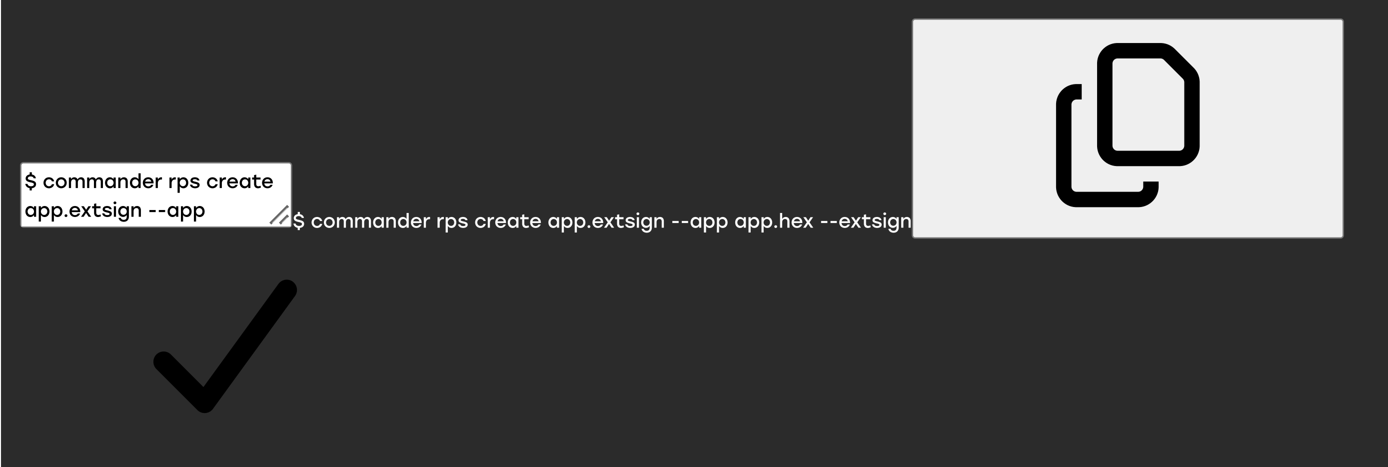
Command Line Syntax

Copyright © 2026 Silicon Laboratories. All rights reserved.



```
$ commander rps create <output filename> --app <application filename> --extsign [--sha <sha type>]
```

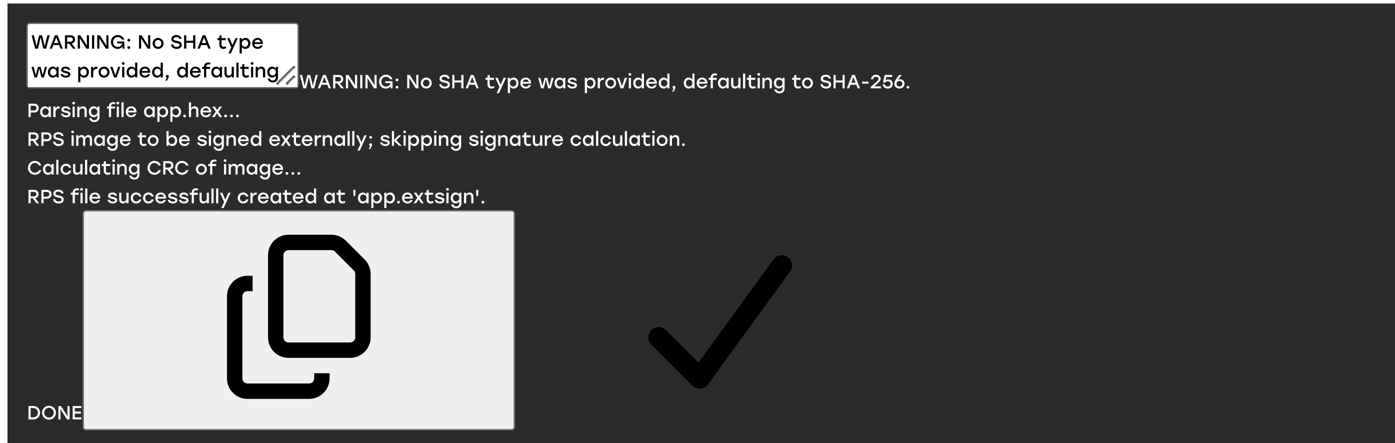
Command Line Input Example



```
$ commander rps create app.extsign --app app.hex --extsign
```

This command line creates an intermediate RPS image 'app.extsign' from the application image 'app.hex', ready for being signed by an external signer.

Command Line Output Example



```
WARNING: No SHA type was provided, defaulting to SHA-256.
Parsing file app.hex...
RPS image to be signed externally; skipping signature calculation.
Calculating CRC of image...
RPS file successfully created at 'app.extsign'.
DONE
```

## Externally Sign an RPS File

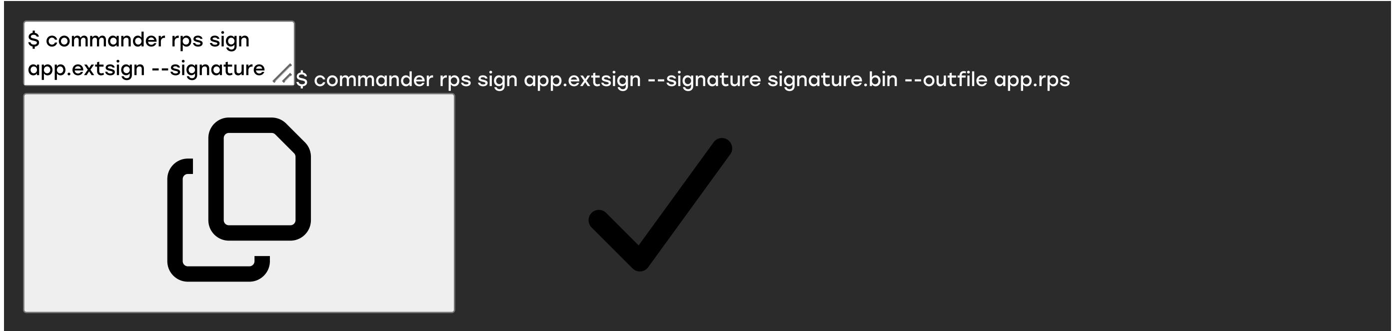
Append an externally generated signature file (binary, DER-formatted) to an RPS file created using the `rps sign` command.

> Note: The externally generated signature must be at most 72 bytes long. Shorter signatures will be padded with zeroes.

Command Line Syntax

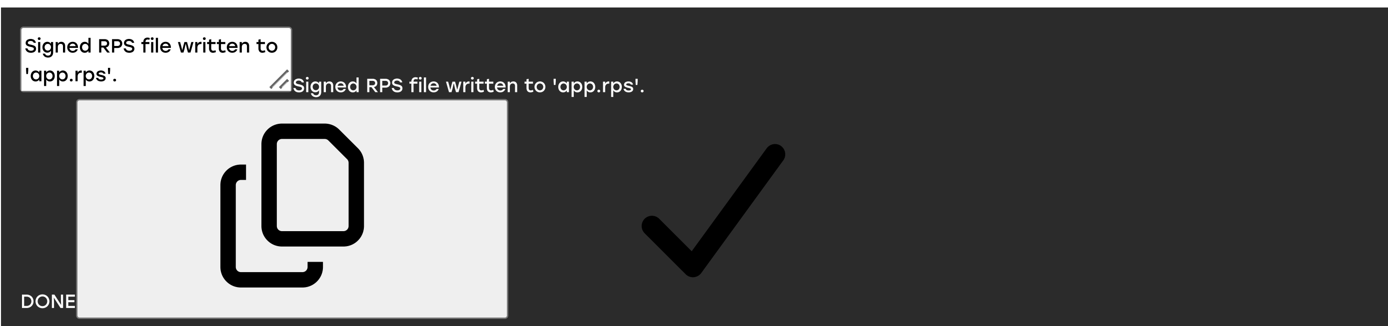
Copyright © 2026 Silicon Laboratories. All rights reserved.

Command Line Input Example




This command line appends the signature in 'signature.bin' to the intermediate RPS file 'app.extsign', writing the completed signed RPS file to 'app.rps'.

Command Line Output Example



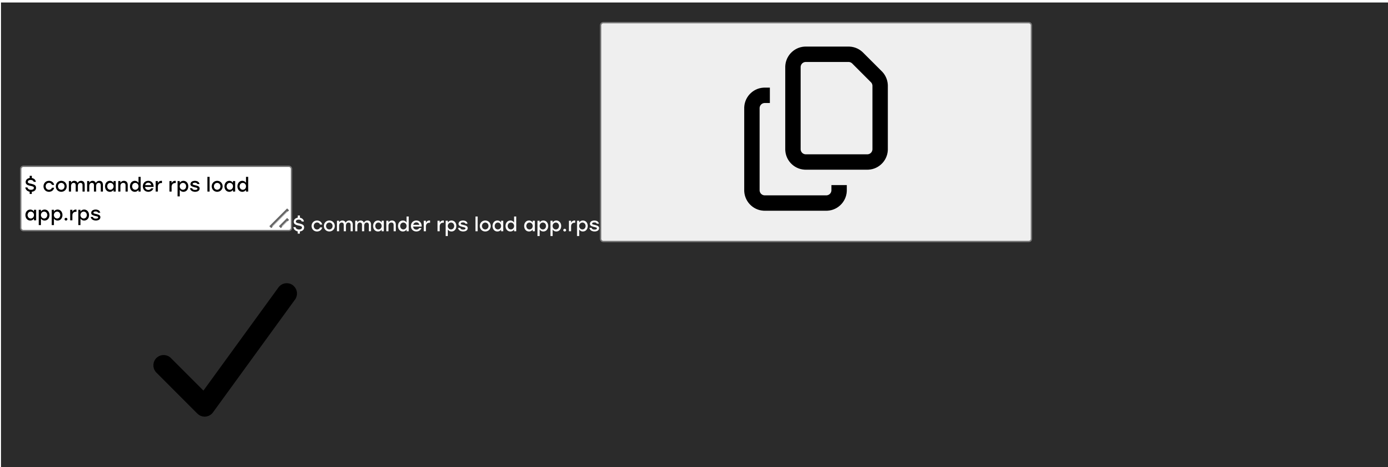

## Load RPS Image Onto Device

Simplicity Commander can load RPS images onto SiWx91x devices using the `rps load` command. Both M4 and NWP (TA) application images can be loaded using this command. If the `--eraseapp` option is used, the M4 application will be erased after the NWP firmware has been loaded.

Command Line Syntax

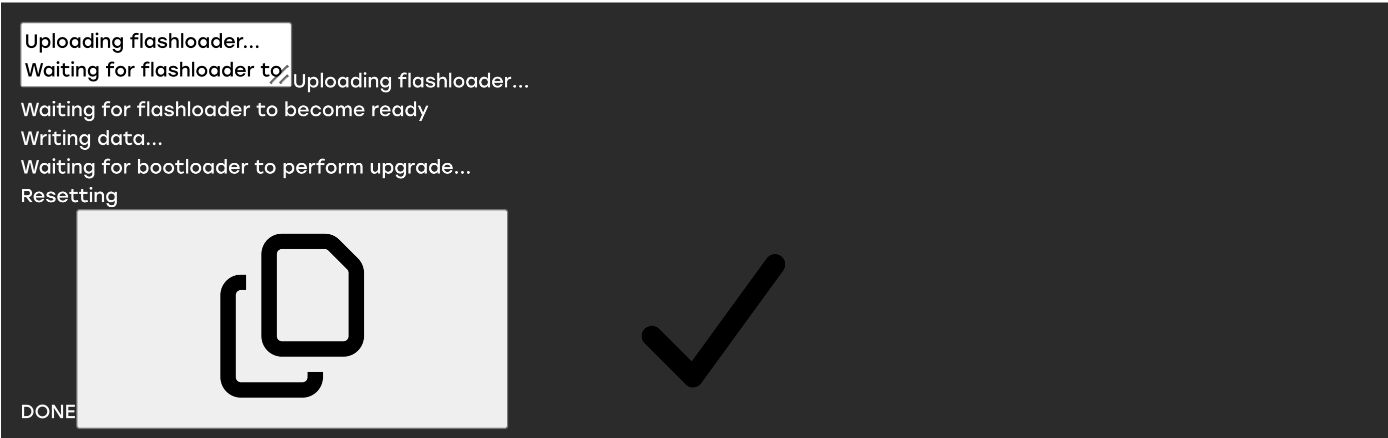
Copyright © 2026 Silicon Laboratories. All rights reserved.

Command Line Input Example




This command line takes the RPS image 'app.rps' and loads it onto the device.

Command Line Output Example



VUART Commands

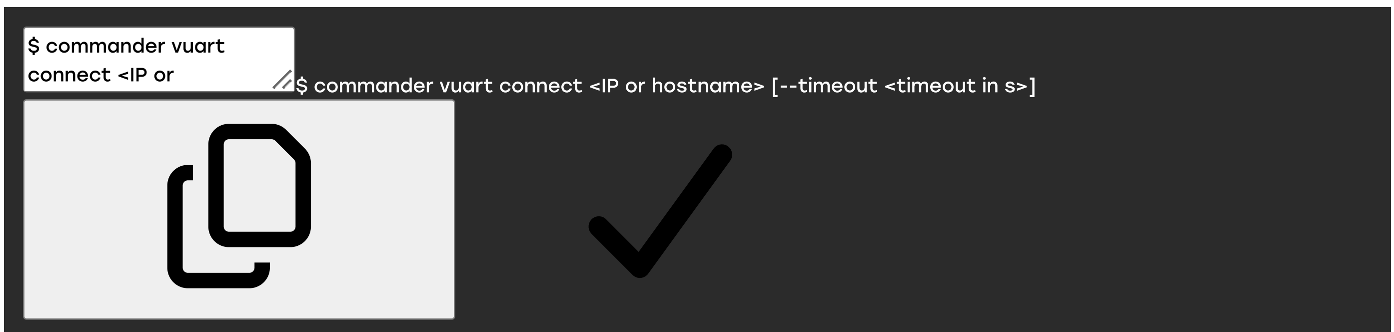
# VUART Commands

Simplicity Commander supports reading and sending data over Virtual UART (VUART) over IP using the `vuart connect` command. When the command is executed, a TCP socket is opened and connected to the hostname/IP address of the adapter, using port 4900. The command will then allow for sending and receiving data over the VUART line until termination by either pressing CTRL+C, or by meeting one of the conditions described below.

## VUART Communications Until Timeout

If the `--timeout` option is used, the command will terminate if no data is received from the target within the specified time (in seconds).

Command Line Syntax



```
$ commander vuart connect <IP or hostname> [--timeout <timeout in s>]
```

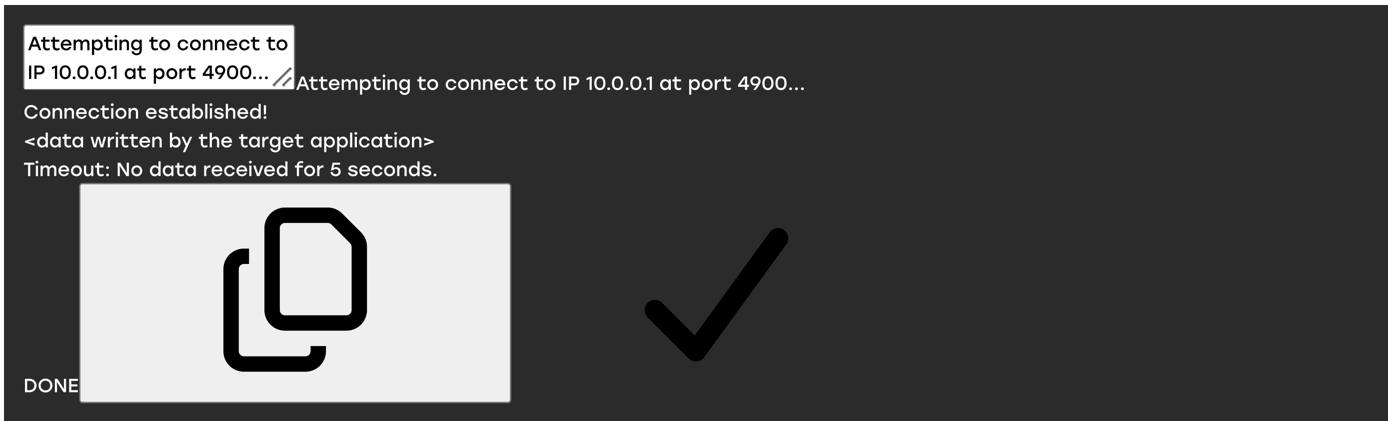
Command Line Input Example

```
$ commander vuart connect 10.0.0.1 --timeout 5
```

This command line connects to the target device via VUART and will terminate if no data is received in 5 seconds.

Command Line Output Example

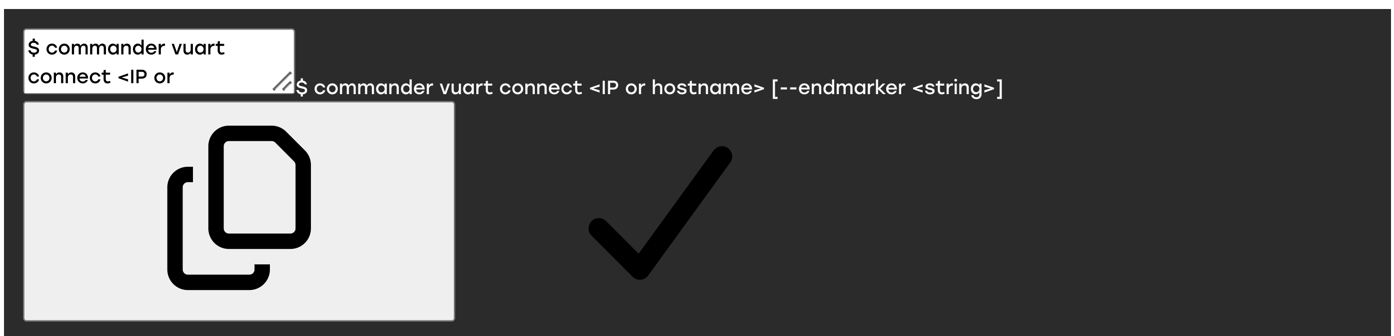
Copyright © 2026 Silicon Laboratories. All rights reserved.



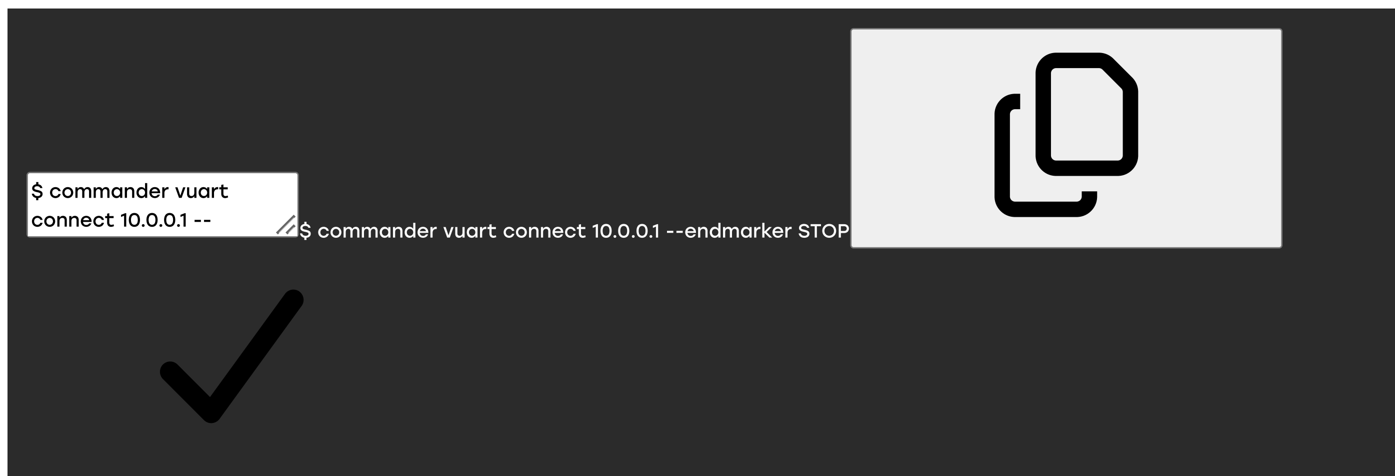

## VUART Communications Until a Marker is Found

If the `--endmarker` option is used, the command will terminate after finding the specified string in the incoming VUART data stream.

Command Line Syntax




Command Line Input Example




This command line connects to the target at IP 10.0.0.1 via VUART and terminates if the string 'STOP' is found in the data coming from the target.

Command Line Output Example

Copyright © 2026 Silicon Laboratories. All rights reserved.

Copyright © 2026 Silicon Laboratories. All rights reserved.

# RTT Commands

# RTT Commands

Simplicity Commander supports reading data from and sending data to the target via SEGGER Real Time Transfer (RTT) using the `rtt connect` command. The communications will be active until terminated by pressing CTRL+C, or if one of the conditions described below is met.

By default, the target will be reset during the initialization of the RTT connection. Providing the `--noreset` option will prevent this.

## RTT Communications Until a Marker is Found

If the `--endmarker` option is used, the command will terminate after finding the specified string in the RTT data stream.

Command Line Syntax

```
$ commander rtt connect [--endmarker <string>]
```

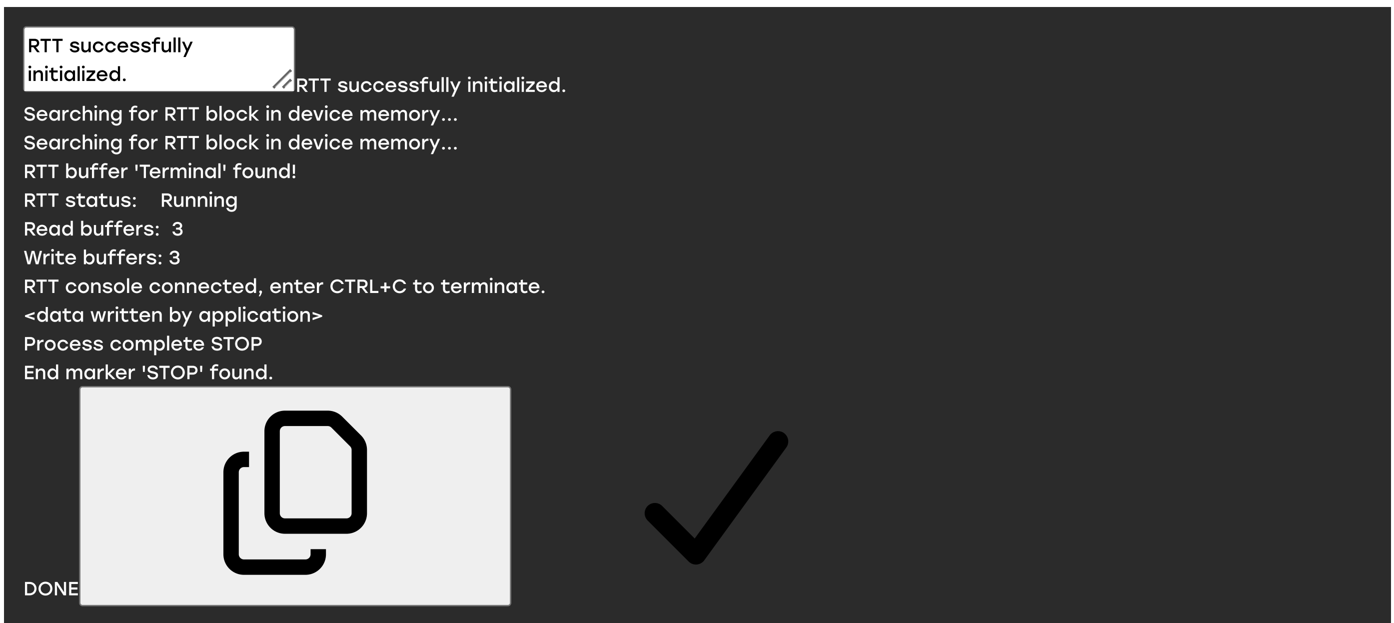
Command Line Input Example

```
$ commander rtt connect --endmarker STOP
```

This command line starts RTT communications with the target device and will terminate if the string 'STOP' is received from the target device.

Command Line Output Example

Copyright © 2026 Silicon Laboratories. All rights reserved.



```
RTT successfully initialized.
Searching for RTT block in device memory...
Searching for RTT block in device memory...
RTT buffer 'Terminal' found!
RTT status:    Running
Read buffers:  3
Write buffers: 3
RTT console connected, enter CTRL+C to terminate.
<data written by application>
Process complete STOP
End marker 'STOP' found.
DONE
```

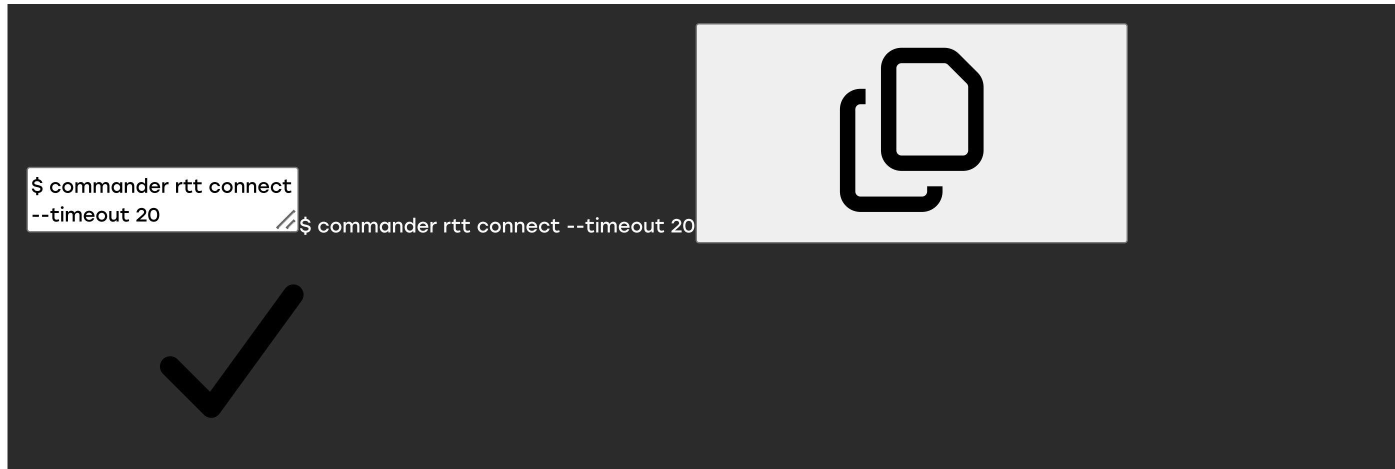
## RTT Communications Until Timeout

If the `--timeout` option is used, the command will terminate if no data is received from the target within the specified time (in seconds).

Command Line Syntax

```
$ commander rtt connect [--timeout <timeout in s>]
```

Command Line Input Example



```
$ commander rtt connect --timeout 20
```

Copyright © 2026 Silicon Laboratories. All rights reserved.

This command line starts RTT communications with the target device and will time out after 20 seconds if no more data is received.

Command Line Output Example

```
RTT successfully initialized.
Searching for RTT block in device memory...
Searching for RTT block in device memory...
RTT buffer 'Terminal' found!
RTT status:    Running
Read buffers:  3
Write buffers: 3
RTT console connected, enter CTRL+C to terminate.
<data written by application>
Timeout: No data received for 20 seconds.
DONE
```

## RTT Communications Over Virtual Terminals

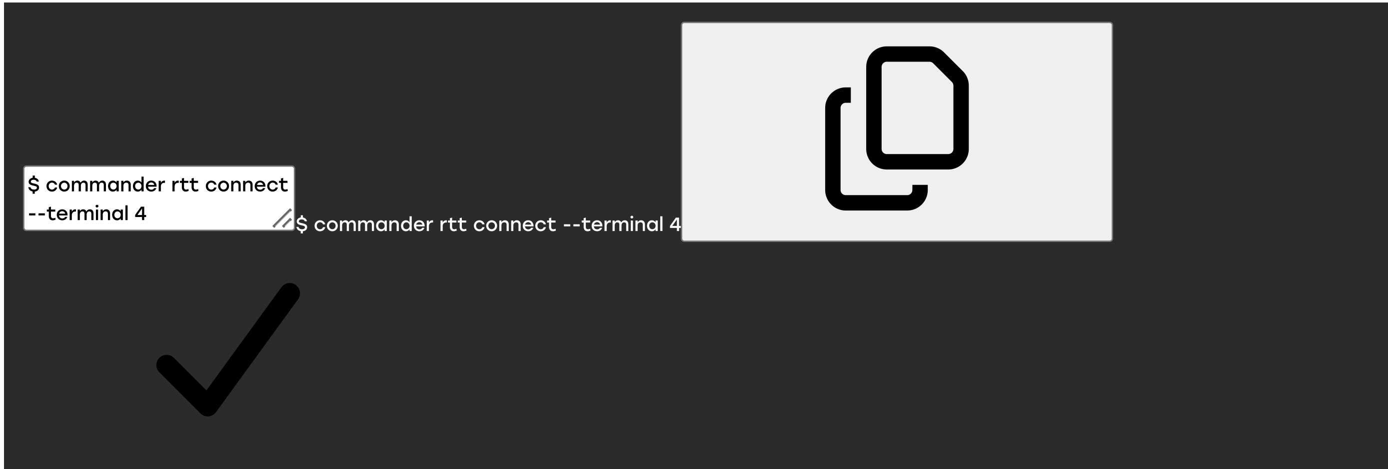
Commander supports reading data from 16 virtual RTT terminals (indexed 0-15), specified by the `--terminal` option.

The default virtual terminal used is virtual terminal 0. Virtual terminals are only supported on read channel 0 ("Terminal").

Command Line Syntax

```
$ commander rtt connect [--terminal <virtual terminal index>]
```

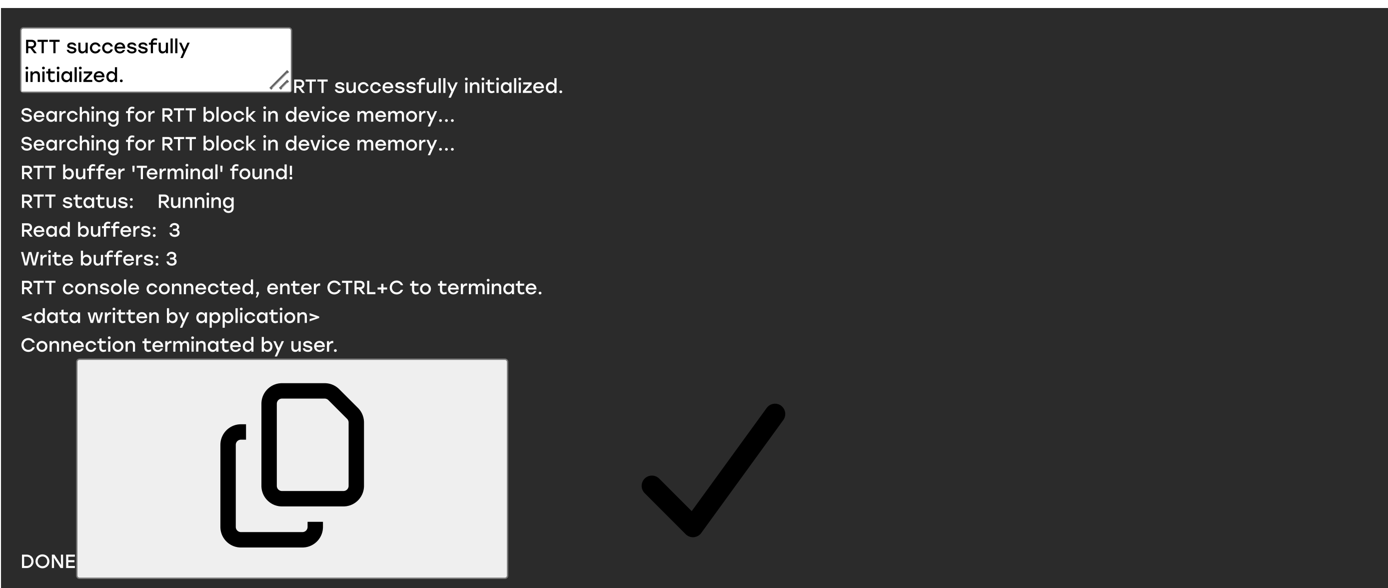
Command Line Input Example



```
$ commander rtt connect --terminal 4
```

This command line starts RTT communications with the target device and listens to RTT virtual terminal 4.

Command Line Output Example



```
RTT successfully initialized.
Searching for RTT block in device memory...
Searching for RTT block in device memory...
RTT buffer 'Terminal' found!
RTT status:    Running
Read buffers:  3
Write buffers: 3
RTT console connected, enter CTRL+C to terminate.
<data written by application>
Connection terminated by user.
DONE
```

## RTT Communications With a Custom RTT Buffer Configuration

By default Commander will try to locate the RTT block automatically. However, the RTT block address may be specified explicitly by providing the `--blockaddress` option. The default read buffer (RTT up-buffer) and write buffer (RTT down-buffer) indices default to 0, but may be set by providing the `--readbuffer` and `--writebuffer` options, respectively. Buffers may also be specified by name.

> Note: Commander will look for the specified RTT read buffer during the RTT block search. If this buffer was not initialized by the target application before the search was started, the RTT block search may fail.

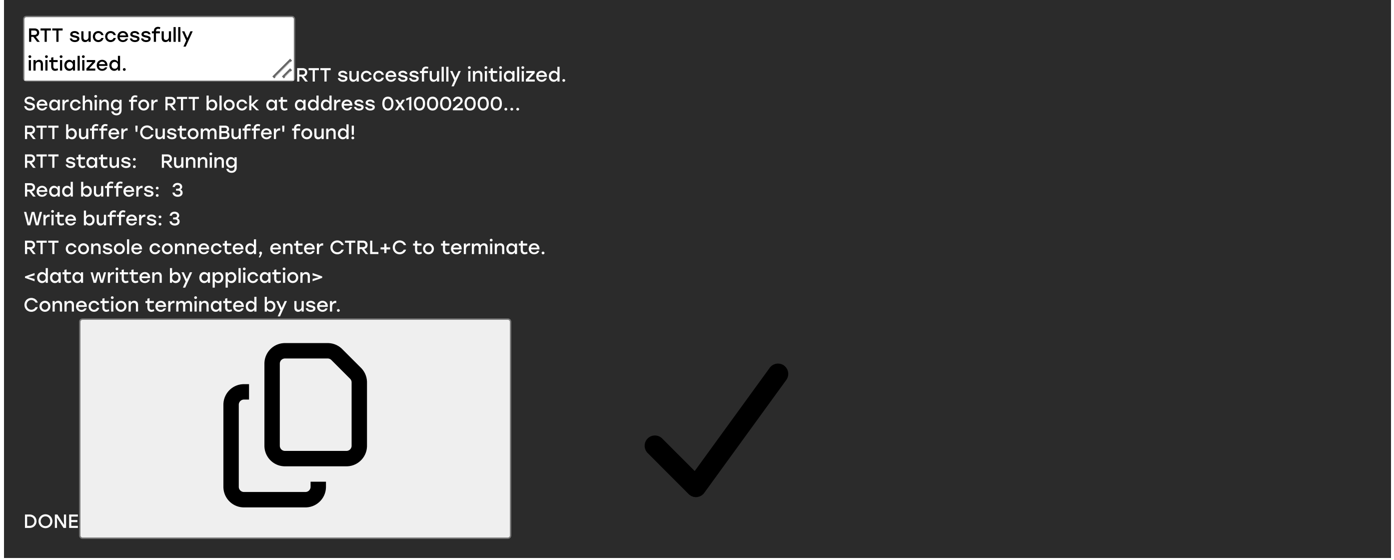
Command Line Syntax

Copyright © 2026 Silicon Laboratories. All rights reserved.

Command Line Input Example

This command line starts RTT communications with the target device and looks for RTT read buffer "CustomBuffer" in the RTT block located at address 0x10002000 in the device memory. After the connection is established, data will be read from the "CustomBuffer" RTT up-buffer, and data will be written to the target via the RTT down-buffer of index 2.

Command Line Output Example




Copyright © 2026 Silicon Laboratories. All rights reserved.

# Serial Commands

# Serial Commands

Simplicity Commander can be used to transfer files to SiWx917 devices over the adapter's serial (VCOM) port, using the Embedded Kermit protocol. These files include M4 or NWP application images, as well as tokens for unlocking debug access to either device core.

All `serial` commands require a physical data connection (i.e. USB cable) between the host computer and the adapter. The serial port can be explicitly provided using the `--serialport` option; this will also bypass all J-Link specific handling of the adapter board/kit. If the J-Link serial number is provided via the `--serialno` option, the adapter's serial port is automatically inferred by Commander.

`serial` file transfers can be aborted by pressing CTRL+C. Providing `--showprogress` will display a progress bar for the ongoing file transfer.

Simplicity Commander will attempt to configure the serial communication to use the highest available speed (921600 baud), depending on the specific adapter board and the target device. If this configuration is not desired, you may provide the `--fixedspeed` option to let Commander skip this step.

> Note: Prior to running any of the `serial` commands, the target device must be booted in ISP mode. Some adapter boards support programmatically restarting their target devices in ISP mode; in these cases Commander will attempt to do so automatically.

## Load an RPS Application Over Serial

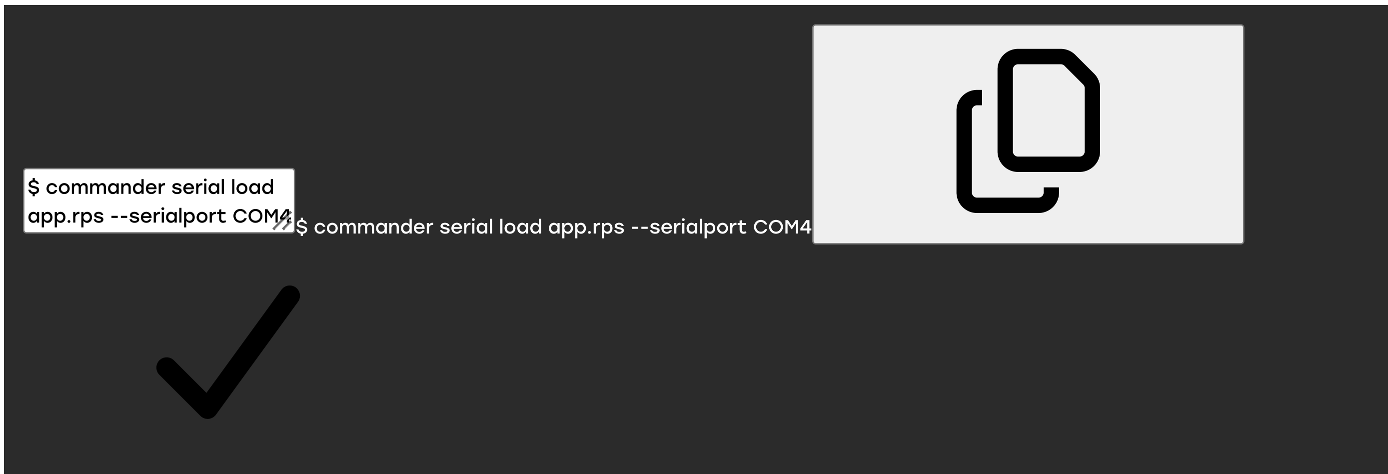
RPS images can be loaded to either the M4 or the NWP core of the SiWx917 device using the `serial load` command. The core to which the application is loaded is determined by the contents of the image's RPS header.

Command Line Syntax

```
$ commander serial load <RPS filename> [--serialport <port name> --showprogress --fixedspeed]
```

Command Line Input Example

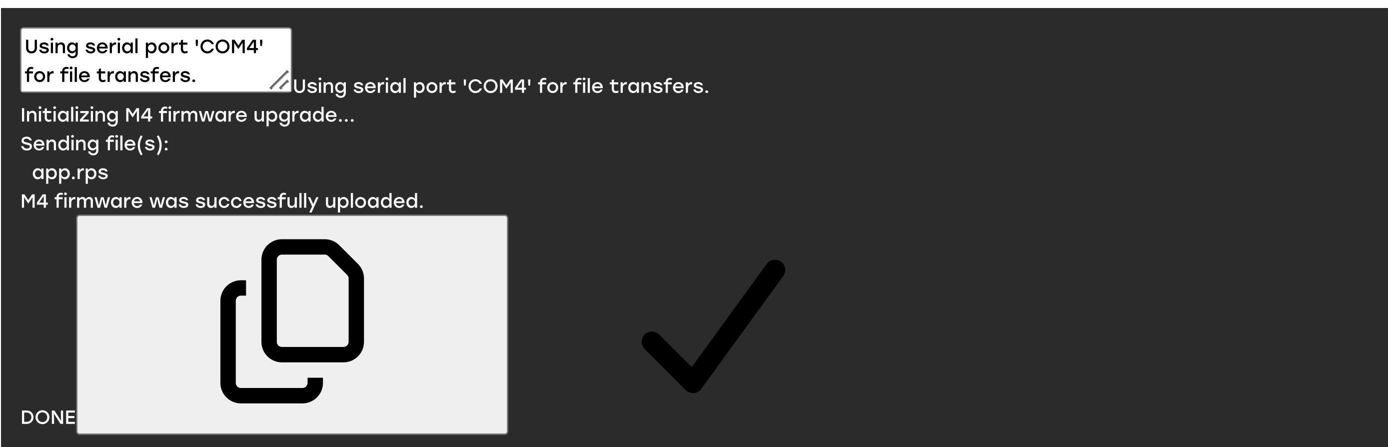
Copyright © 2026 Silicon Laboratories. All rights reserved.



```
$ commander serial load app.rps --serialport COM4
```

This command line loads the application image 'app.rps' to the device, using serial port COM4.

Command Line Output Example



```
Using serial port 'COM4' for file transfers.
Initializing M4 firmware upgrade...
Sending file(s):
  app.rps
M4 firmware was successfully uploaded.
DONE
```

## Lock Debug Access to M4/NWP Core

Simplicity Commander can lock debug access via the JTAG interface to both the M4 and the NWP(TA) core of SiWx917 devices.

Providing the `--token` option, a token can be created upon locking, which can be used for unlocking debug access to the device later. Creating this token requires a private ECDSA key provided via the `--key` option, used for signing the token.

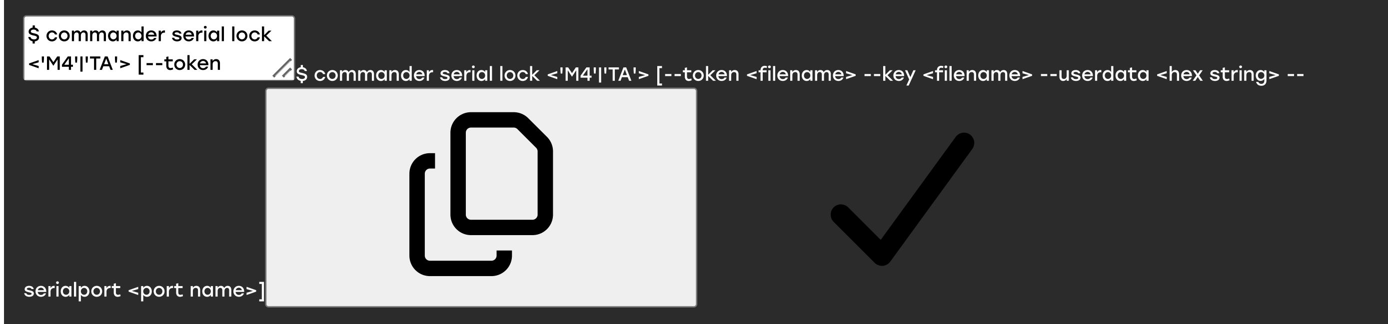
For the sake of redundancy, in case the process of saving the token file should fail, the complete token raw data is always printed to the console.

Optionally, 7 bytes (exactly) of user data can be provided using the `--userdata` option, to be stored in the token file. These bytes are provided as a hex string.

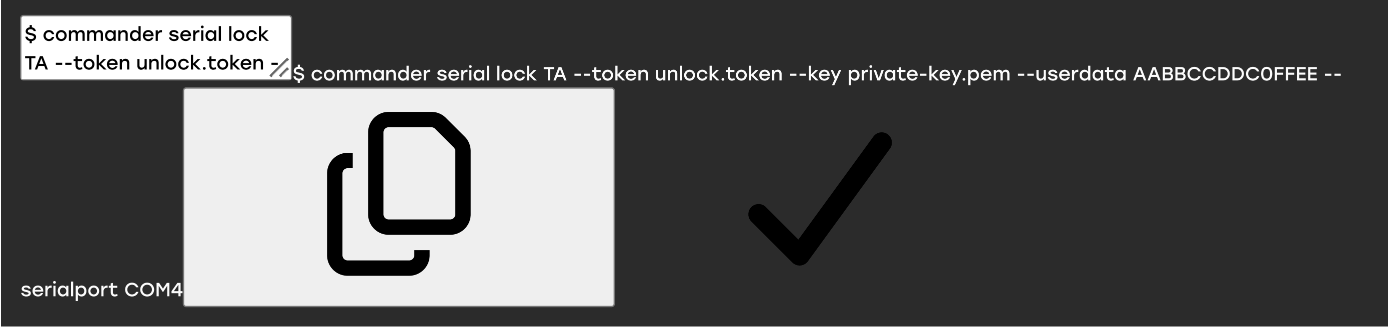
> Note: After the `serial lock` command has been run, the device needs to be power cycled for the debug access changes to take effect.

Command Line Syntax

Copyright © 2026 Silicon Laboratories. All rights reserved.

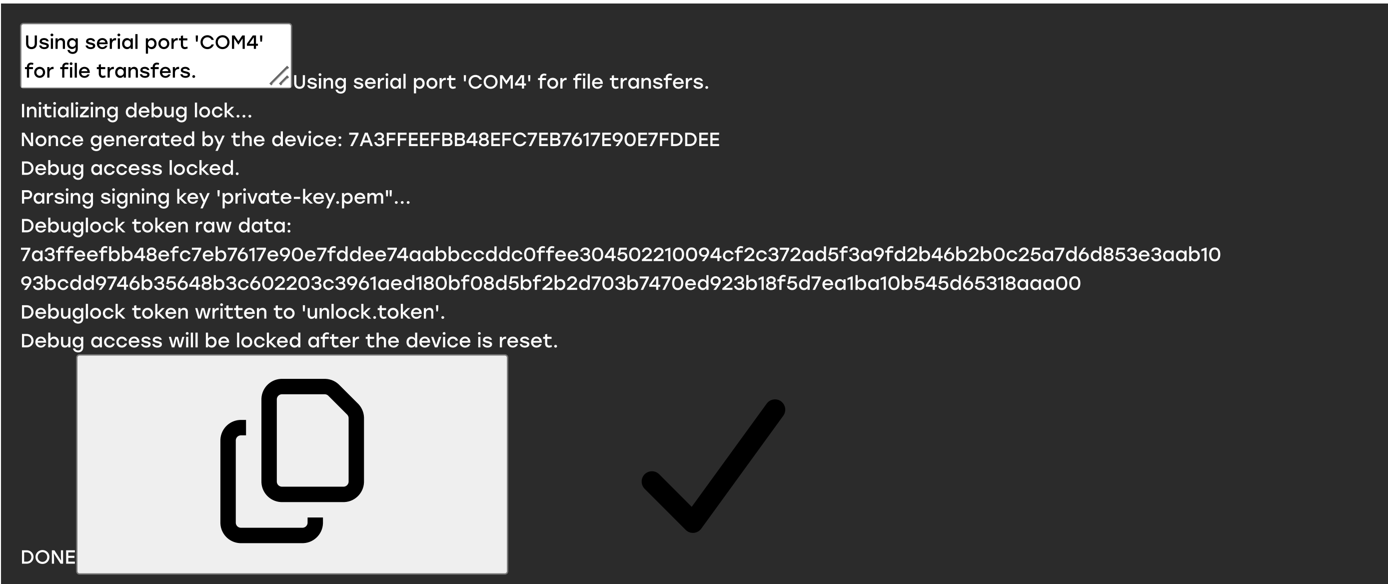



Command Line Input Example




This command line locks the JTAG debug access to the NWP core of the device, and saves the debug access unlock token to the file 'unlock.token'. The bytes `AABBCCDDC0FFEE` are stored in the user data section of the token, and the token is signed by the 'private-key.pem' ECDSA key.

Command Line Output Example



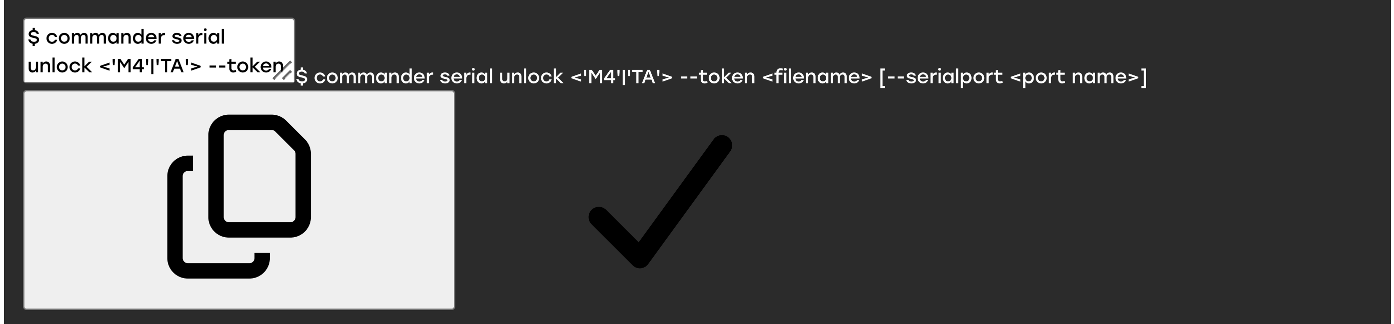

## Unlock Debug Access to M4/NWP Core With Existing Token

If the unlock token from the last time the device was locked is available, debug access to the M4/NWP(TA) core over the JTAG interface can be unlocked by using the `serial unlock` command.

Copyright © 2026 Silicon Laboratories. All rights reserved.

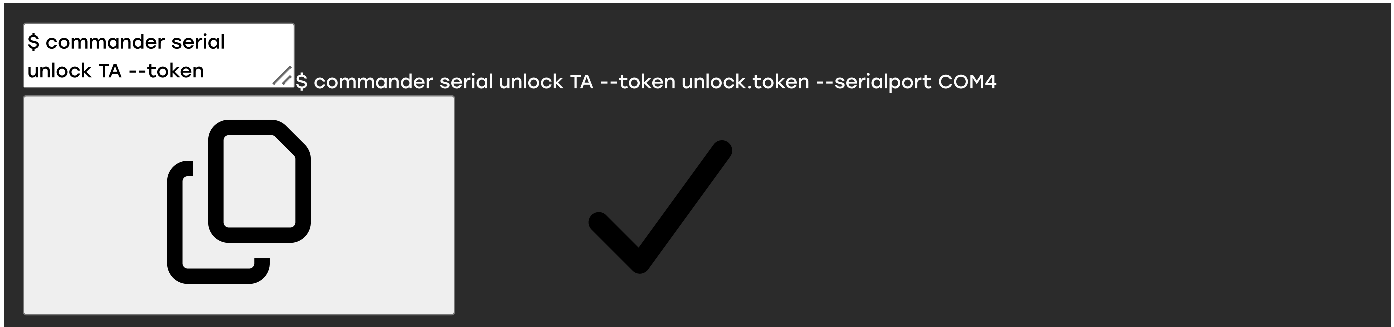
> Note: After the `serial unlock` command has been run, the device needs to be power cycled for the debug access changes to take effect.

Command Line Syntax



```
$ commander serial unlock <'M4'|'TA'> --token <filename> [--serialport <port name>]
```

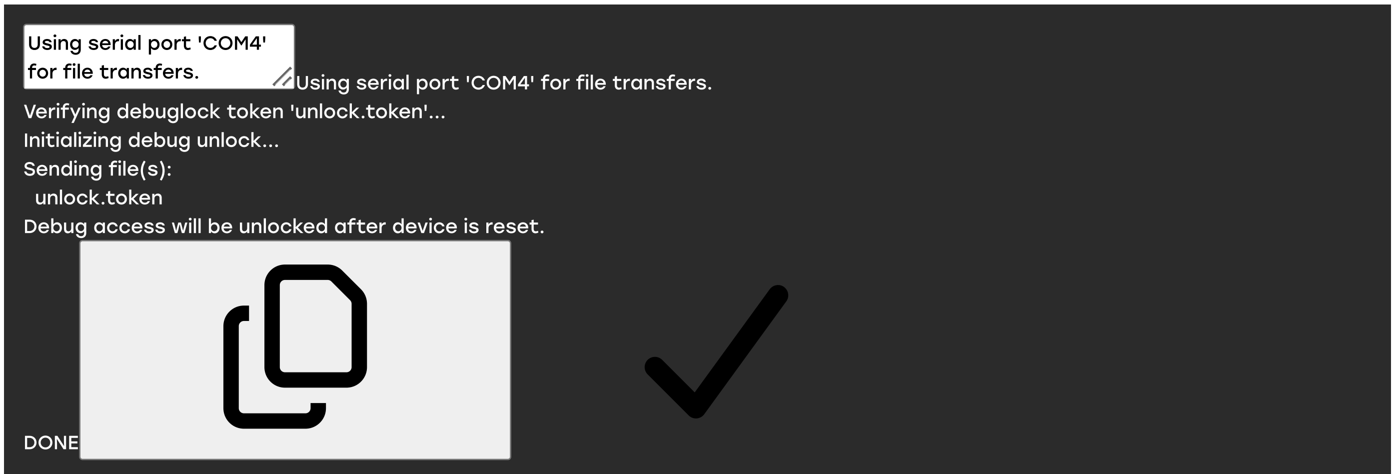
Command Line Input Example



```
$ commander serial unlock TA --token unlock.token --serialport COM4
```

This command line unlocks debug access to the NWP core, by sending the token 'unlock.token' to the device.

Command Line Output Example



```
Using serial port 'COM4' for file transfers.
Verifying debuglock token 'unlock.token'...
Initializing debug unlock...
Sending file(s):
  unlock.token
Debug access will be unlocked after device is reset.
DONE
```

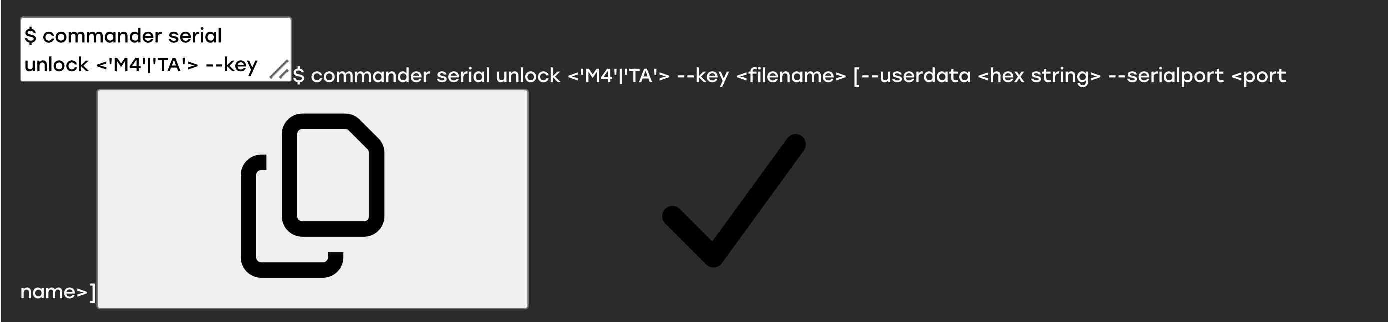
## Unlock Debug Access to M4/NWP Core Without Existing Token

Simplicity Commander can unlock the debug access to the M4/NWP(TA) core without the token from when the device was last locked. This is effectively done by locking the device (thus generating a new token), immediately followed by unlocking the device using this new, intermediate token. The `--key` option is required in this configuration, as the intermediate token needs to be signed using a private ECDSA key. The `--userdata` option is optional.

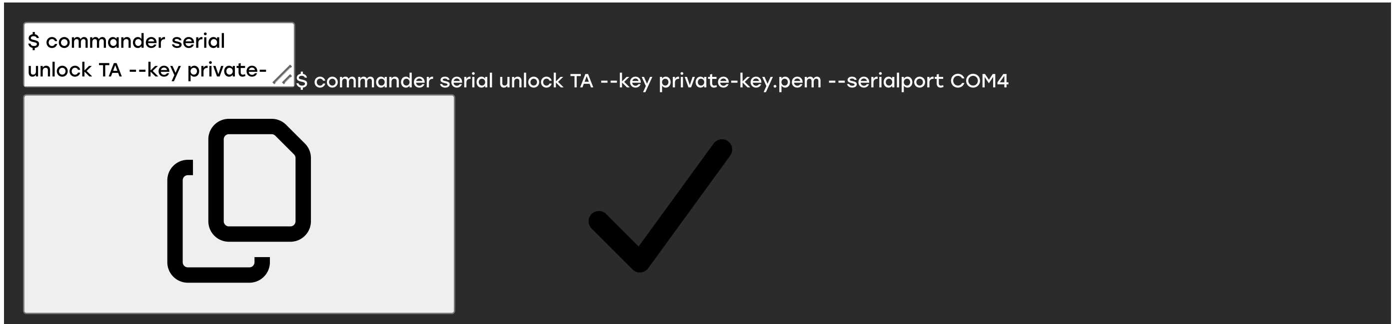
Copyright © 2026 Silicon Laboratories. All rights reserved.

> Note: After the `serial unlock` command has been run, the device needs to be power cycled for the debug access changes to take effect.

Command Line Syntax

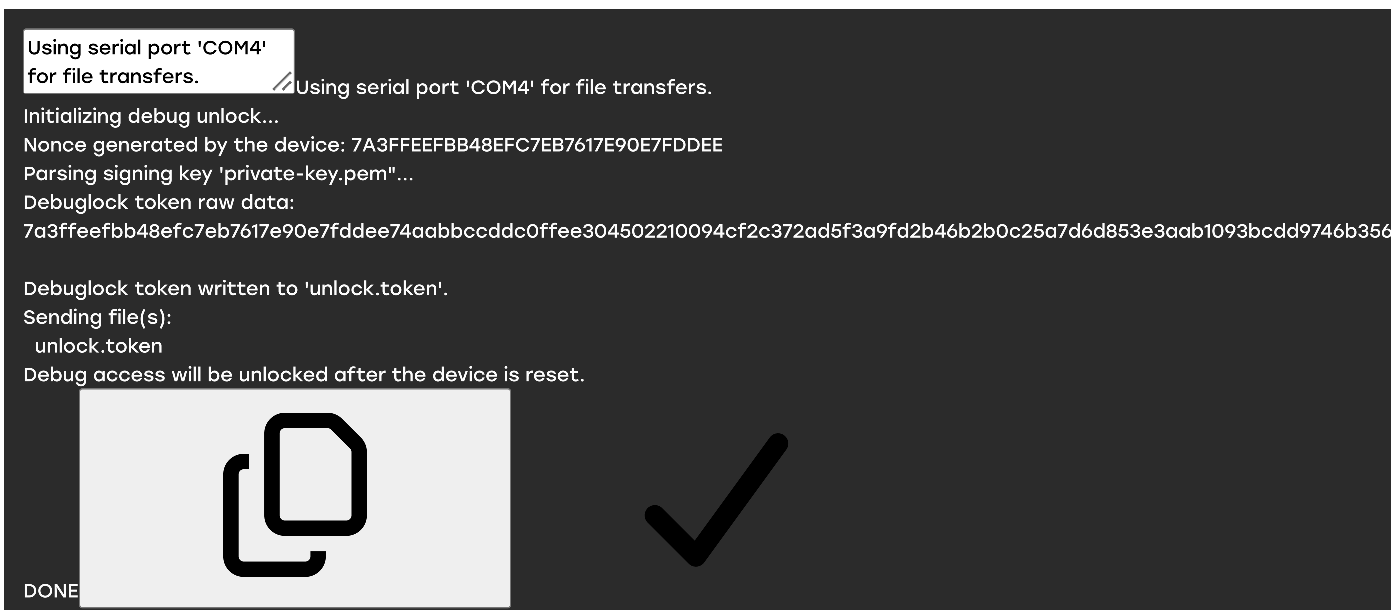



Command Line Input Example




This command line unlocks debug access to the NWP core by creating a temporary token file that is signed by the ECDSA key 'private-key.pem'.

Command Line Output Example




## Extract Device Part Number

Copyright © 2026 Silicon Laboratories. All rights reserved.

Simplicity Commander can be used to extract the device part number stored on an SiWx917.

Command Line Syntax

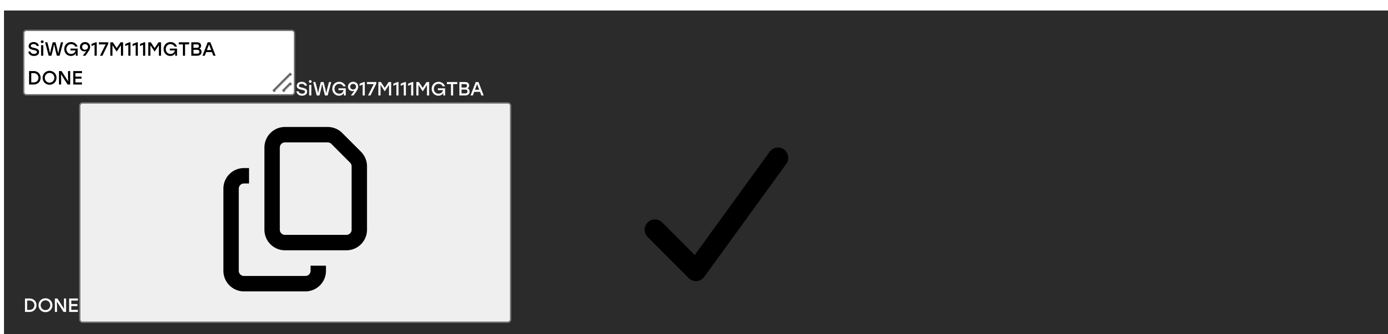
```
$ commander serial getopn
```

Command Line Input Example

```
$ commander serial getopn
```

This command line collects the device part number stored on the device.

Command Line Output Example



```
SiWG917M111MGTBA
DONE
```

Copyright © 2026 Silicon Laboratories. All rights reserved.

# Manufacturing Commands

# Manufacturing Commands

> Note: Since Simplicity Commander version 1.16.3, the `manufacturing` keyword has been changed to `mfg917`. The `manufacturing` alias is still available, but it is considered deprecated and may therefore be removed without notice in any future release of Simplicity Commander.

Simplicity Commander provides tools for provisioning SiWx91x devices with initial configuration and keys for signing/encryption. The tools are suitable for use in a manufacturing setting and can be used to write and read data to and from regions related to both M4 and NWP(TA) cores.

For writing/erasing manufacturing data, or running the key provisioning processes (including its initialization), Simplicity Commander depends on custom RAM code being loaded to the device. Loading this code can be skipped by providing the `--skipload` flag. This option must only be provided if you are completely certain that the RAM code is already loaded and running on your device, and the use of the `--skipload` flag is therefore generally discouraged.

In case an external flash configuration is used, the flash pinset index can be set using the `--pinset` option.

For configuring radio-/network co-processor (RCP/NCP) devices, Simplicity Commander communicates via a proprietary serial protocol to an interface device, which in turn communicates with the RCP/NCP device via serial peripheral interface (SPI) or secure digital input output (SDIO) commands. In this case, Simplicity Commander requires that you also provide the `--serialinterface` option along with the `--device` option. In case you are running multiple `mfg917` commands in sequence and without resetting the target device in between, you may also provide the `--skipinit` option to skip initializing the target device's SPI/SDIO interface.

> Note: `mfg917` commands are currently only supported on SiWx917 devices.

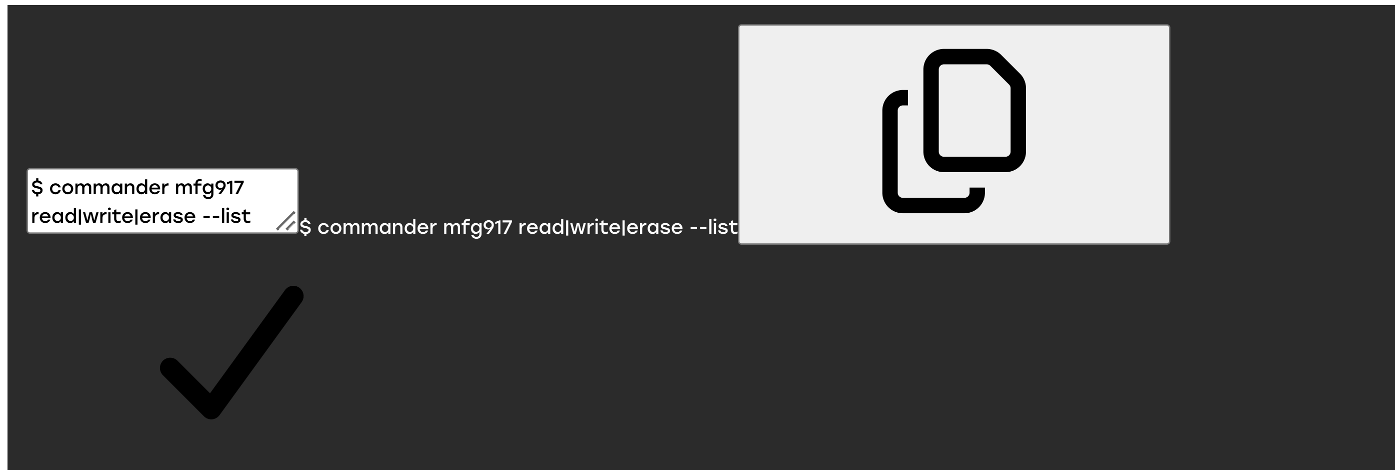
## List Available Memory Regions

Simplicity Commander can be used to read/write/erase certain memory regions, including but not limited to:

- M4 and NWP (TA) master boot records (MBR)
- Efuse
- Boot descriptors
- Keys
- User data

To see the available memory regions on your device as well as their ability for producing JSON output, provide the `--list` option with either of the commands `read`, `write`, or `erase`.

Command Line Syntax




Command Line Input Example

This command line lists all available memory regions that can be read using the `manufacturing` commands.

Command Line Output Example

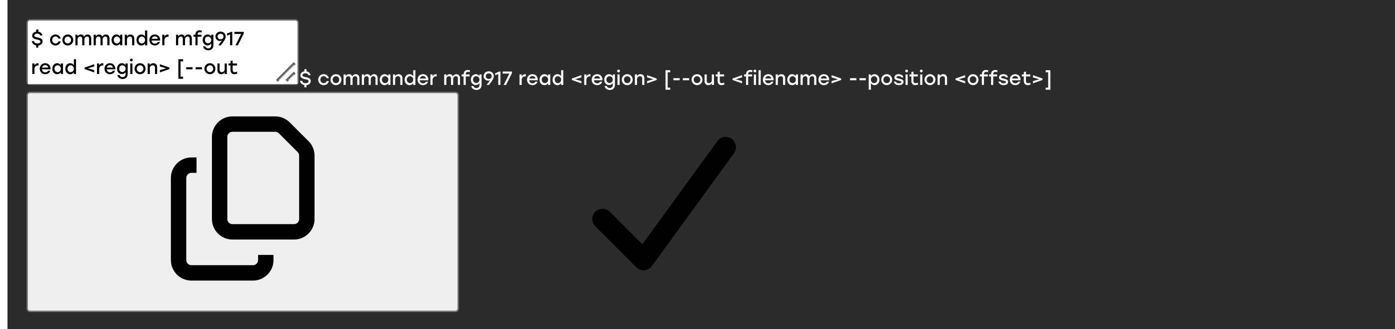
Copyright © 2026 Silicon Laboratories. All rights reserved.

```
Region name          JSON supported   Description
bfc                  No               Bootloader firmware controller
certs                No               Certificates
efuse                Yes              Efuse
efusecopy            Yes              Efuse copy
efuseipmu            Yes              Efuse intelligent power management unit
keydesctable         No               Key descriptor table
m4efusemapversioncf  No               M4 Efuse map version (common flash)
m4efusemapversiondf  No               M4 Efuse map verion (dual flash)
m4fmccf              No               M4 core flash memory controller (common flash)
m4fmcdf              No               M4 core flash memory controller (dual flash)
m4ipmucf             Yes              M4 core intelligent power management unit (common flash)
m4ipmudf             Yes              M4 core intelligent power management unit (dual flash)
m4mbrcf              Yes              M4 core master boot record (common flash)
m4mbrdf              Yes              M4 core master boot record (dual flash)
m4ptinfocf           No               M4 production information (common flash)
m4ptinfodf           No               M4 production information (dual flash)
pufactkey            No               PUF activation key
rompatch             No               ROM patches
signature            No               Signature
statickeys           No               Static keys
storeconf            No               Store configuration
tafmc                No               NWP flash memory controller
tafwimg              No               NWP firmware image
taipmu               Yes              NWP core intelligent power management unit
tambr                Yes              NWP core master boot record
userdata             No               User data
DONE
```

## Read Memory Region Data From Device

The read data can be output to the terminal or saved to a raw binary file. For supported regions, the output data can also be stored in a human-readable JSON file. If JSON output is supported and desired, provide an output filename with '.json' extension. If you want to read the region data from a certain offset relative to the start address of that region, you can provide that offset using the `--position` option.

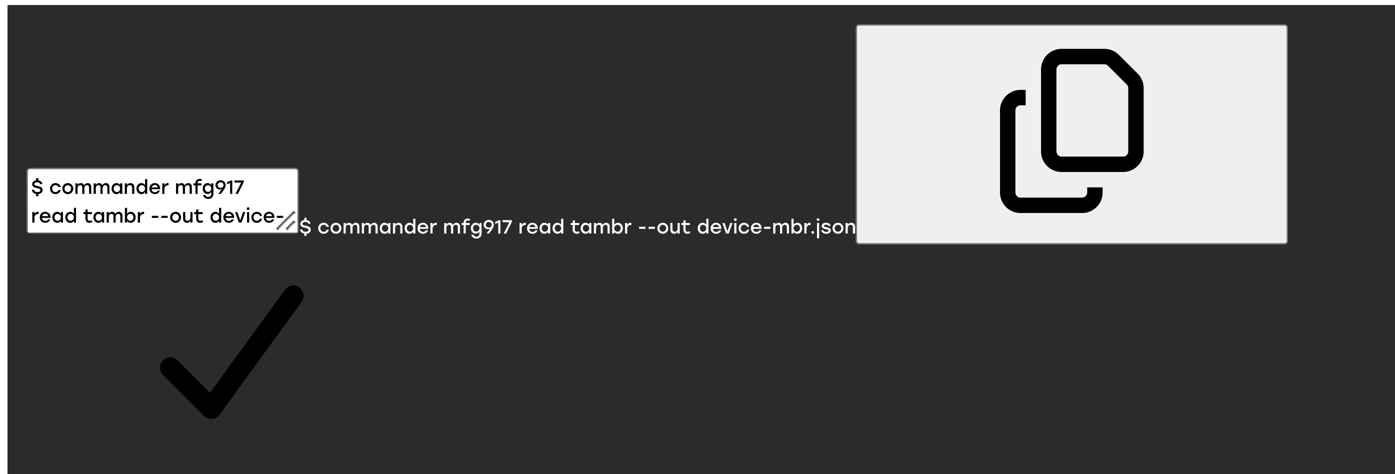
Command Line Syntax



```
$ commander mfg917 read <region> [--out <filename> --position <offset>]
```

Command Line Input Example

Copyright © 2026 Silicon Laboratories. All rights reserved.



```
$ commander mfg917 read tambr --out device-mbr.json
```

This command line reads the `tambr` region of the device and stores the data output to the file 'device-mbr.json' in JSON format.

Command Line Output Example

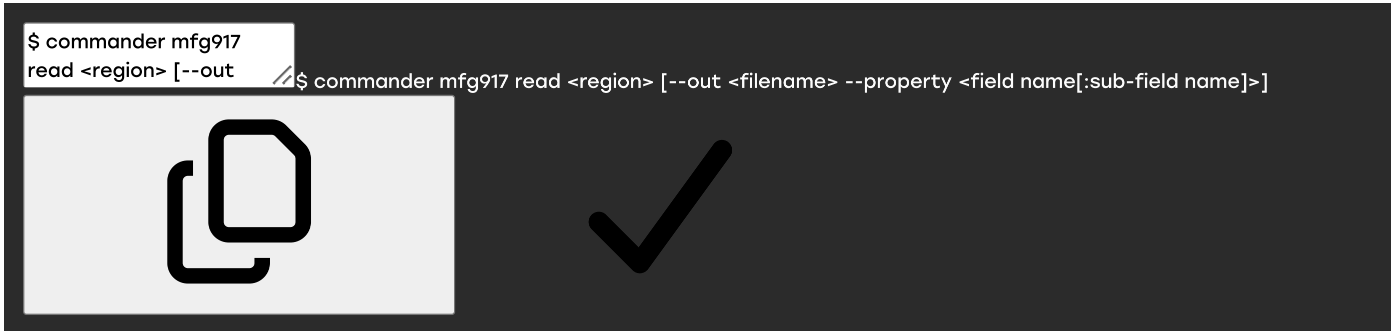
```
Reading data from region: tambr
Reading 496 bytes from 0x04000000
Writing JSON...
Manufacturing data saved to file 'device-mbr.json'
DONE
```

## Read Specific Fields From Memory Region

If you are interested in reading only certain data fields from a JSON-supported region, singular values can be extracted by providing the `--property` option along with which field and/or sub-field to read from. The `--property` option can be provided multiple times, and the fields/sub-fields are given on the format `field:sub-field`. If you provide only the field name for a data field that also contains named sub-fields, all the sub-fields will be included in the output.

If JSON output is supported and desired, provide an output filename with '.json' extension to store the selected fields in a JSON file.

Command Line Syntax



```
$ commander mfg917 read <region> [--out <filename> --property <field name[:sub-field name]>]
```

Command Line Input Example

Copyright © 2026 Silicon Laboratories. All rights reserved.

This command line reads the selected fields from the `tambr` region of the device.

Command Line Output Example

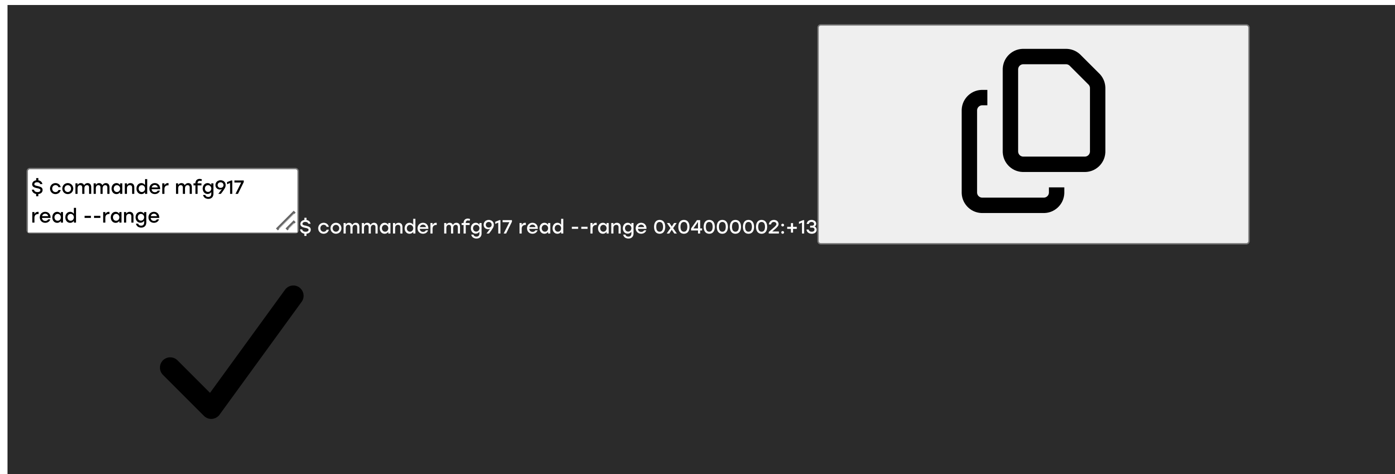
## Read Address Range From Device

Arbitrary address ranges (except any addresses inside the Efuse/OTP area) can be read from the device by omitting the region name and instead providing the `--range` option. The `--range` option accepts input as either `<startaddress:endaddress>` or as `<startaddress:+length>`. Hexadecimal values can be provided using the '0x' prefix.

Command Line Syntax

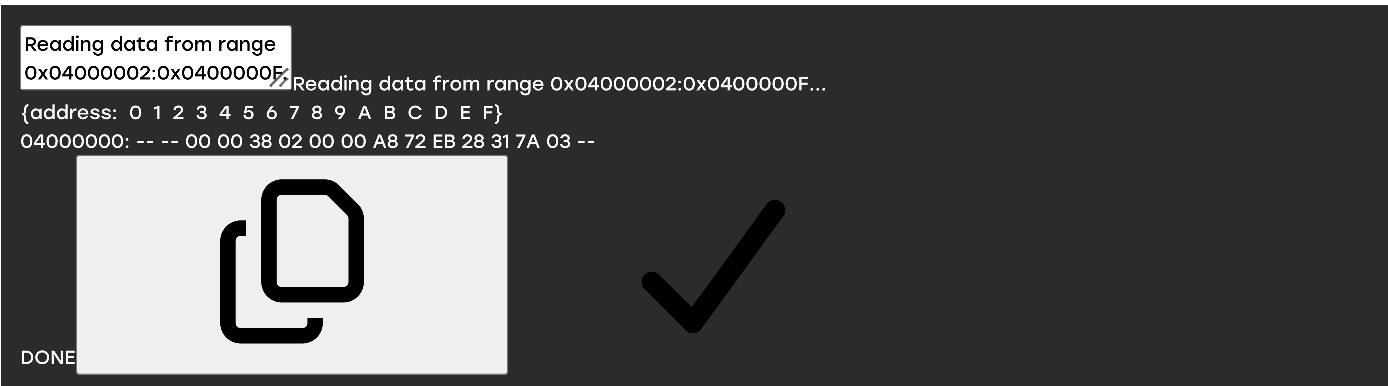
Command Line Input Example

Copyright © 2026 Silicon Laboratories. All rights reserved.




This command line reads 13 bytes from the target device from start address 0x04000002 and prints the data to the console.

Command Line Output Example




## Write Memory Region Data to Device

Manufacturing data that is to be written to a memory region of the device must be provided in either binary format (.s37 and .hex also supported), or, for eligible memory regions, as a JSON file.

> Note: Proceed with caution when writing manufacturing data to your device; writing unsupported/erroneous data/configurations may result in your device being rendered unrecoverable!

If the provided data is a binary file, the data is written to the device verbatim.

If a JSON file is provided, the region data is instead updated; the memory region is first read from the device, and the fields present in the JSON file are then used to update the corresponding region data. Lastly, the updated region data is written back to the device. If applicable, Simplicity Commander will by default also update the region's CRCs/integrity checks when changes have been made. This can be omitted by providing the `--nocrc` option.

If you want to write the data starting from a certain offset relative to the start address of that region, you can provide that offset using the `--position` option. This is only supported if you are writing binary data; not if you are providing a JSON file.

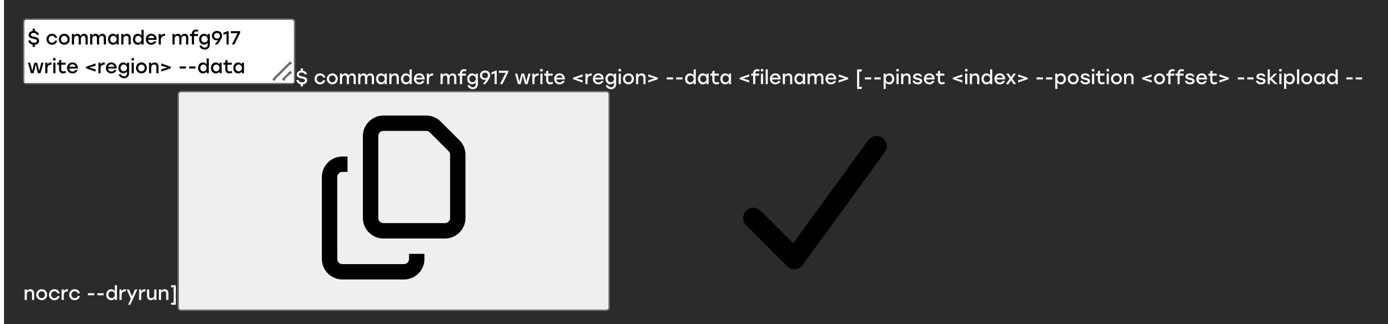
The `--dryrun` flag can be added to output the new region data to the terminal instead of writing it to the device.

This command provides no safeguards as to what data is written to the device; it is generally not recommended to write TA (NWP) MBR data to the device using this command; use the `mfg917 provision`

Copyright © 2026 Silicon Laboratories. All rights reserved.

command for that instead.

Command Line Syntax



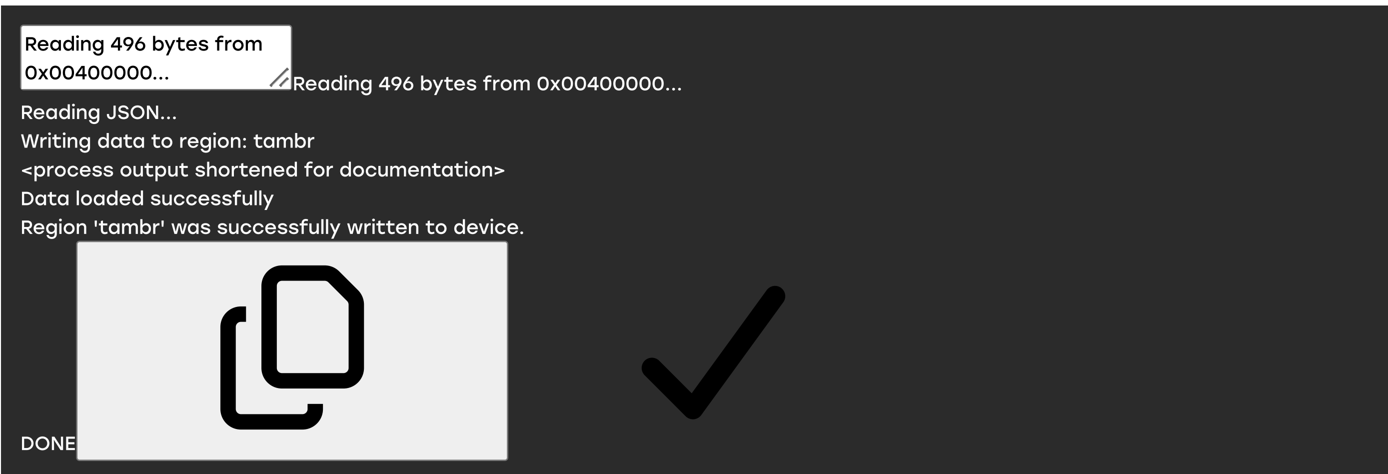
```
$ commander mfg917 write <region> --data <filename> [--pinset <index> --position <offset> --skipload --nocrc --dryrun]
```

Command Line Input Example

```
$ commander mfg917 write tambr --data mbr-updates.json
```

This command line writes an updated `tambr` to the device by first reading the `tambr` region of the device, applying the changes from the fields provided in the 'mbr-updates.json' JSON file, and finally writing this updated `tambr` data back to the device.

Command Line Output Example



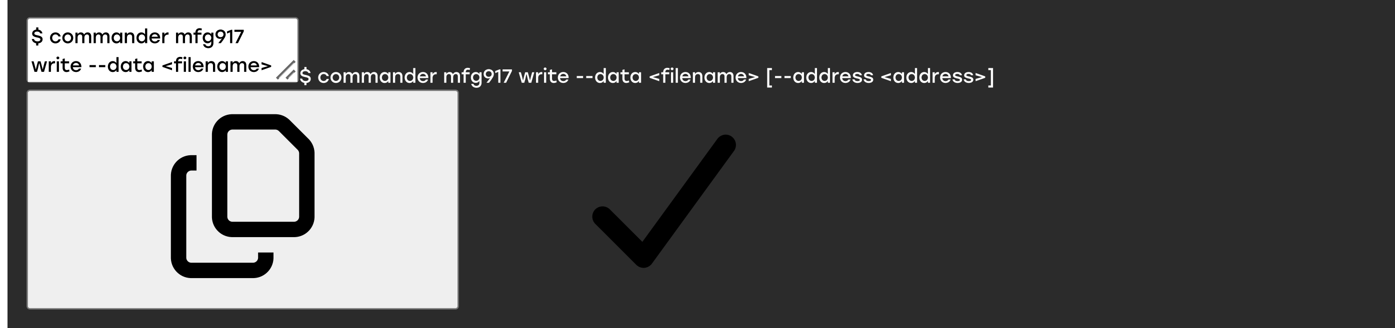
```
Reading 496 bytes from 0x00400000...
Reading JSON...
Writing data to region: tambr
<process output shortened for documentation>
Data loaded successfully
Region 'tambr' was successfully written to device.
DONE
```

## Write Data to Address

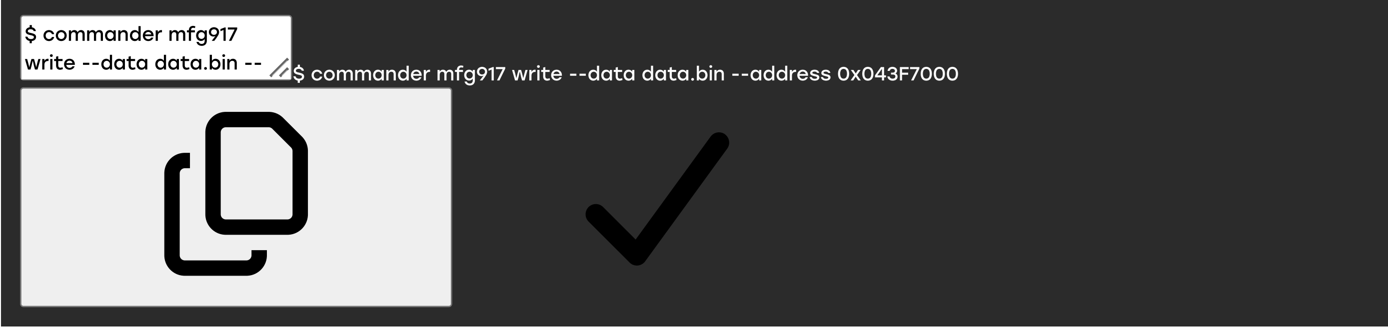
Simplicity Commander supports writing data to arbitrary memory addresses (except addresses inside Efuse/OTP areas and MBR regions) by omitting the region name and instead providing the `--address` option. If you are providing an .s37 or .hex-formatted data file, the `--address` option is ignored and Commander will instead infer the address from the address encoded in the data file.

Command Line Syntax

Copyright © 2026 Silicon Laboratories. All rights reserved.




Command Line Input Example




This command line will write the contents of 'data.bin' to address 0x043F7000.

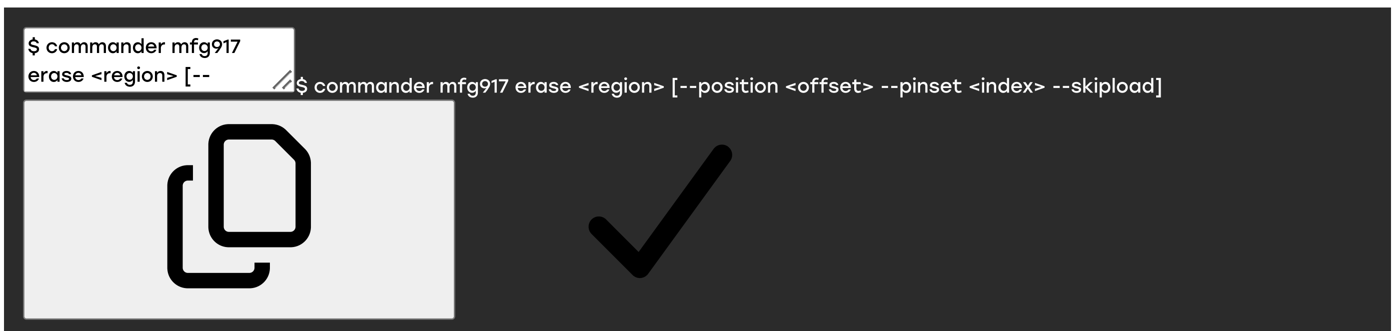
Command Line Output Example

## Erase Memory Region Data From Device

Simplicity Commander can be used to erase the data in a memory region, using the `mfg917 erase` command.

If you want to erase the region data starting from a certain offset relative to the start address of that region, you can provide that offset using the `--position` option.

Command Line Syntax



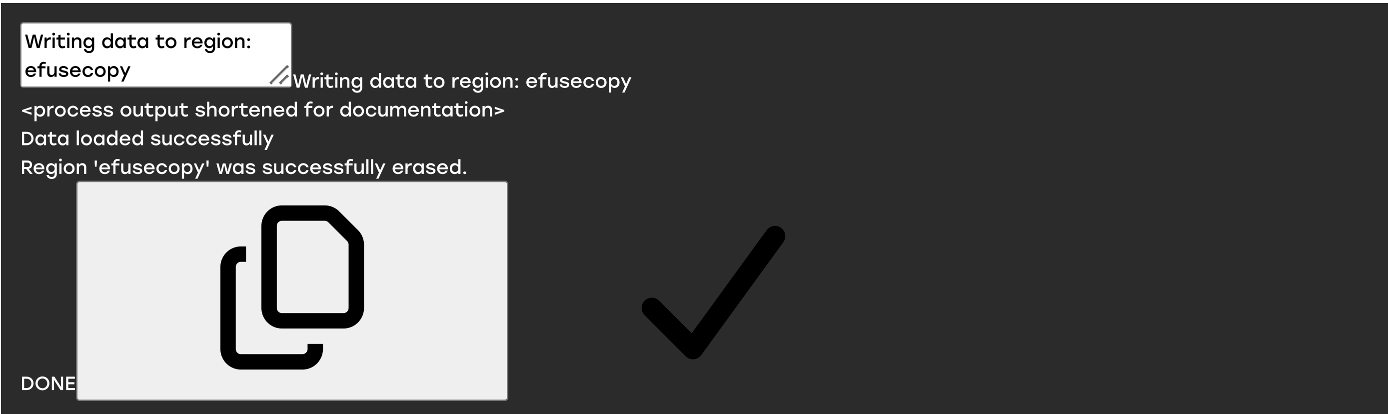

Copyright © 2026 Silicon Laboratories. All rights reserved.

Command Line Input Example

```
$ commander mfg917 erase efusecopy
```

This command line erases the contents in the `efusecopy` region of the device.

Command Line Output Example



```
Writing data to region: efusecopy
<process output shortened for documentation>
Data loaded successfully
Region 'efusecopy' was successfully erased.
DONE
```

## Erase Address Range From Device

Simplicity Commander supports erasing arbitrary memory ranges (except addresses inside Efuse/OTP areas and MBR regions) by omitting the region name and instead providing the `--range` option.

The `--range` option accepts input as either `<startaddress:endaddress>` or as `<startaddress:+length>`. Hexadecimal values can be provided using the '0x' prefix.

Command Line Syntax

```
$ commander mfg917 erase --range <startaddress>:[<endaddress> OR +<length>]
```

Command Line Input Example

Copyright © 2026 Silicon Laboratories. All rights reserved.

```
$ commander mfg917 erase --range 0x047CF000:+32
```

This command line starts erases a 32 byte range starting from address 0x047CF000.

Command Line Output Example

```
Erasing data in range 0x047CF000:0x047CF020...
<process output shortened for documentation>
Range was successfully erased.
DONE
```

## Dump Configuration Data of Device

Simplicity Commander supports dumping all data regions containing configuration data to a zip archive, using the `mfg917 dump` command.

> Note: For zip file compression functionality, the `mfg917 dump` command requires Microsoft PowerShell version 5.0 or above on Windows, and the `zip` and `unzip` system utilities on Linux/Mac

Command Line Syntax

```
$ commander mfg917 dump <zip archive filename>
```

Command Line Input Example

```
$ commander mfg917 dump device_data.zip
```

This command line reads and dumps all data regions containing configuration data into the 'device_data.zip' archive file.

Command Line Output Example

```
Reading 40960 bytes from 0x0400f000
Reading 184320 bytes from 0x047cf000
Reading 1024 bytes from 0x40012000
Reading 1024 bytes from 0x040003e0
Reading 58 bytes from 0x40012181
Reading 88 bytes from 0x04000300
Reading 496 bytes from 0x04000000
Reading 1024 bytes from 0x041c0000
Reading 496 bytes from 0x04000000
Reading 58 bytes from 0x041b0258
Reading 496 bytes from 0x04000000
Reading 496 bytes from 0x041b0000
Reading 496 bytes from 0x04000000
Reading 3 bytes from 0x041b0292
Reading 1192 bytes from 0x04002000
Reading 200 bytes from 0x04000238
Reading 72 bytes from 0x040001f0
Reading 4096 bytes from 0x04005000
Reading 3072 bytes from 0x04010400
Reading 1024 bytes from 0x04010000
Reading 58 bytes from 0x04000561
Reading 496 bytes from 0x04000000
Zip archive created at "path/to/device_data.zip".
DONE
```

## Initialize PUF And Generate Activation Code

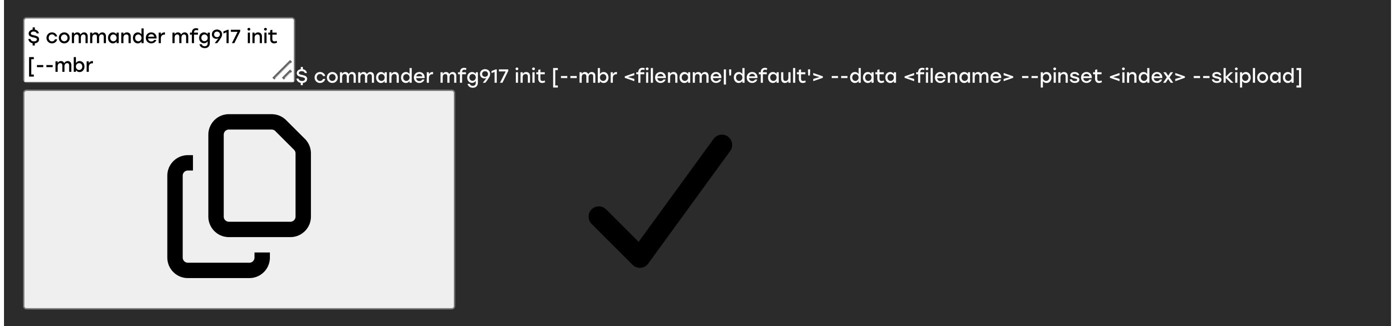
To enable security features (encryption/MIC integrity check/signing) on your SiWx917 device, the device's PUF first needs to be initialized and an activation code must be generated on the device. This can be done via the `mfg917 init` command.

Copyright © 2026 Silicon Laboratories. All rights reserved.

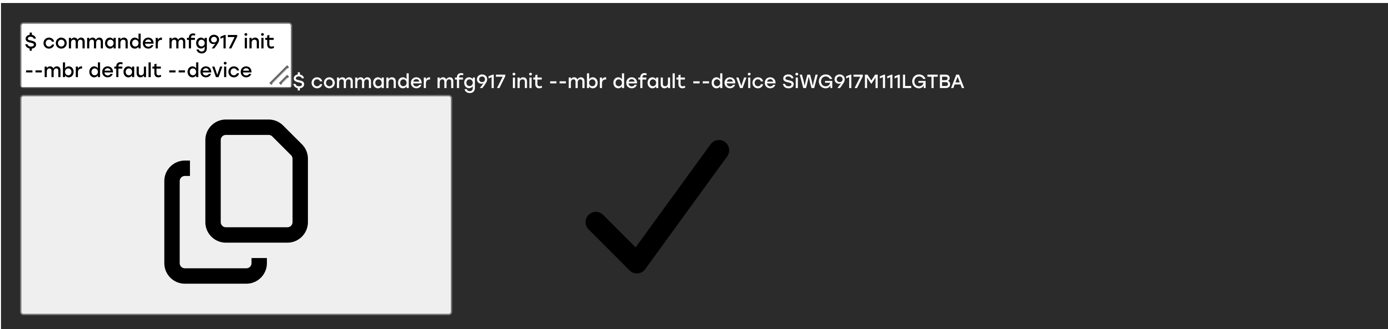
Providing an NWP (TA) MBR (as a binary file) using the `--mbr` option is optional, and its purpose is to provide information about the destination address of the activation code. If required, updates to the NWP MBR can be applied by providing a JSON file with the `--data` option. If the `--mbr` option is omitted, the default activation code address is used. Providing `--mbr 'default'` will use the default NWP MBR for your device, based on the provided device part number using the `--device` option.

> Note: After the `mfg917 init` command has been run, the device needs to be power cycled for any changes to take effect.

Command Line Syntax

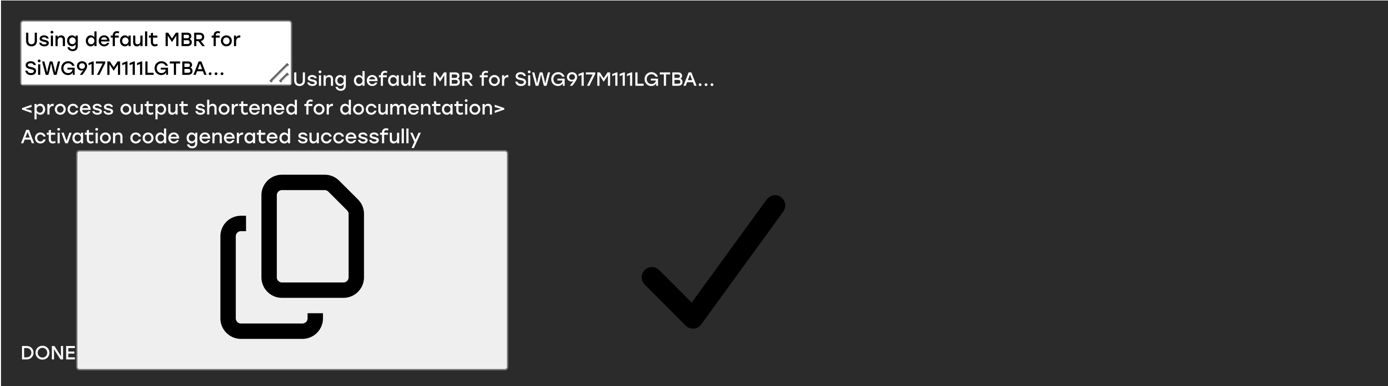



Command Line Input Example




This command line initializes the device's PUF and generates an activation code, using a default NWP MBR for the device part number 'SiWG917M111LGTBA'.

Command Line Output Example




## Provision Security Keys to the Device

Provisioning device keys is done using the `mfg917 provision` command, by providing a key configuration JSON file containing the keys you want to store on your device with the `--keys` option. Supported keys for storing on

Copyright © 2026 Silicon Laboratories. All rights reserved.

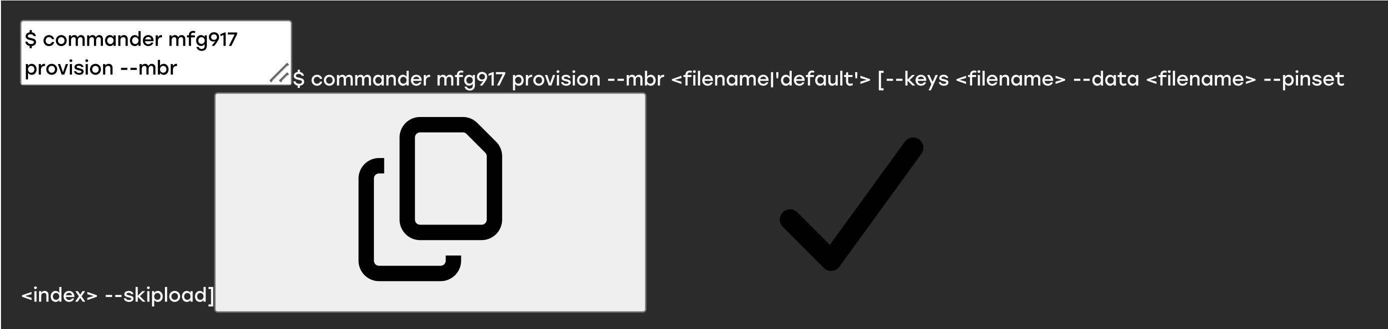
the device are M4/NWP (TA) public keys and M4/NWP OTA keys. In addition, a private attestation key is required for the provisioning sequence.

If you don't want to provision any keys during the provisioning sequence, the `--keys` option may be omitted.

If required, updates to the NWP MBR can be applied before writing it to the device by providing a JSON file with the `--data` option.

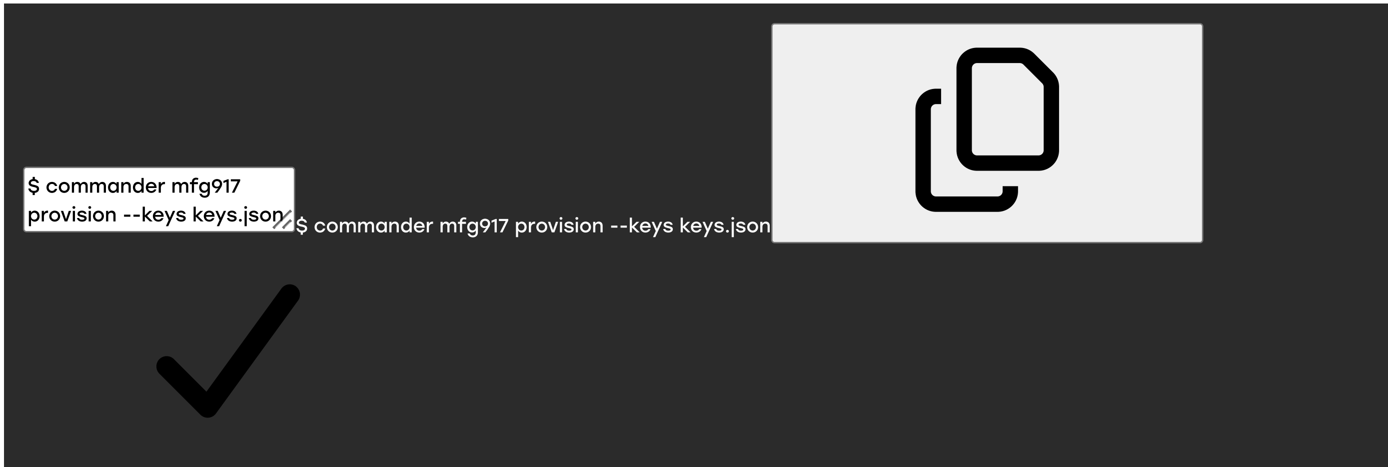
> Note: After the `mfg917 provision` command has been run, the device needs to be power cycled for any changes to take effect.

Command Line Syntax



```
$ commander mfg917 provision --mbr <filename|'default'> [--keys <filename> --data <filename> --pinset <index> --skipload]
```

Command Line Input Example

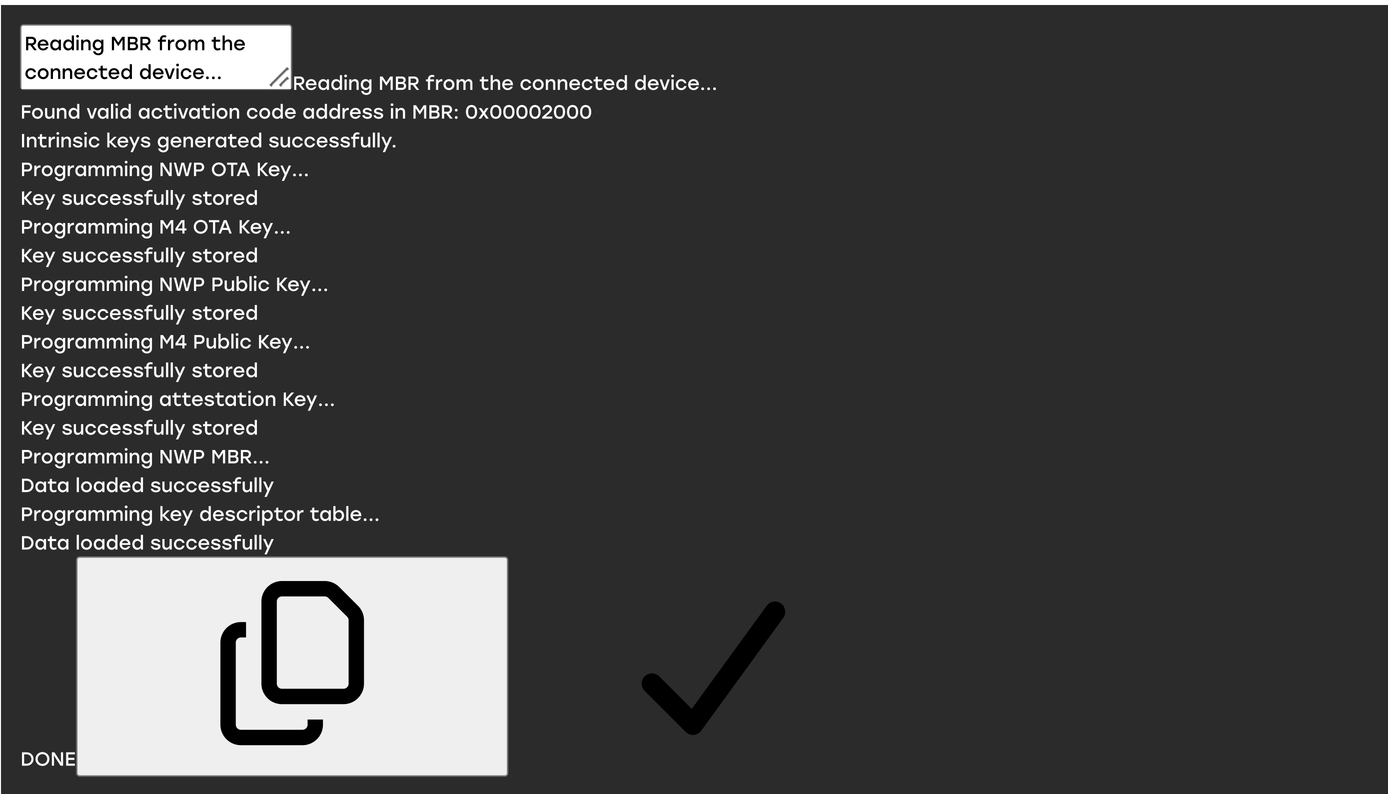


```
$ commander mfg917 provision --keys keys.json
```

This command line provisions the keys contained in the JSON file 'keys.json' to the device.

Command Line Output Example

Copyright © 2026 Silicon Laboratories. All rights reserved.



```
Reading MBR from the connected device...
Found valid activation code address in MBR: 0x00002000
Intrinsic keys generated successfully.
Programming NWP OTA Key...
Key successfully stored
Programming M4 OTA Key...
Key successfully stored
Programming NWP Public Key...
Key successfully stored
Programming M4 Public Key...
Key successfully stored
Programming attestation Key...
Key successfully stored
Programming NWP MBR...
Data loaded successfully
Programming key descriptor table...
Data loaded successfully
DONE
```

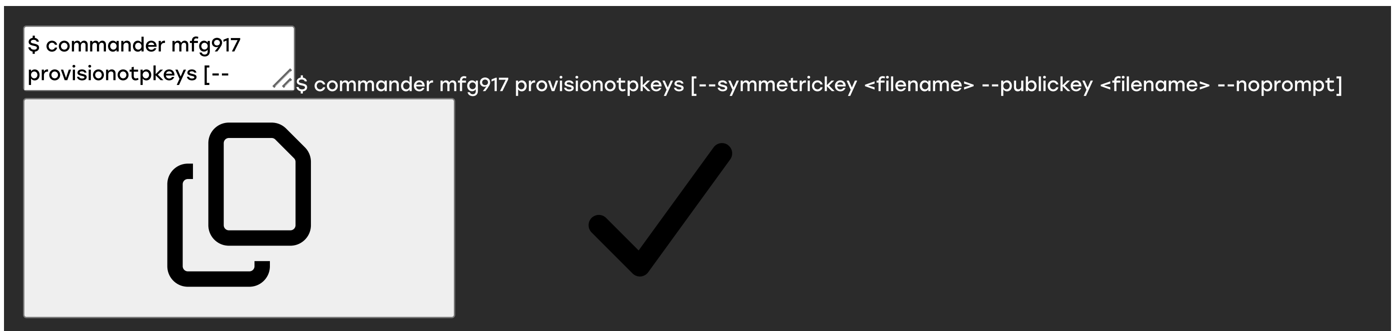
## Provision OTP Security Keys to the Device

Provisioning one-time programmable (OTP) keys is done using the `mfg917 provisionotpkeys` command. Symmetric (AES) and public (ECDSA) keys can be provided via the `--symmetrickey` and `--publickey` options, respectively. These keys can be used later for enabling MIC and signature-based write protection of the device configuration (see Protection Device Configuration).

The symmetric key must be provided as a .bin file, or as a hex-encoded string in a .txt file. The public key must be provided as a DER-formatted binary file (.der), or as a .pem file. Alternatively, both options accept a key configuration JSON file, as long as the relevant OTP keys are available in the JSON file.

> Note: Writing OTP keys is a permanent action and cannot be reverted.

Command Line Syntax



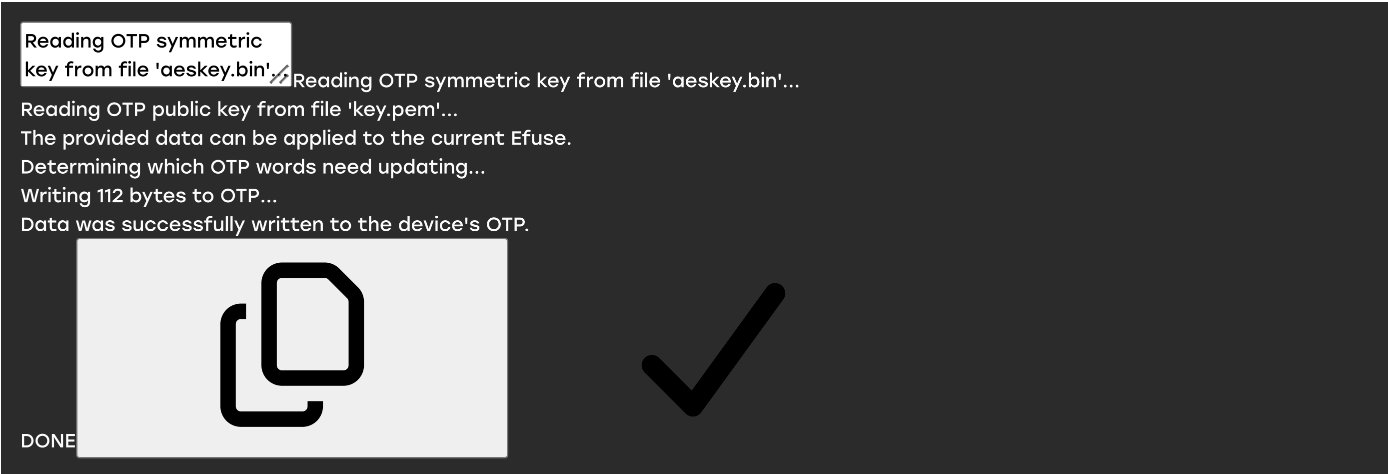
```
$ commander mfg917 provisionotpkeys [--symmetrickey <filename> --publickey <filename> --noprompt]
```

Command Line Input Example

Copyright © 2026 Silicon Laboratories. All rights reserved.

This command line reads the symmetric key 'aeskey.bin' and the public key 'key.pem' and writes them into the OTP of the device.

Command Line Output Example



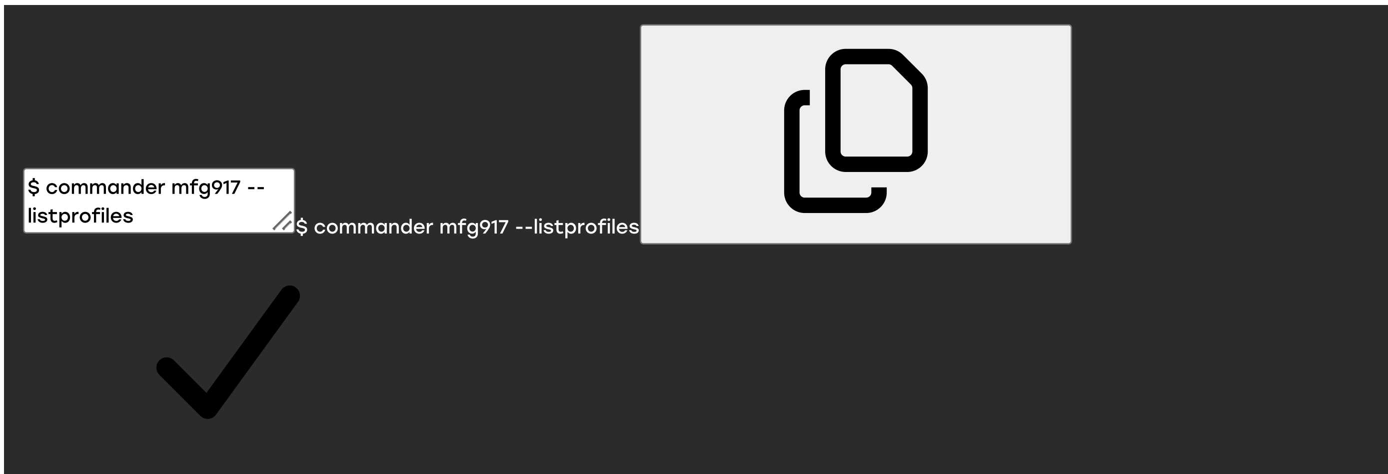

## List Available Device Profiles

Simplicity Commander can be used to store and apply frequently used device configuration JSON files, referred to as 'profiles'. A list of the available profiles can be produced using the `mfg917 provision --listprofiles` command. Along with this list, the location of user-defined profiles is also shown, including instructions on how to add such custom profiles.

Command Line Syntax

Command Line Input Example

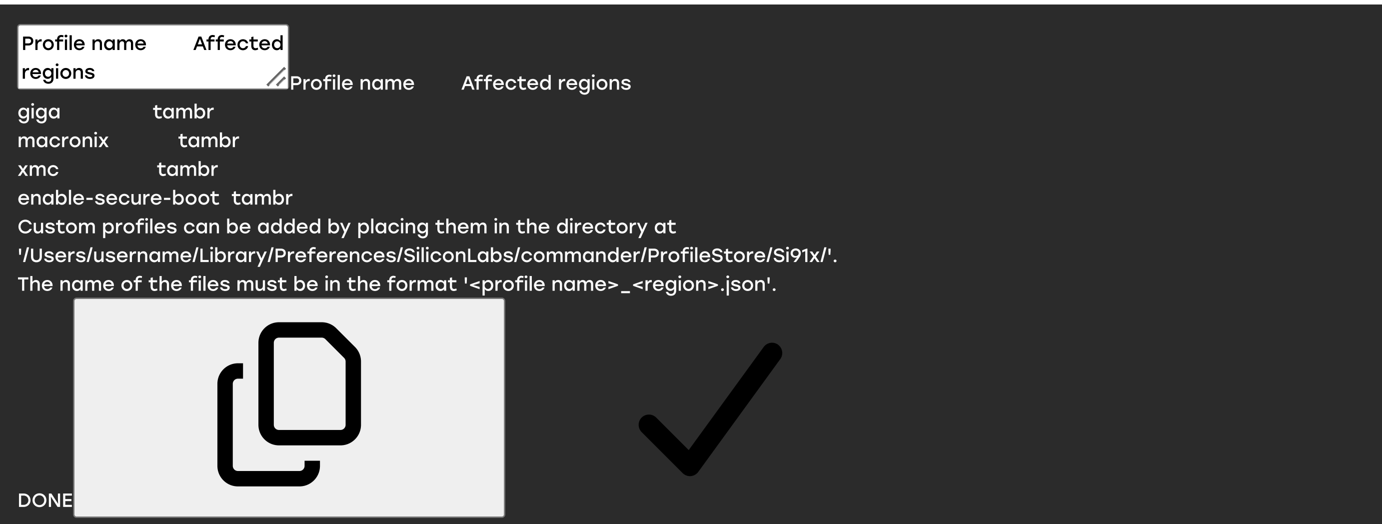
Copyright © 2026 Silicon Laboratories. All rights reserved.



```
$ commander mfg917 --listprofiles
```

This command line displays the available profiles on your system, along with the location of custom user-defined profiles.

Command Line Output Example



```
Profile name      Affected regions
giga              tambr
macronix          tambr
xmc               tambr
enable-secure-boot  tambr
Custom profiles can be added by placing them in the directory at
'/Users/username/Library/Preferences/SiliconLabs/commander/ProfileStore/Si91x/'.
The name of the files must be in the format '<profile name>_<region>.json'.
DONE
```

## Provision Device Profile to the Device

Applying a device profile is done through the `mfg917 provision` command, using the `--profile` option.

> Note: Only TA (NWP) MBR data can be applied as part of a profile.

These profiles may be useful for quickly changing between e.g. secure and non-secure device configurations, or for applying a set of changes across multiple devices in e.g. a production environment.

Profiles can be provided along with both `--mbr` and `--data` options; the changes denoted by the profile will be applied as the last step before writing the data to the device.

Command Line Syntax

Copyright © 2026 Silicon Laboratories. All rights reserved.

```
$ commander mfg917 provision --profile <name> [--mbr <filename|'default'> --data <filename>]
```

Command Line Input Example

```
$ commander mfg917 provision --profile enable-secure-boot
```

This command line applies the 'enable-secure-boot' profile to the device's TA (NWP) MBR and provisions it to the device.

Command Line Output Example

```
Using MBR from the connected device...
Applying profile 'enable-secure-boot' to tambr...
Reading JSON...
Loading RAM code (321776 bytes)...
Starting RAM code...
Programming ROM patch...
Data loaded successfully
Programming NWP MBR...
Data loaded successfully
Programming M4 MBR...
Data loaded successfully
Programming IPMU data...
Data loaded successfully
Programming M4 Efuse map version...
Data loaded successfully
DONE
```

## Protect Device Configuration

Simplicity Commander can be used to write-protect the device configuration using the `mfg917 protectconfig` command. This protection is permanent and is applied over a user-specified length, starting from address

Copyright © 2026 Silicon Laboratories. All rights reserved.

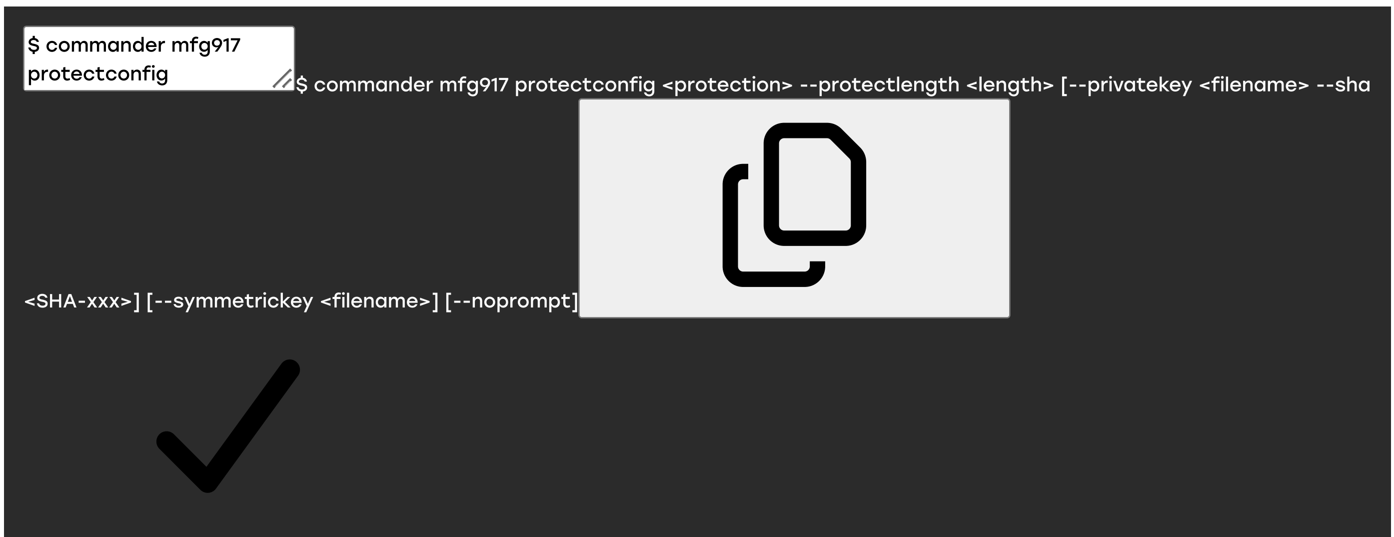
0x04000000 (the start of the NWP (TA) configuration region). Note that only certain protection lengths are supported, ranging from 1024-65536 bytes. Commander will provide suggestions of the nearest allowed lengths if the provided length is not supported.

Two types of protection are available: message integrity check (MIC)-based protection and signature-based protection. MIC-based protection requires a 16 byte AES key as a .bin file or as a hex-encoded string in a .txt file, provided with the `--symmetrickey` option. Signature-based protection requires a private ECDSA key, either as a DER-formatted binary file (.der), or as a .pem file, provided with the `--privatekey` option. Alternatively, both options accept a key configuration JSON file, as long as the relevant OTP keys are available in the JSON file.

When enabling signature-based protection, the public key counterpart of the provided private key must be present on the device. If there is no such public key stored in the device, Commander will extract the public key from the provided private key and store it on the device before proceeding.

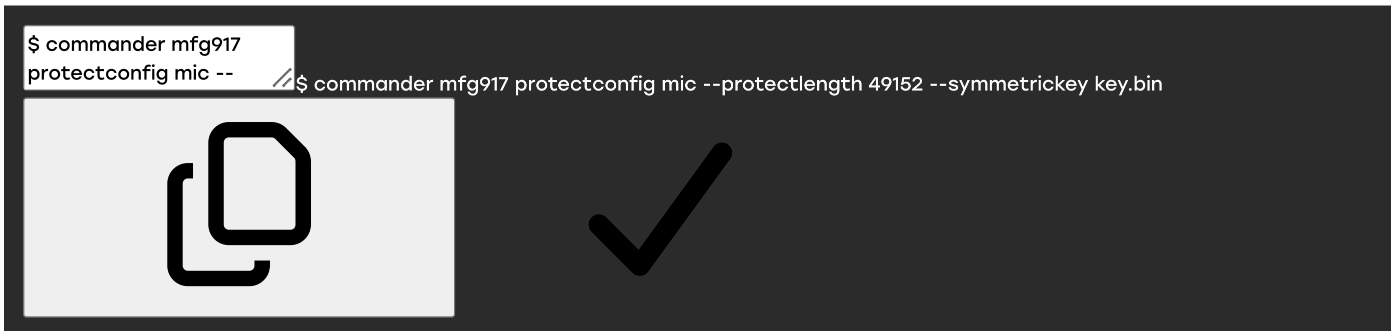
> Note: Enabling configuration protection is permanent and cannot be reverted. Any subsequent attempts at writing new configurations to a protected device will fail and/or may render your device unrecoverable.

Command Line Syntax



```
$ commander mfg917 protectconfig <protection> --protectlength <length> [--privatekey <filename> --sha <SHA-xxx>] [--symmetrickey <filename>] [--noprompt]
```

Command Line Input Example

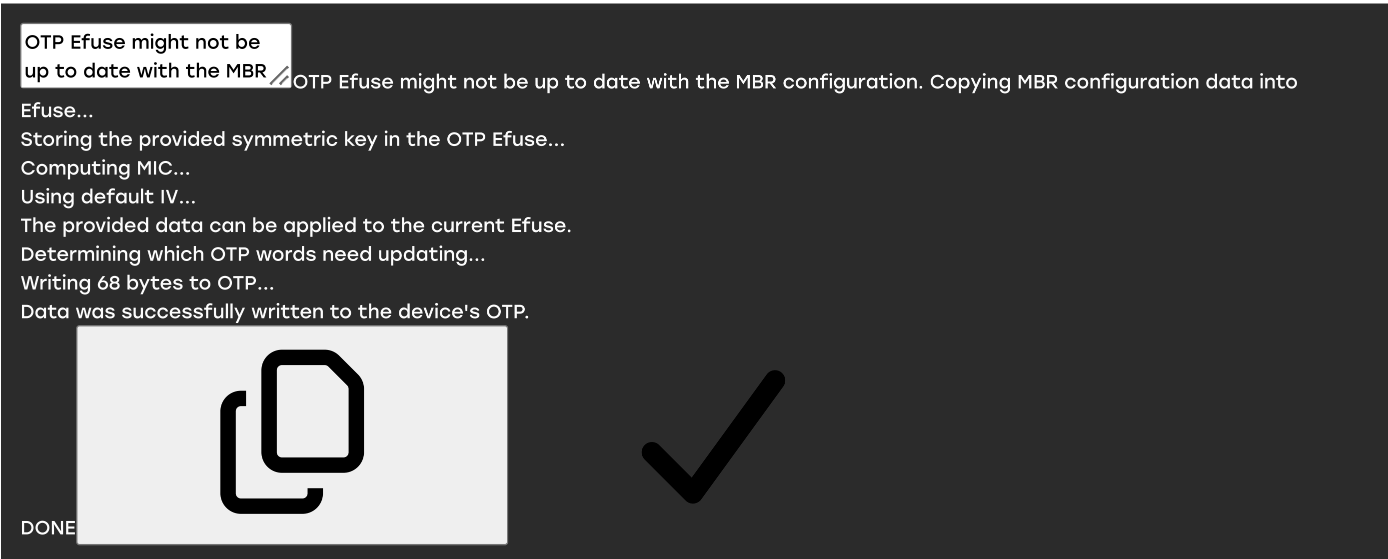


```
$ commander mfg917 protectconfig mic --protectlength 49152 --symmetrickey key.bin
```

This command line will enable MIC based protection of 49152 bytes of the device configuration, using 'key.bin' as the key for the MIC computation.

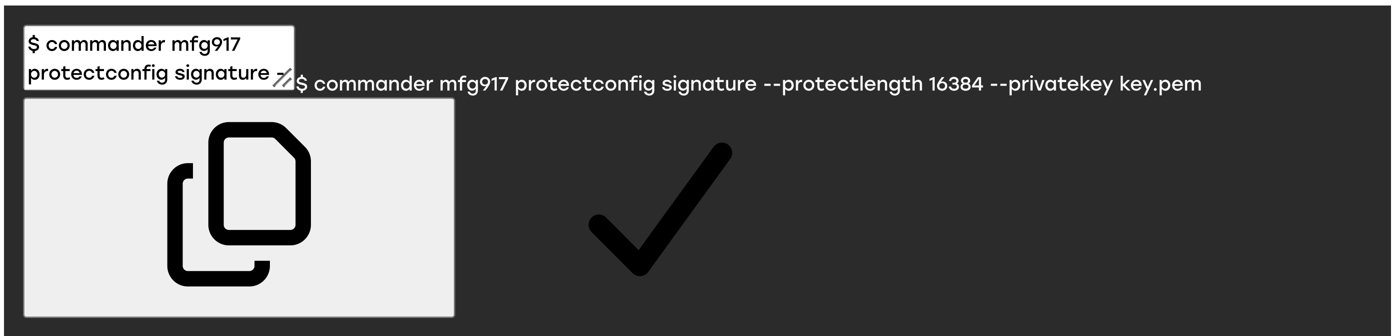
Command Line Output Example

Copyright © 2026 Silicon Laboratories. All rights reserved.



```
OTP Efuse might not be up to date with the MBR configuration. Copying MBR configuration data into Efuse...
Storing the provided symmetric key in the OTP Efuse...
Computing MIC...
Using default IV...
The provided data can be applied to the current Efuse.
Determining which OTP words need updating...
Writing 68 bytes to OTP...
Data was successfully written to the device's OTP.
DONE
```

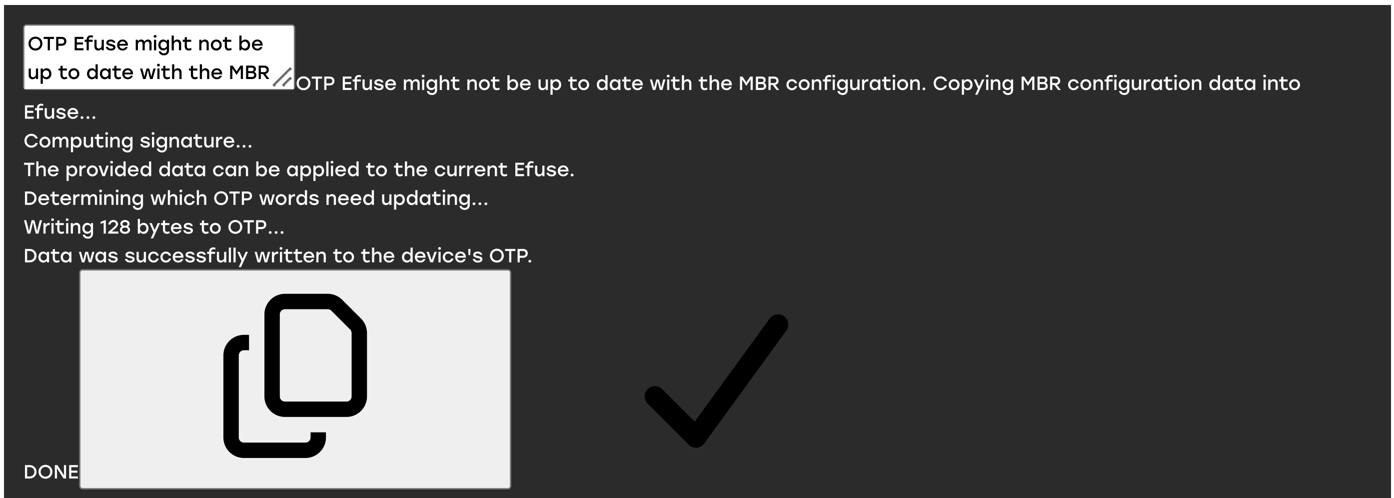
Command Line Input Example



```
$ commander mfg917 protectconfig signature --protectlength 16384 --privatekey key.pem
```

This command line enables signature-based protection of the first 16384 bytes of the device configuration, using the private ECDSA key 'key.pem' for generating the signature. The signature is placed at the end of the protected region.

Command Line Output Example

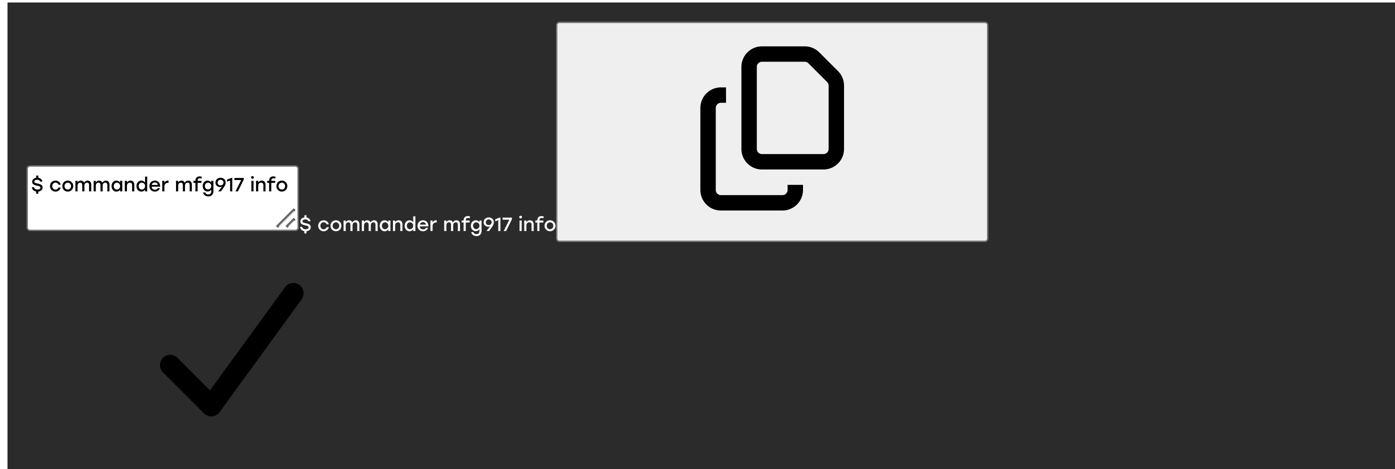


```
OTP Efuse might not be up to date with the MBR configuration. Copying MBR configuration data into Efuse...
Computing signature...
The provided data can be applied to the current Efuse.
Determining which OTP words need updating...
Writing 128 bytes to OTP...
Data was successfully written to the device's OTP.
DONE
```

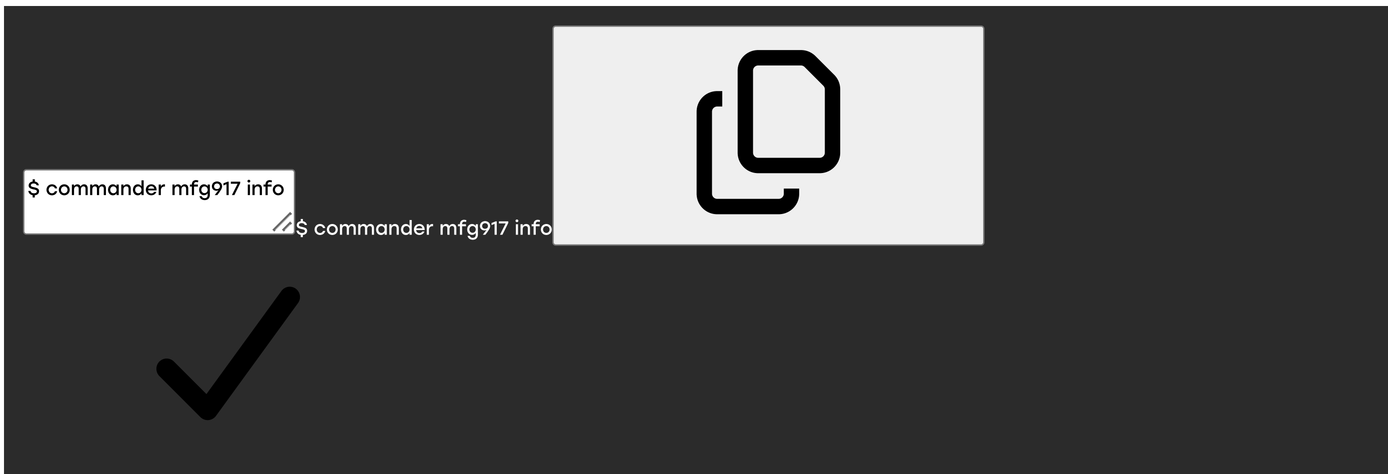
## Get Information About Device Configuration

Key information about the device configuration can be extracted using the `mfg917 info` command. Simplicity Commander will read certain regions of your device (including the NWP (TA) MBR and the Efuse regions), parse the information, and present key information like the device's MAC address, flash size, and current security parameters.

Copyright © 2026 Silicon Laboratories. All rights reserved.




Command Line Input Example




This command line extracts information about the current configuration of the target device.

Command Line Output Example

Copyright © 2026 Silicon Laboratories. All rights reserved.

```
Reading 1024 bytes from 0x040003e0
Reading 1024 bytes from 0x040003e0
OPN                                    : SiWG917M111MGTBA
WiFi MAC address                       : 6C5CB1C43F90
BLE MAC address                        : 6C5CB1C43F92
Flash size                             : 8MB
NWP firmware version                   : 1711.2.10.1.3.0.7
MBR variant                            : 0x1F (1.8 MB)
Manufacturing SW version               : 2.4 (36)
Application region start address       : 0x081f0000
Application code start address         : 0x08202000
Application region end address         : 0x0840ffff
Flash configuration                    : Common flash
Mode                                   : SOC
Integrity protection active            : None
NWP roll-back prevention               : Disabled
NWP digital signature validation       : Disabled
NWP firmware encryption                : Disabled
NWP secure boot                        : Disabled
Application roll-back prevention       : Disabled
Application digital signature validation : Disabled
Application code encryption            : Disabled
Application secure boot                : Disabled
DONE
```

Copyright © 2026 Silicon Laboratories. All rights reserved.

VCOM Commands

# VCOM Commands

Simplicity Commander supports configuring the adapter board's Virtual COM (VCOM) configuration (baud rate and handshake) using the `vcom config` command.

Additionally, Simplicity Commander can connect to the adapter board's VCOM port to communicate with the target device using the `vcom connect` command. Communications will be active until terminated by pressing CTRL+C.

## Configure Adapter VCOM Settings

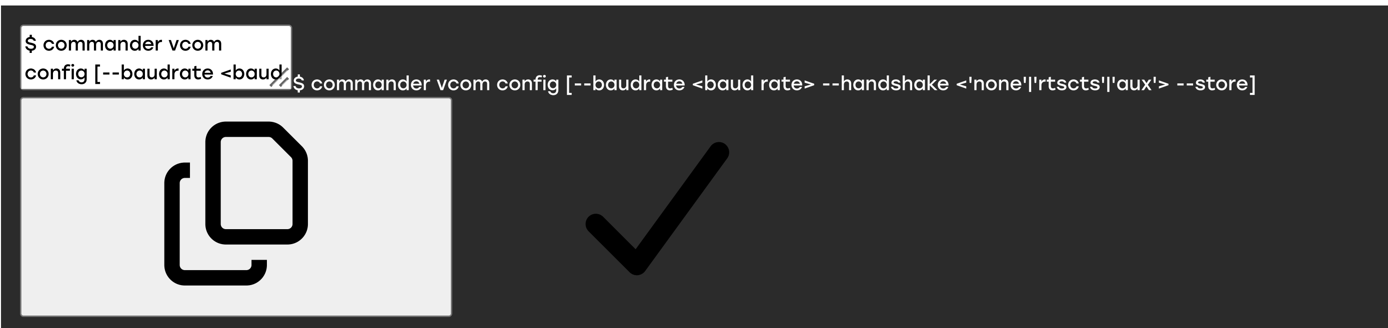
The adapter's VCOM baudrate is set with the `--baudrate` option. The handshake type is set with the `--handshake` option, and the available handshake configurations are 'none' (disabled), 'rtscts' (hardware flow control, RTS+CTS), and 'aux' (auxiliary UART).

By default, the VCOM settings that are configured using this command are *not* stored permanently on the adapter board (i.e. rebooting the adapter will revert the adapter to the previously stored configuration). Provide the `--store` option if you want the applied configuration to be persistent across adapter reboots.

> Note: Handshake type 'aux' is not supported on all adapter boards.

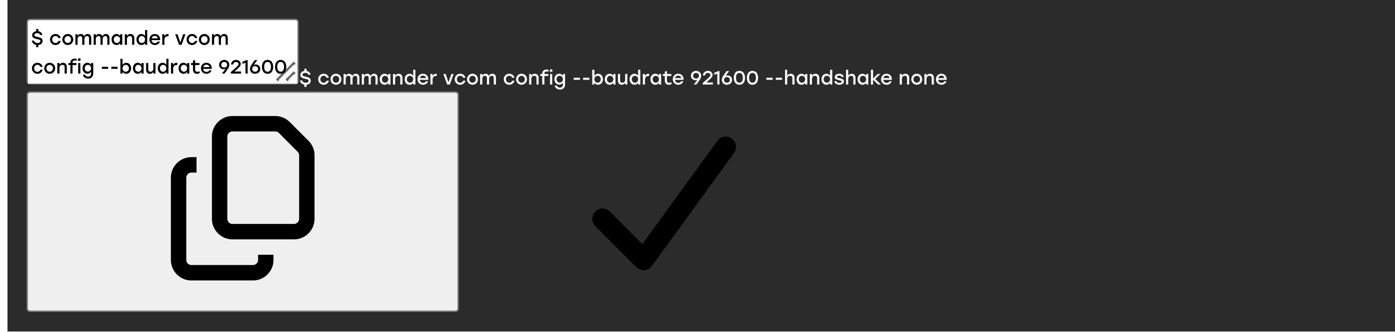
> Note: Not all adapter boards support setting a baud rate other than 115200 baud while handshake type 'aux' is enabled.

Command Line Syntax



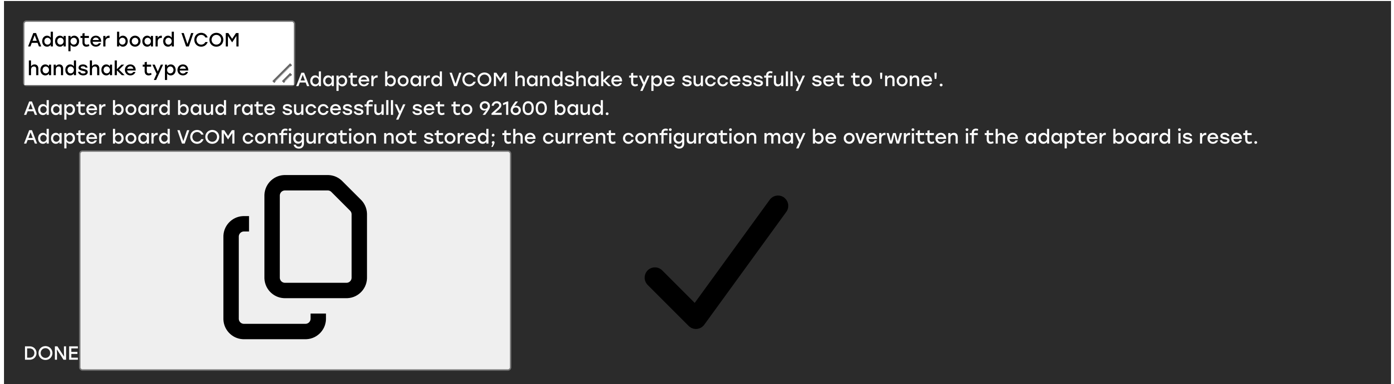

Command Line Input Example

Copyright © 2026 Silicon Laboratories. All rights reserved.




This command line sets the adapter board's VCOM baud rate to 921600 baud and disables hardware flow control.

Command Line Output Example




## VCOM Communications

Using the `vcom connect` command, you can communicate with the target device on your adapter board via a serial connection. If you are connecting to an adapter board over USB (i.e. by providing `--serialno`), Simplicity Commander will communicate via the serial port of the adapter. If you are instead connecting over the network (i.e. by providing `--ip`), Simplicity Commander will open a TCP socket and connect to the adapter's IP address via port 4901.

The line ending to use can be specified with the `--lineending` option. Carriage return (CR) and line feed (LF) is the default line ending.

Providing the `--hex` option will handle all input and output as hexadecimal strings. No line ending is added in this case, i.e. any provided line endings are ignored.

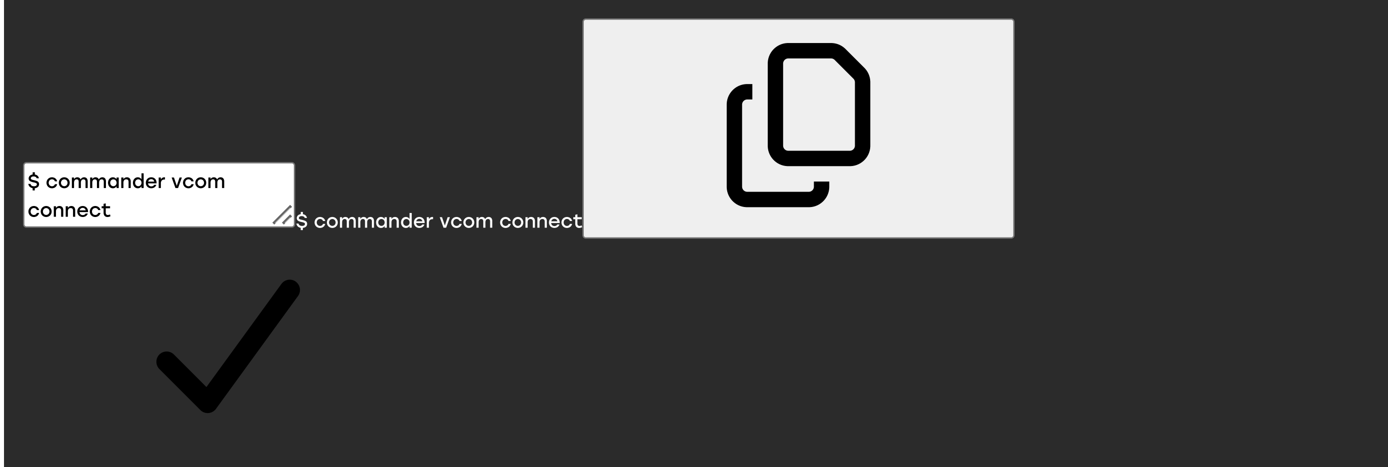
For certain adapter boards, you can restart the target in ISP mode upon connecting by providing the `--restartinisp` option. This is only supported for SiWx917 devices.

Command Line Syntax

Copyright © 2026 Silicon Laboratories. All rights reserved.

```
$ commander vcom connect [--hex --restartinisp --lineending <'cr'|'lf'|'crlf'|'none'>]
```

Command Line Input Example



```
$ commander vcom connect
```

This command line starts VCOM communications with the target device using the default line ending.

Command Line Output Example

```
Opening serial port 'COM3'...
Connection established!
<data written by the target application>
^C
Serial port was disconnected.
Connection terminated by user.
DONE
```

Copyright © 2026 Silicon Laboratories. All rights reserved.

Completion Commands

# Completion Commands

To simplify the user experience while using Simplicity Commander's command line interface (CLI), command completion (TAB completion) scripts can be generated using the `completion generate` command. The scripts can be used in your shell environment in order to provide completion for all available commands, sub commands and options. The scripts also provide automatic serial number completion for currently connected JLink devices when providing the `--serialno` option (or its short form, `-s` ), as well as completions for device part numbers to the `--device` option (or its short form, `-d` ).

Completion scripts can be generated for the following Unix-based shells:

- bash
- zsh
- fish

Generating a completion script is fast, and thus running Simplicity Commander at the startup of the shell is a simple way to ensure that the TAB completions are available in the shell session. This will also ensure that with later releases of Simplicity Commander (assuming the previous application package is replaced by the newer version), any new commands, sub commands or options will be added to the TAB completion.

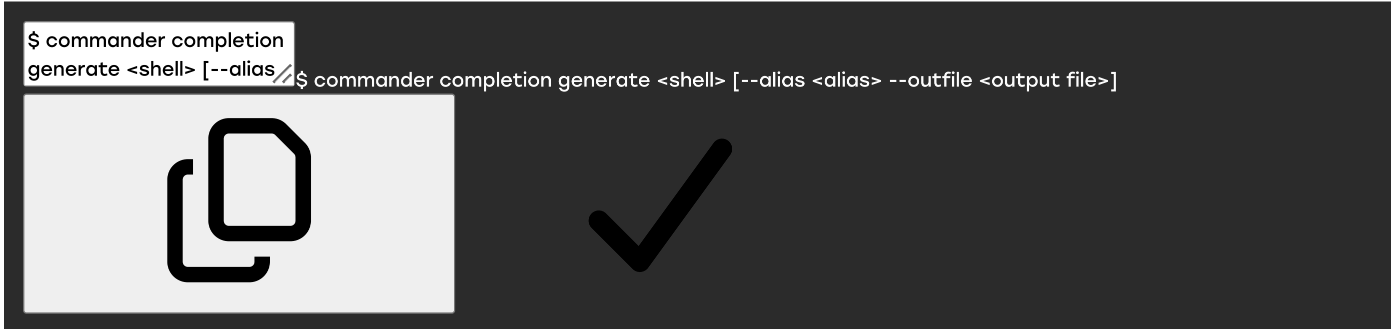
The procedures for employing the completion scripts depend on the shell in use, and on the exact setup of the shell environment. For bash and zsh, the completion script can be installed by sourcing the script in the shell's configuration file ( `.bashrc` and `.zshrc` , respectively). For fish, one common practice is to place completion scripts within the `completions` directory inside fish's configuration folder, which on most systems will be `~/.config/fish/completions` .

## Generate Completion Script

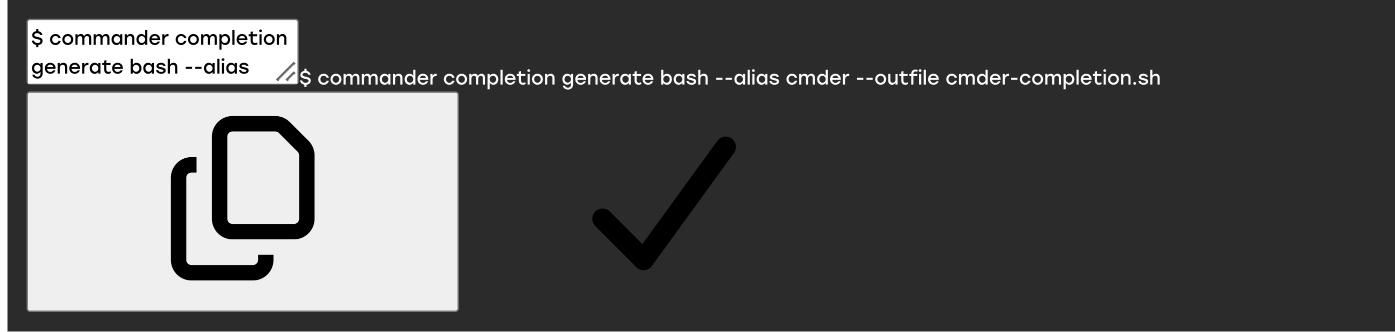
Simplicity Commander supports generating completion scripts using the `completion generate` command. You can provide an alternative alias for which the completions will trigger, using the `--alias` option. This is useful if you have an alias for your installation of Simplicity Commander already defined in your shell environment. Aliases must contain alphanumeric characters only (underscores are also allowed), and cannot start with a number. The default alias is 'commander'.

By default, this command will output the completion script directly to the console. If you instead want to save the output script to a file, you can provide the `--outfile` option.

Command Line Syntax

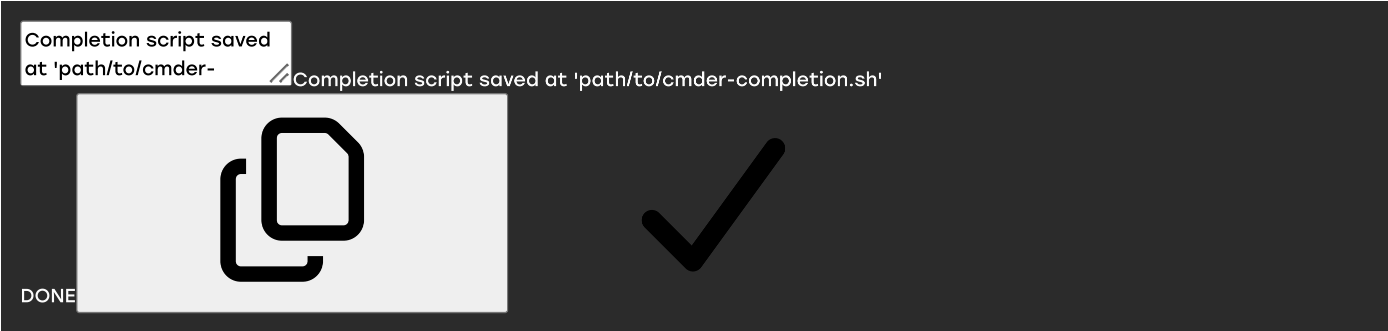



Command Line Input Example




This command line generates a TAB completion script for bash, which will trigger for the 'cmder' keyword, and saves it to the file 'cmder-completion.sh'.

Command Line Output Example



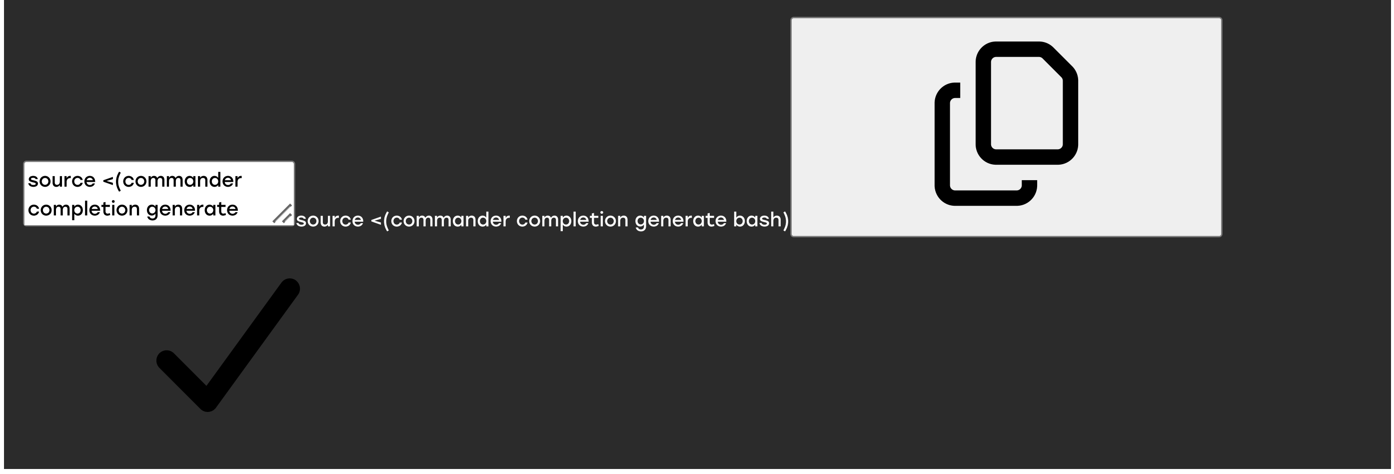

## Install Completion Script

Here follows basic examples on how to install the scripts on bash, zsh and fish shells. All these approaches require that the Simplicity Commander executable is in your system's PATH or is otherwise visible to your shell environment.

Using this approach, whenever newer versions of Simplicity Commander are released, simply replace the existing Commander application package with the new version. All completions will be kept up-to-date, making this installation a one-time only procedure.

bash

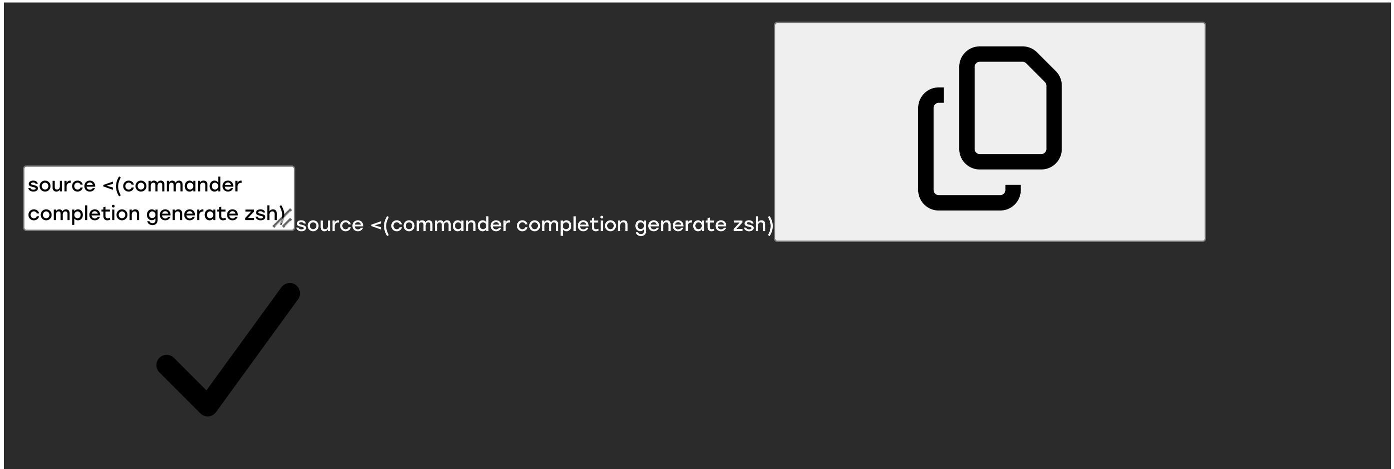
Add the following line to your `.bashrc` configuration file:




zsh

Add the following line to your `.zshrc` configuration file:

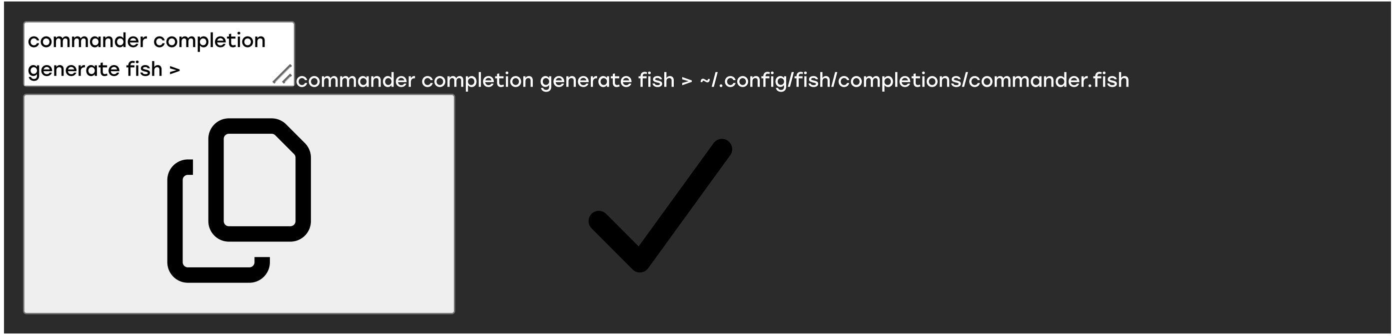
Copyright © 2026 Silicon Laboratories. All rights reserved.



```
source <(commander completion generate zsh)
```

fish

Add the following line to your `~/.config/fish/config.fish` configuration file:



```
commander completion generate fish > ~/.config/fish/completions/commander.fish
```

Copyright © 2026 Silicon Laboratories. All rights reserved.

LittleFS Commands

# LittleFS Commands

Simplicity Commander supports initializing and manipulating instances of LittleFS filesystem binary files using the `commander littlefs` commands. LittleFS is an open-source lightweight filesystem designed for microcontrollers, with features including power-loss resilience and flash wear leveling mechanisms.

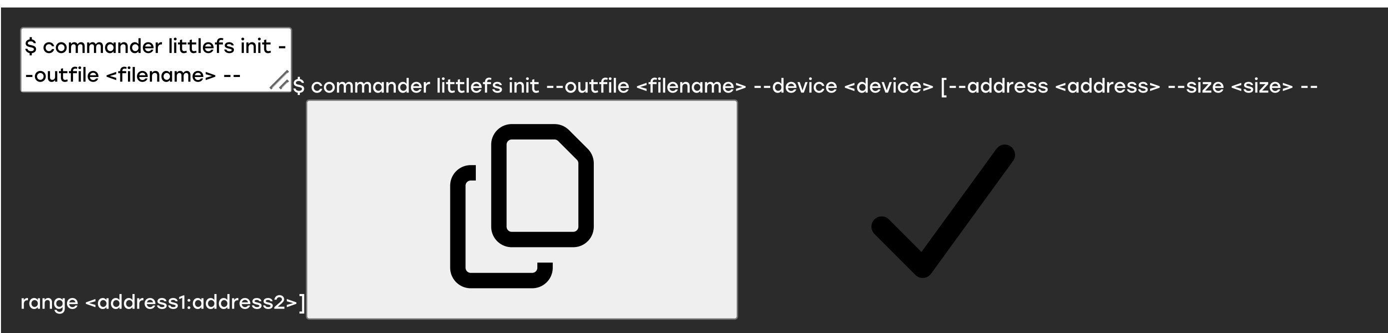
For all `littlefs` commands where a LittleFS instance can be provided using the `--infile` option, the option can be omitted to instead let Commander extract the LittleFS instance from the device's flash. By default, Commander will look for the LittleFS instance in the device's main flash region, however a search start address or range can be specified using the `--address` or `--range` options, respectively.

When working with LittleFS instances off-device, Commander needs to be aware of certain device-specific details. The `--device` option is therefore required for all `littlefs` commands when a LittleFS instance is provided to Commander using the `--infile` option.

## Initialize an Empty LittleFS Instance

Initializing an empty LittleFS instance is done using the `littlefs init` command. The size of the instance must be specified by either setting the start address and the size using the `--address` and `--size` options, or by providing the range using the `--range` option. The LittleFS instance can be stored in .bin, .hex and .s37 file formats.

Command Line Syntax



```
$ commander littlefs init --outfile <filename> --device <device> [--address <address> --size <size> --range <address1:address2>]
```

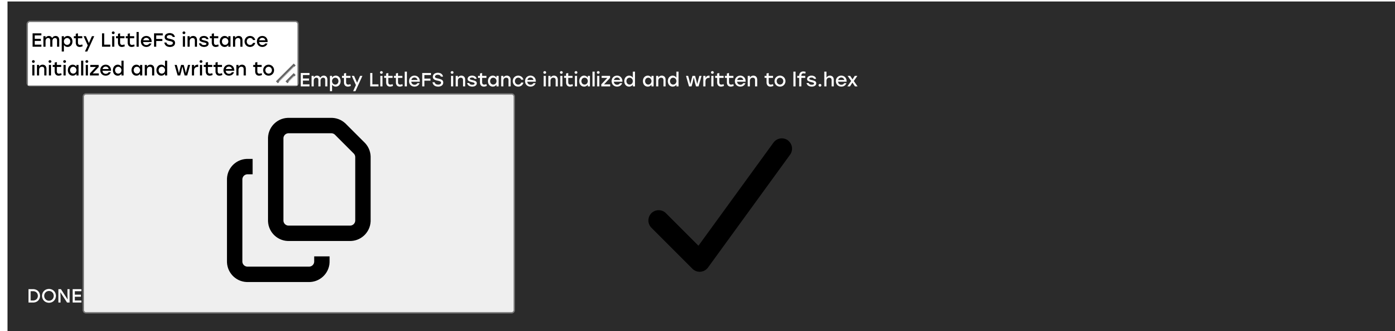
Command Line Input Example

```
$ commander littlefs init --outfile lfs.hex --device SiWG917M111MGTBA --address 0x02100000 --size 0x40000
```

This command line initializes an empty LittleFS instance for an SiWG917 device, starting from address 0x02100000 with a size of 262144 bytes, and stores the instance in the file 'lfs.hex'.

Command Line Output Example

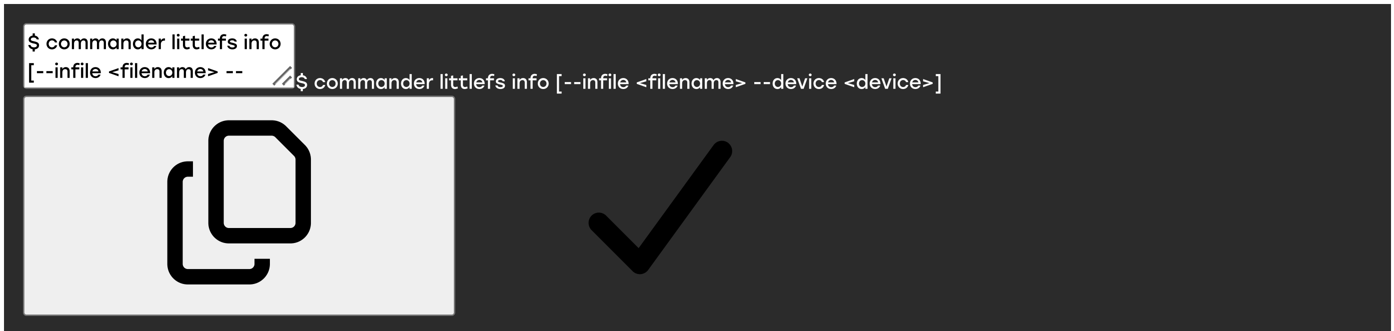
Copyright © 2026 Silicon Laboratories. All rights reserved.




## Get Information About a LittleFS Instance

Using the `littlefs info` command, information about the LittleFS instance can be retrieved. This information includes metadata about the LittleFS instance, as well as statistics about the storage use.

Command Line Syntax

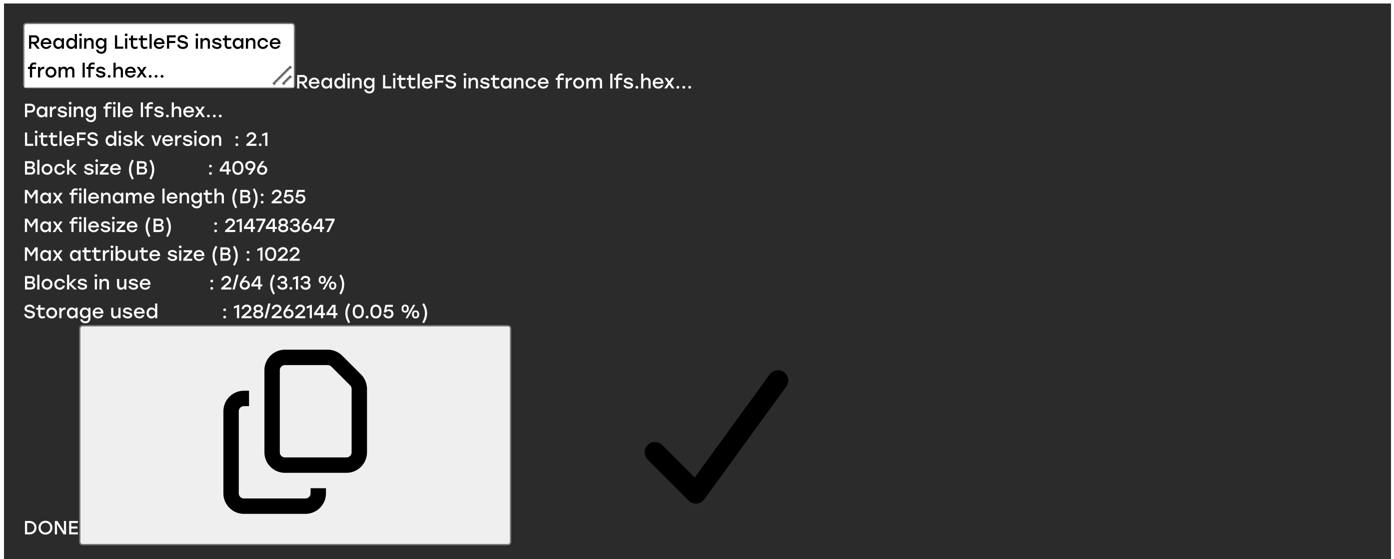



Command Line Input Example

This command line parses the LittleFS instance 'lfs.hex' and displays information about the instance to the console.

Command Line Output Example

Copyright © 2026 Silicon Laboratories. All rights reserved.



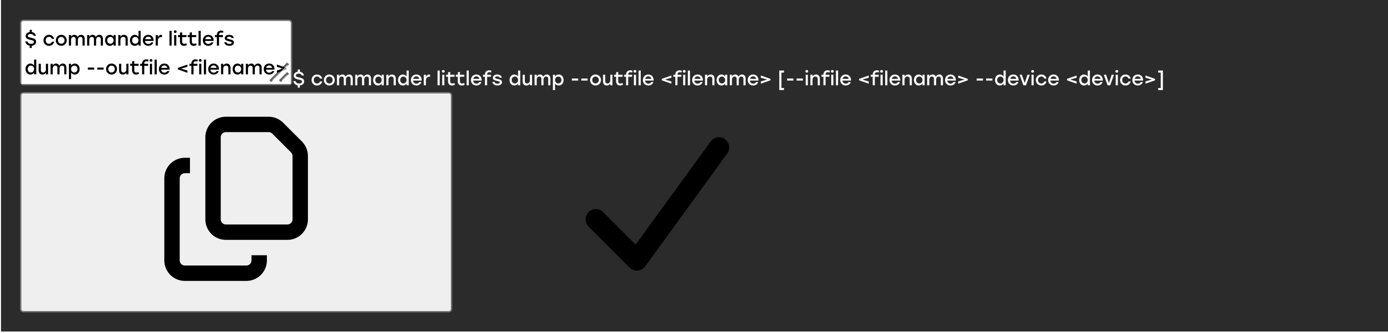
```
Reading LittleFS instance from lfs.hex...
Parsing file lfs.hex...
LittleFS disk version  : 2.1
Block size (B)         : 4096
Max filename length (B): 255
Max filesize (B)       : 2147483647
Max attribute size (B) : 1022
Blocks in use          : 2/64 (3.13 %)
Storage used           : 128/262144 (0.05 %)
DONE
```

## Dump a LittleFS Instance From Device

Extracting a LittleFS instance from a device flash or from an application binary can be done using the `littlefs dump` command. If the `--infile` option is omitted, Commander will search for the LittleFS instance in the connected device's main flash. If the `--infile` option is provided, Commander will instead look for the LittleFS instance in the provided binary file.

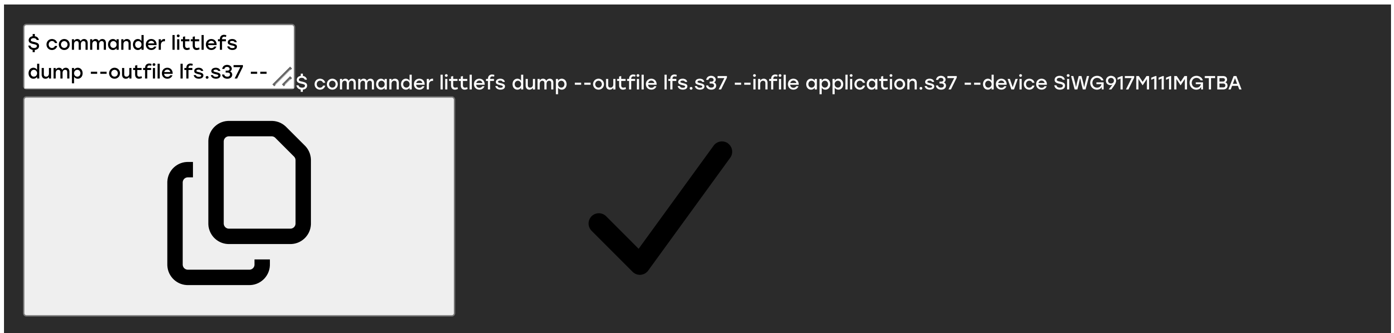
> Note: The LittleFS instance must be aligned to the start of a flash page/block.

Command Line Syntax



```
$ commander littlefs dump --outfile <filename> [--infile <filename> --device <device>]
```

Command Line Input Example

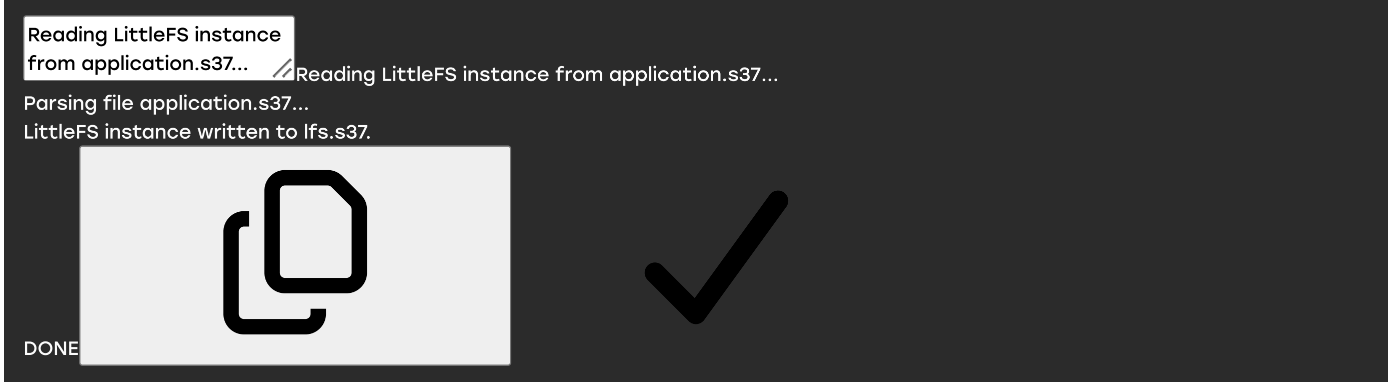


```
$ commander littlefs dump --outfile lfs.s37 --infile application.s37 --device SiWG917M111MGTBA
```

This command line parses the input file 'application.s37', extracts the LittleFS instance and saves it to the output file 'lfs.s37'.

Command Line Output Example

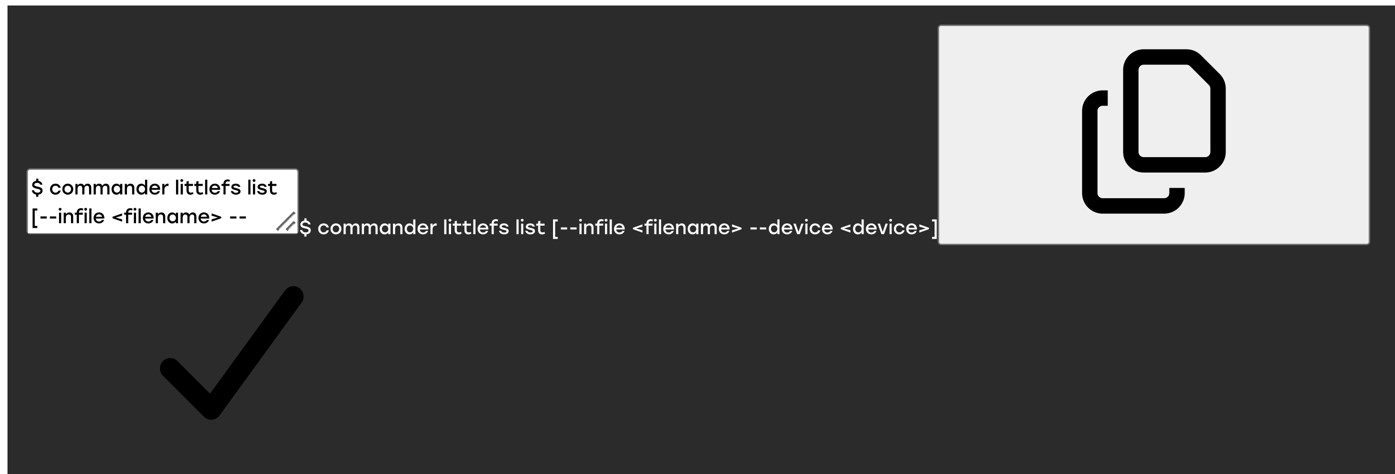
Copyright © 2026 Silicon Laboratories. All rights reserved.



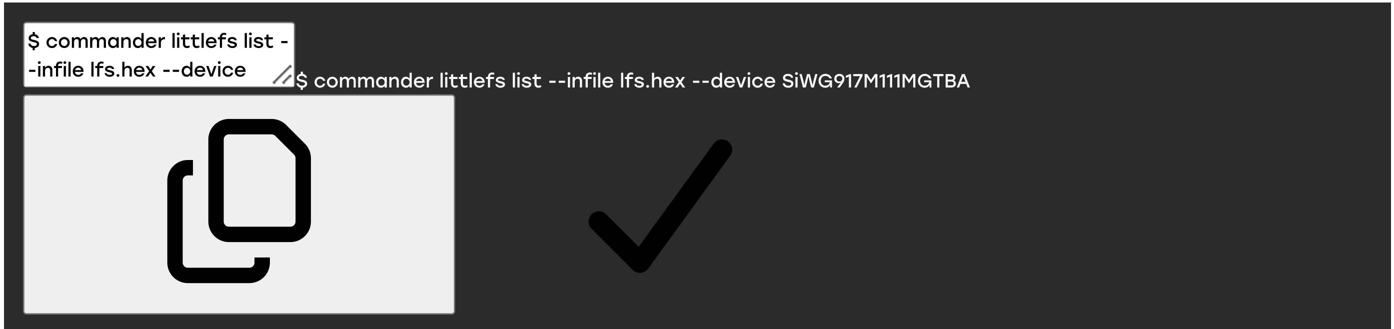

## List Contents in a LittleFS Instance

Listing the files in a LittleFS instance can be done using the `littlefs list` command. The file sizes are printed with each file in the directory tree.

Command Line Syntax



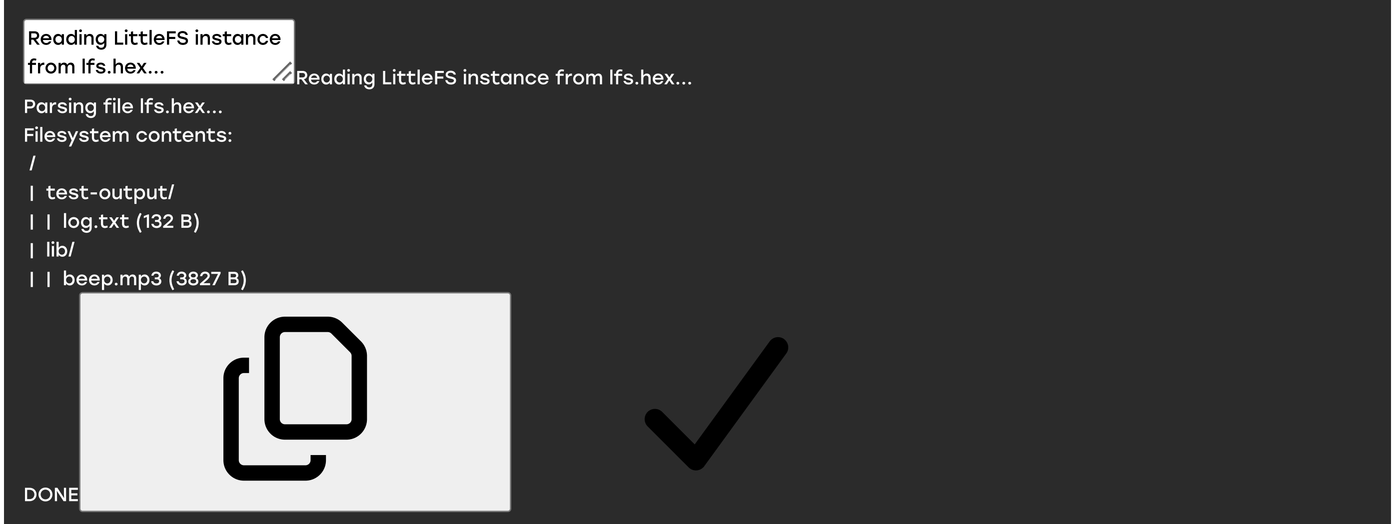

Command Line Input Example




This command line lists the files in the LittleFS instance 'lfs.hex'.

Command Line Output Example

Copyright © 2026 Silicon Laboratories. All rights reserved.

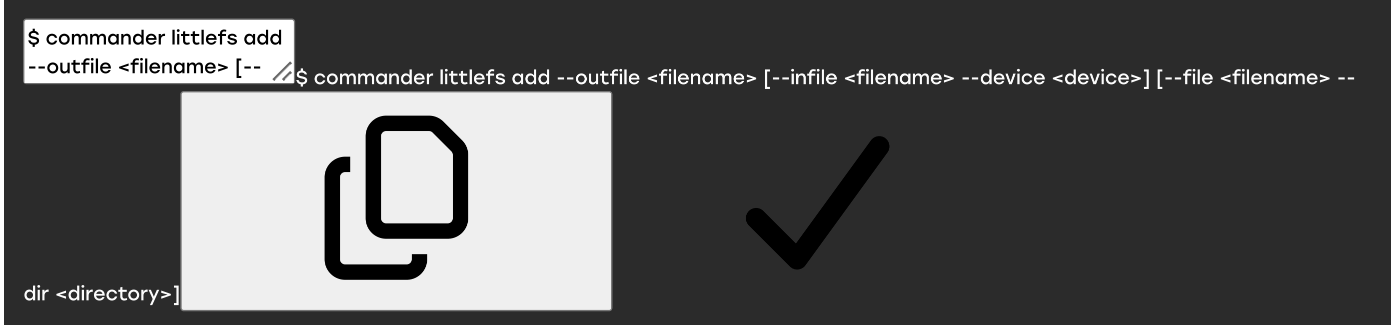



## Add Files to a LittleFS Instance

Adding files to a LittleFS instance is done using the `littlefs add` command. Singular files may be added using the `--file` option, whereas directories can be added using the `--dir` option. If any directories are specified, Commander will traverse those directories and recursively add all contained files and subdirectories, maintaining the directory structure in the resulting LittleFS instance.

All filenames/paths provided to Commander must be relative to the current working directory of Commander.

Command Line Syntax

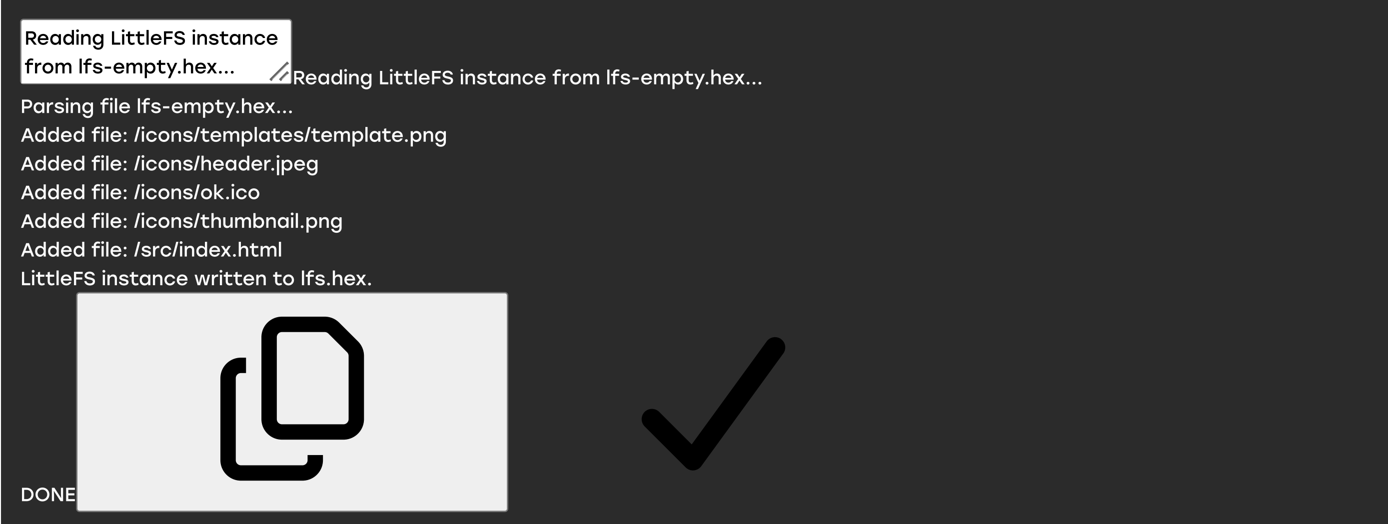



Command Line Input Example

This command line adds the file 'index.html' and the directory 'icons' (along with any contained files and subdirectories) to the LittleFS instance in 'lfs-empty.hex' and stores the updated LittleFS instance in 'lfs.hex'.

Command Line Output Example

Copyright © 2026 Silicon Laboratories. All rights reserved.

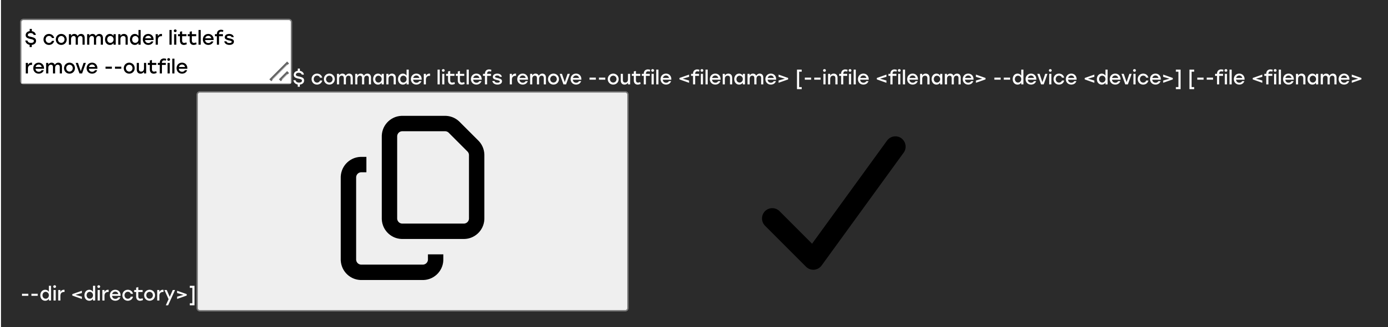



## Remove Files From a LittleFS Instance

Removing files from a LittleFS instance is done using the `littlefs remove` command. Singular files may be removed using the `--file` option, whereas directories can be removed using the `--dir` option. If any directories are specified, Commander will traverse those directories and recursively remove all contained files and subdirectories.

All filenames/paths provided to Commander must be absolute with respect to the LittleFS instance's root directory, `/`.

Command Line Syntax

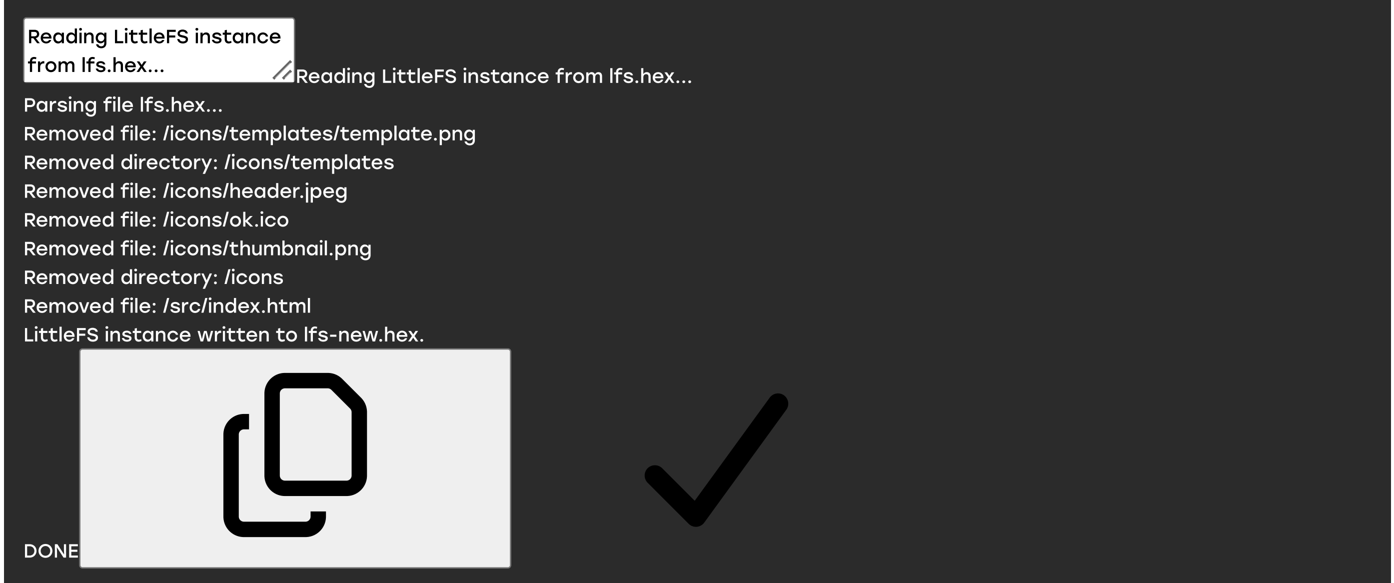



Command Line Input Example

This command line removes the file 'index.html' and the directory 'icons' (along with any contained files and subdirectories) from the LittleFS instance in 'lfs.hex' and stores the updated LittleFS instance in 'lfs-new.hex'.

Command Line Output Example

Copyright © 2026 Silicon Laboratories. All rights reserved.



```
Reading LittleFS instance from lfs.hex...
Parsing file lfs.hex...
Removed file: /icons/templates/template.png
Removed directory: /icons/templates
Removed file: /icons/header.jpeg
Removed file: /icons/ok.ico
Removed file: /icons/thumbnail.png
Removed directory: /icons
Removed file: /src/index.html
LittleFS instance written to lfs-new.hex.
DONE
```

## Extract Files From a LittleFS Instance

Extracting files from a LittleFS instance is done using the `littlefs extract` command. Singular files may be extracted using the `--file` option, whereas directories can be extracted using the `--dir` option. If any directories are extracted, Commander will traverse those directories and recursively extract all files and subdirectories.

All filenames/paths provided to Commander must be absolute with respect to the LittleFS instance's root directory, `/`.

The destination folder of the extracted files is specified using the `--dest` option. If you want the extracted files to be compressed and placed in a zip file, the `--zip` option can be used in place of the `--dest` option.

If only singular files (using the `--file` option) are being extracted, the individual files' paths will be ignored upon extraction, placing the files directly in the destination specified by the `--dest` / `--zip` option.

If no files or directories are specified, Commander will extract all the files in the LittleFS instance.

> Note: Zip file compression functionality requires Microsoft PowerShell version 5.0 or above on Windows, and the `zip` and `unzip` system utilities on Linux/Mac

Command Line Syntax

```
$ commander littlefs extract [--infile <filename> --device <device>] [--file <filename> --dir <directory> --dest <directory> --zip <filename>]
```

Command Line Input Example

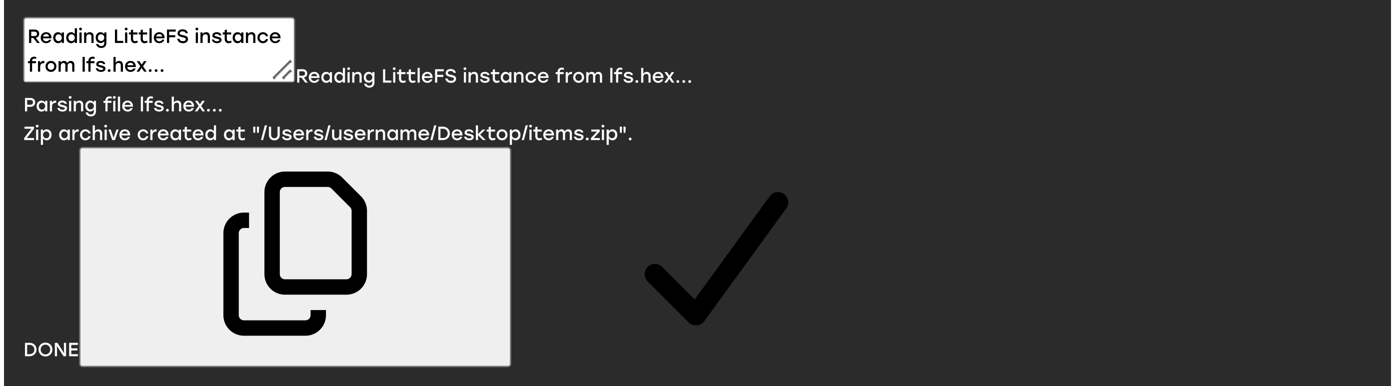


Copyright © 2026 Silicon Laboratories. All rights reserved.

```
$ commander littlefs extract --infile lfs.hex --device SiWG917M111MGTBA --zip items.zip
```

This command line extracts all the files in the LittleFS instance 'lfs.hex', compresses the files and stores them in a zip archive at 'items.zip'.

Command Line Output Example



```
Reading LittleFS instance from lfs.hex...
Parsing file lfs.hex...
Zip archive created at "/Users/username/Desktop/items.zip".
DONE
```

Copyright © 2026 Silicon Laboratories. All rights reserved.

Batch Operations

# Batch Operations

Simplicity Commander includes MultiCommander, a tool for running sequences of Simplicity Commander commands (called *actions*) across multiple adapters in parallel. This capability is especially useful in manufacturing environments, where you need to configure and program many devices efficiently.

MultiCommander provides a user interface that lets you:

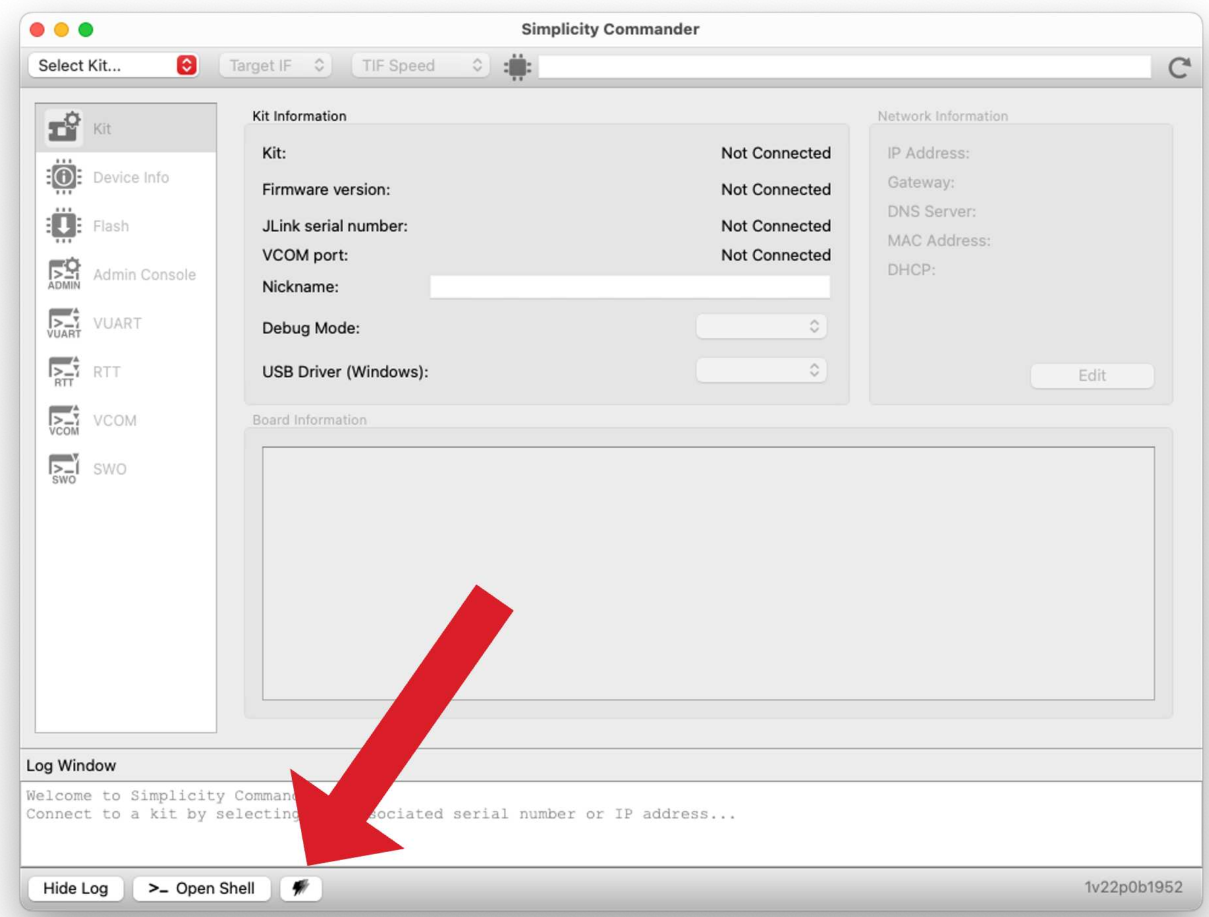
- Discover, add, and select the adapters you want to operate on.
- Define a sequence of actions to run on the selected adapters. Adapter-specific variables, such as serial number, IP address, and target device, give you flexibility when defining actions.
- Run the defined actions across all selected adapters in parallel. You can configure the number of concurrent operations.
- Monitor the progress and results of each action on each adapter in real time, and log output for later review.
- Save and load adapter and action configurations for reuse or sharing.
- Apply debug connection options that are used when connecting to target devices.

You can open MultiCommander from the Simplicity Commander installation by selecting the MultiCommander icon in the lower-left corner of the Simplicity Commander GUI. When you close MultiCommander, the current adapter and action configuration is saved and restored the next time you open the application, allowing you to continue where you left off.

You can also run MultiCommander as a standalone application by launching the MultiCommander executable from the Simplicity Commander installation directory. On macOS, MultiCommander is located inside the Simplicity Commander app bundle.

> Note: This guide includes screenshots of MultiCommander running on macOS. The user interface is functionally identical on all supported platforms, including Windows, macOS, and Linux.

Copyright © 2026 Silicon Laboratories. All rights reserved.




MultiCommander relies on Simplicity Commander being installed on your machine, which it uses as its backend for executing commands. As a result, even though MultiCommander is a separate executable, it cannot function unless Simplicity Commander is present. If you want MultiCommander to use a specific version of Simplicity Commander on your machine, you can configure this in the MultiCommander settings, under Settings > Set Path to Commander Executable....

As of the current Simplicity Commander version, MultiCommander is in a preview state and may be subject to changes in future releases.

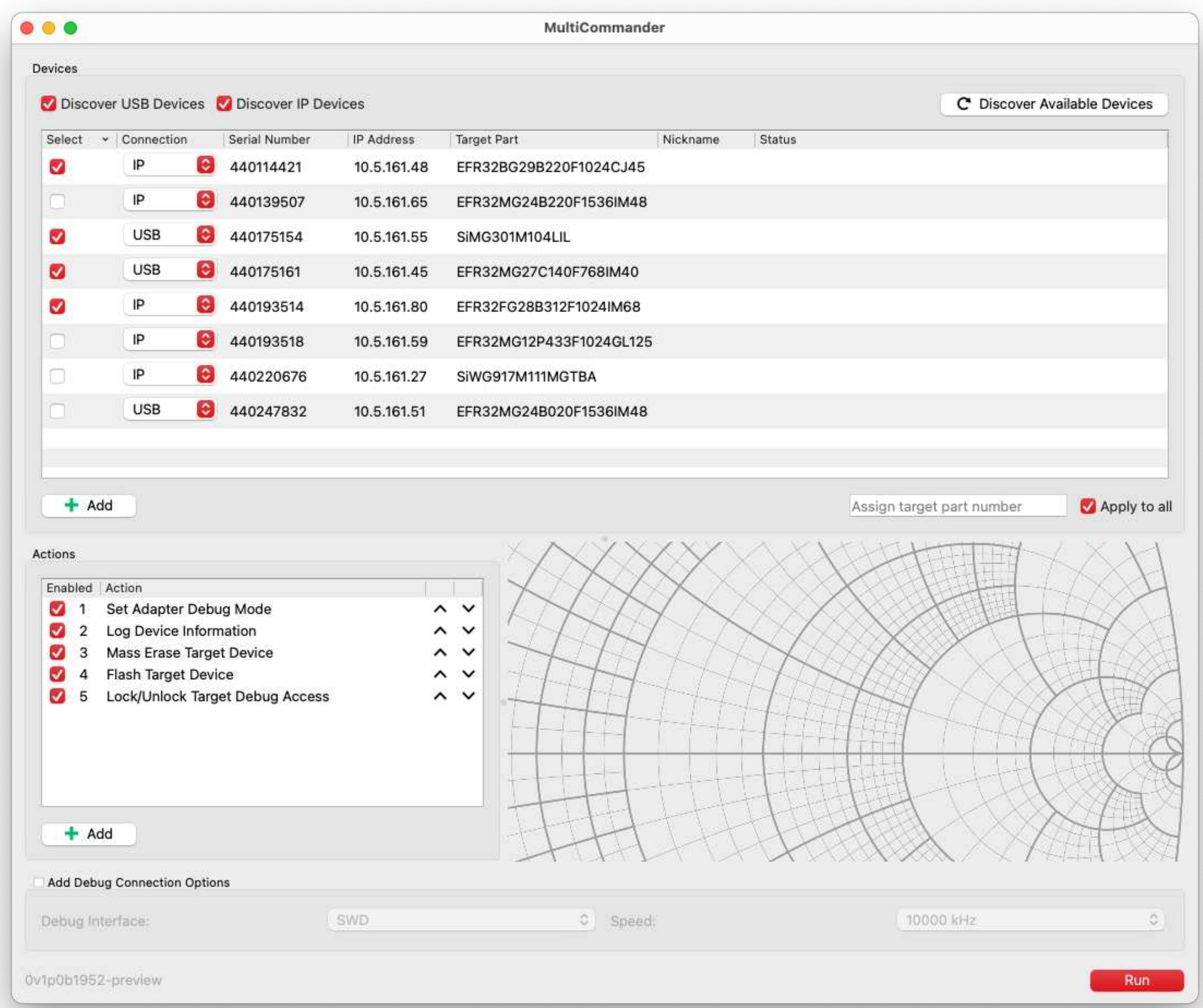
You can find the MultiCommander version number under Help > About MultiCommander. The version is also shown in the lower-left corner of the MultiCommander user interface.

Under the Help menu, you can also find links to this documentation as well to the Simplicity Link Adapter Firmware documentation.

For more details on using MultiCommander, see the following sections:

- Configuring Devices
- Defining Actions
- Applying Debug Connection Options
- Executing a Batch Job
- Importing and Exporting a Batch Job

Copyright © 2026 Silicon Laboratories. All rights reserved.




Copyright © 2026 Silicon Laboratories. All rights reserved.

# Configuring Devices

## Configuring Devices



The upper portion of the MultiCommander user interface is used to discover, select, and manage the adapters you want to operate on. Each adapter appears in a list with several columns that display key information, including:

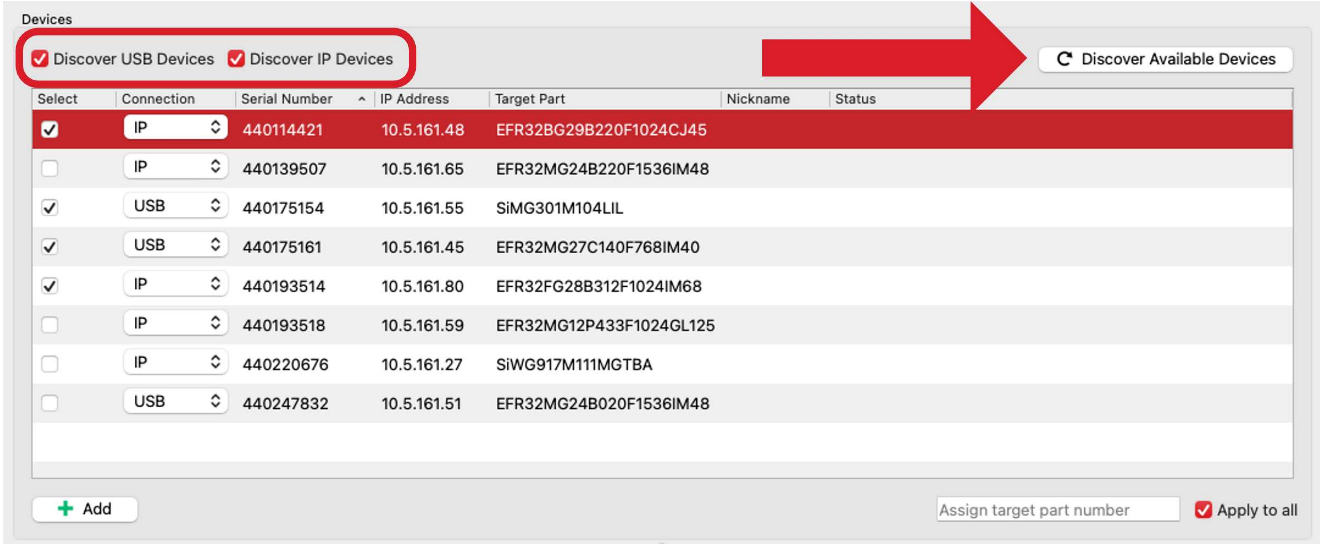
- Selection state, which indicates whether the adapter is selected for operations
- Connection type (USB or IP)
- Adapter serial number
- Adapter IP address
- Target part number
- Nickname
- Status of the last operation performed on the adapter

Copyright © 2026 Silicon Laboratories. All rights reserved.

You can edit the Serial Number, IP Address, and Target Part columns by double-clicking the corresponding cells.

The device list supports sorting by column. Click a column header to organize and locate adapters based on the displayed information.

## Discovering Available Adapters

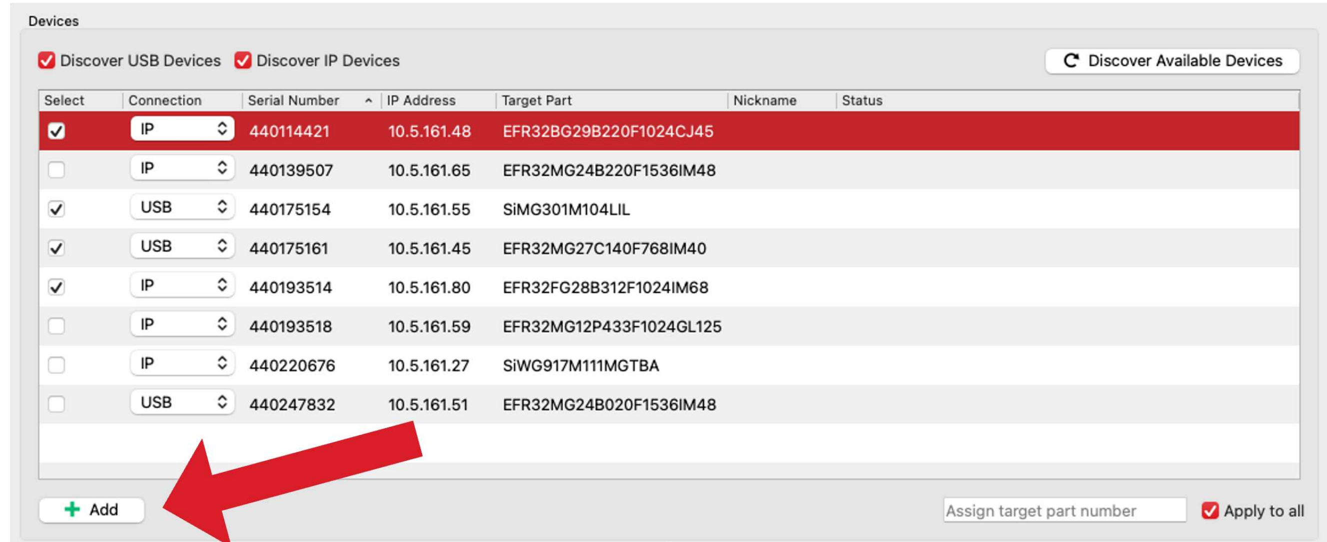



To discover available adapters, click the Discover Available Devices button located above the list of adapters. MultiCommander will scan for connected adapters and populate the list with any devices found. Please note that during the discovery process, *any existing adapters in the list that are not selected will be removed.*

By default, MultiCommander discovers devices connected over USB and devices accessible over the network, provided they are on the same subnet as your machine. To exclude devices with a specific connection method (such as USB or IP), clear the corresponding checkbox above the adapter list.

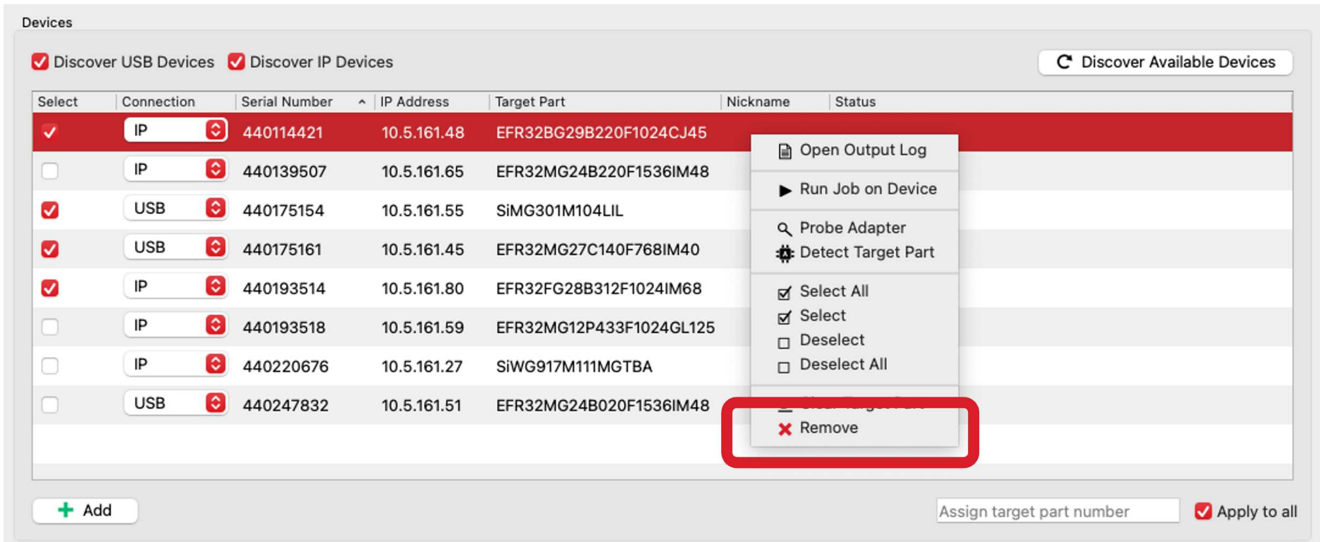
## Adding Adapters Manually

Copyright © 2026 Silicon Laboratories. All rights reserved.




You can also add adapters manually by clicking Add below the device list. This adds an empty row to the list, where you can double-click individual columns to edit values. Manual entry is useful for IP-connected adapters that cannot be discovered automatically.

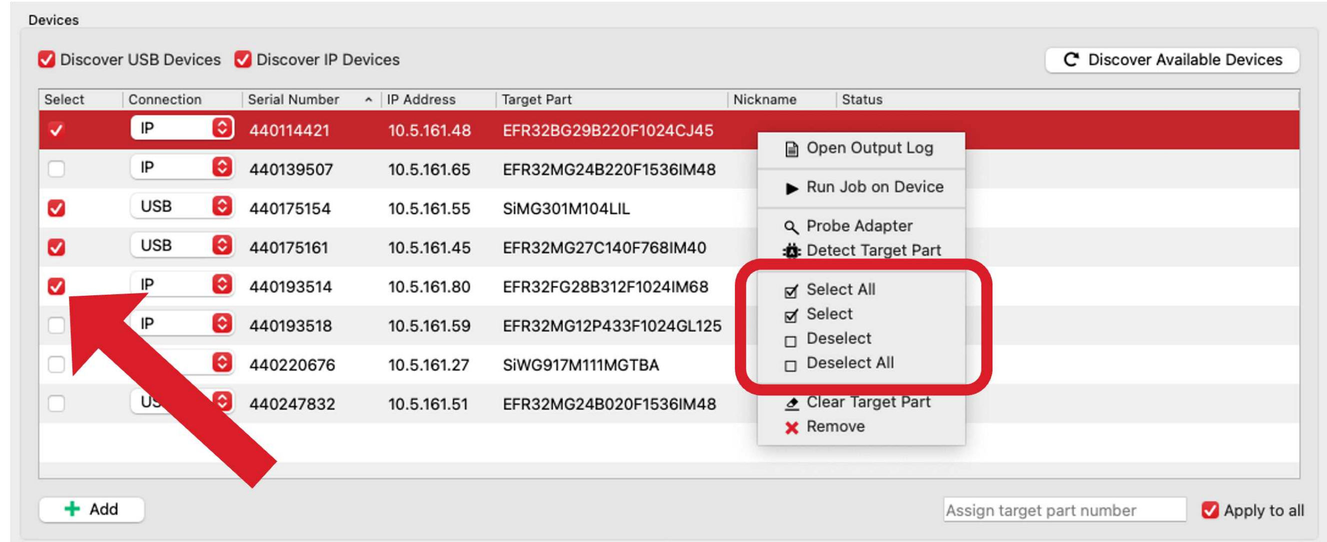
## Removing Adapters




To remove adapters, right-click one or more highlighted adapters and select Remove from the context menu. To remove multiple adapters, select them by holding Ctrl (Windows/Linux), Cmd (macOS), or Shift, and then right-click to remove them.

## Selecting Adapters for Operations

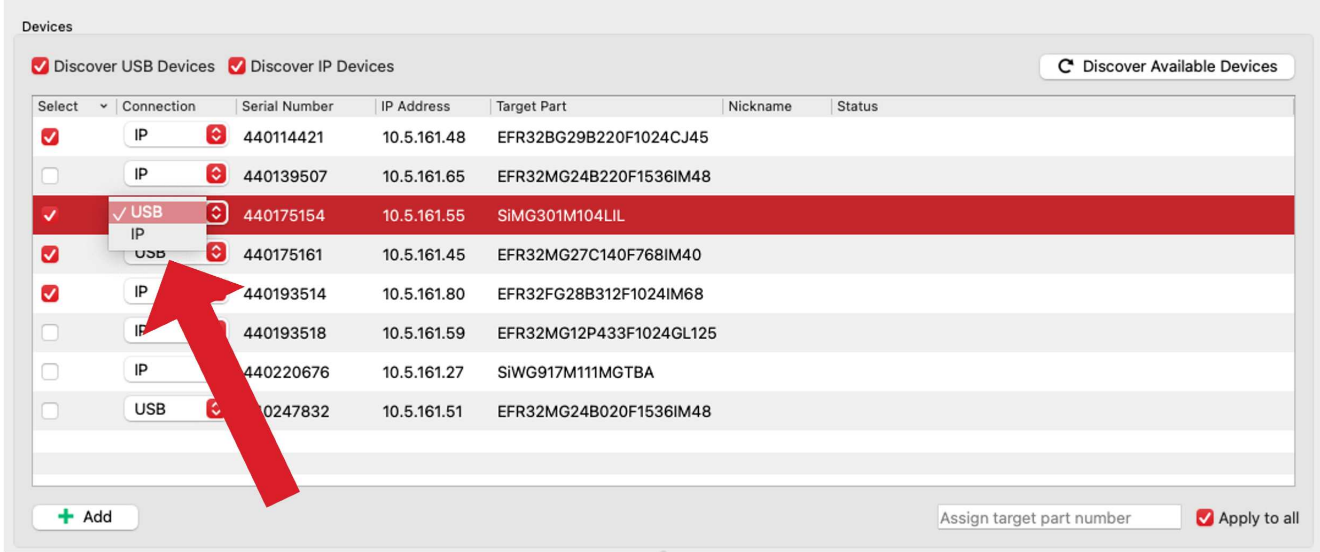
Copyright © 2026 Silicon Laboratories. All rights reserved.




Each adapter includes a checkbox in the first column that controls whether it is selected for operations. You can change the selection state by clicking the checkbox or by right-clicking the adapter and selecting Select or Deselect from the context menu.

Only selected adapters are included when you execute batch operations.

## Selecting Connection Type

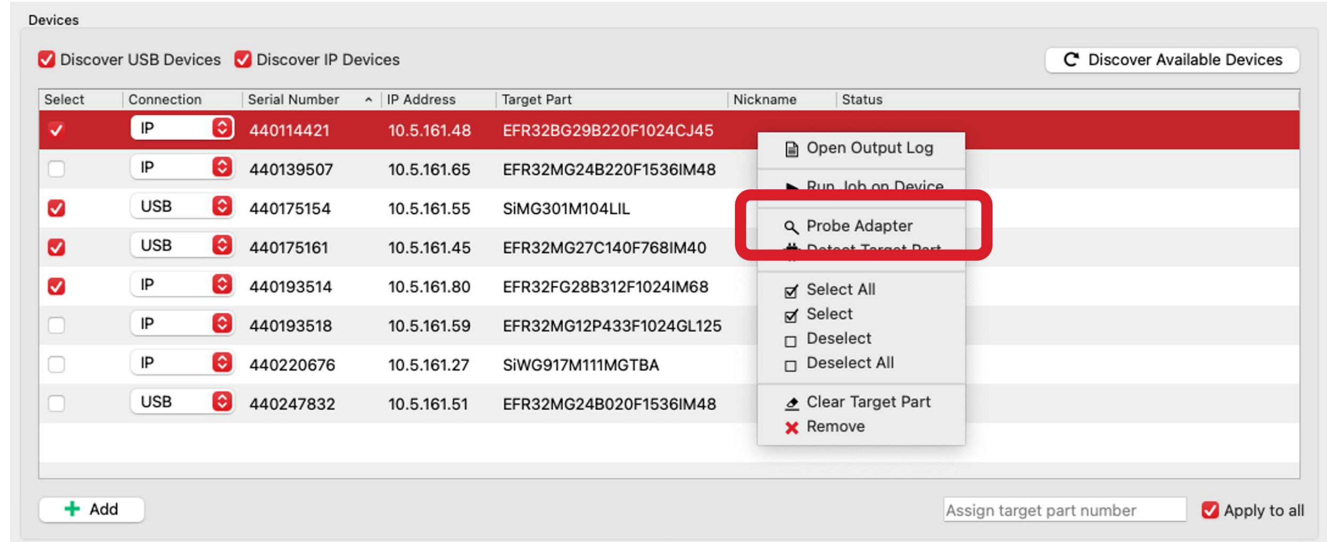



The Connection column indicates how MultiCommander connects to an adapter. If only a serial number is provided, the connection type is set to USB. If only an IP address is provided, the connection type is set to IP.

If both a serial number and an IP address are provided, you can select the desired connection type using the drop-down menu in the Connection column.

## Probing Adapters

Copyright © 2026 Silicon Laboratories. All rights reserved.

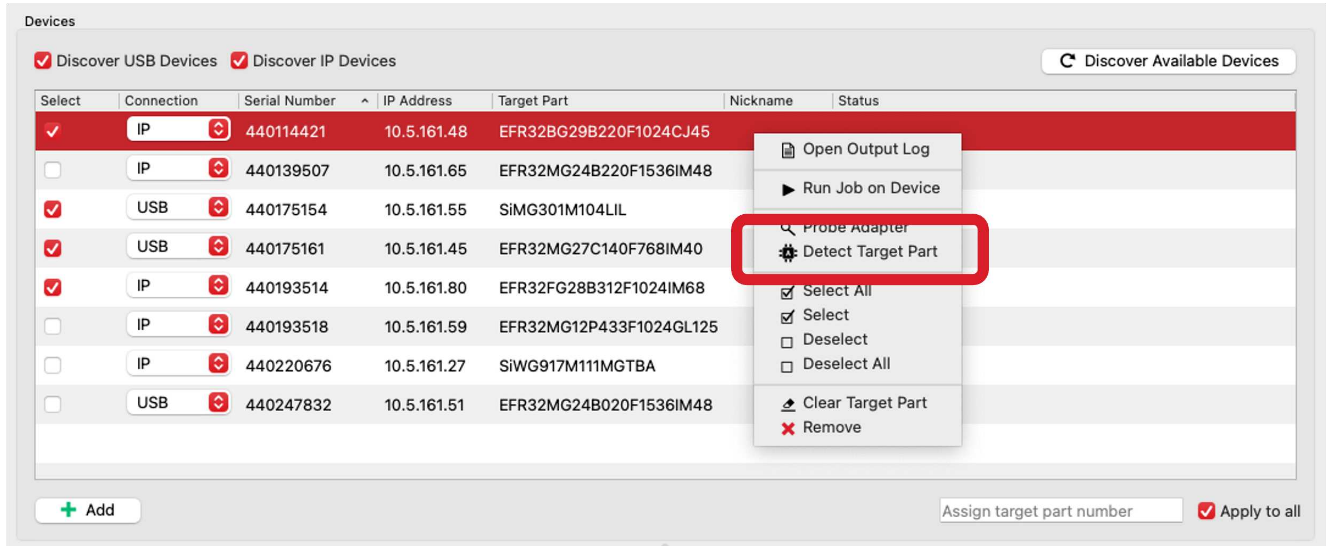



Right-clicking an adapter provides the Probe Adapter option. This option uses Simplicity Commander to connect to the adapter using the selected connection type and query its serial number, IP address, target part, and nickname. The adapter entry in the list is then updated with the retrieved information.

Probing is useful if you manually added an adapter with incomplete information or if you want to refresh the details for an existing adapter. If you are using custom hardware, the target part may not be identified correctly during probing.

Note: Probing adapters only works with Silicon Labs adapters.

## Detecting Target Parts

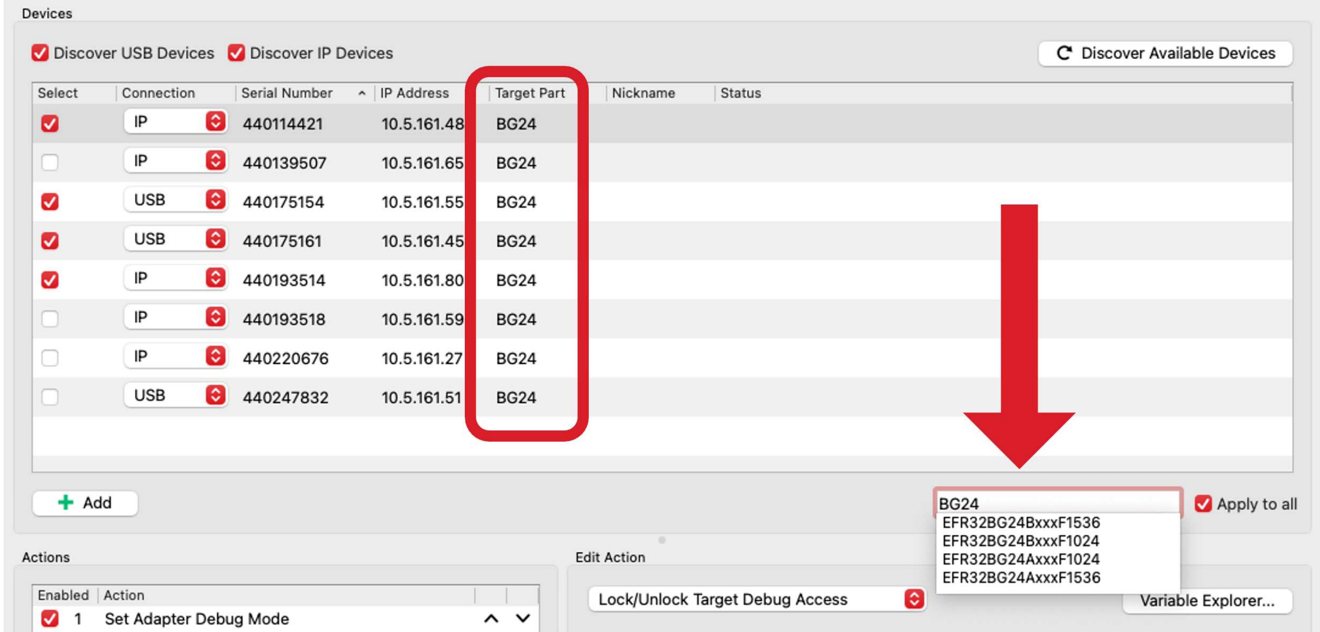



Right-clicking an adapter also provides the Detect Target Part option. This option uses Simplicity Commander to query the connected target device, rather than the adapter, for its part number and updates the Target Part column.

Copyright © 2026 Silicon Laboratories. All rights reserved.

If you are using custom hardware, you can enter a partial or expected part number in the Target Part column before detection. Simplicity Commander uses this hint to help identify the correct part. For example, for an EFR32MG26B510F3200IM48 device, entering `EFR32MG26` is sufficient.

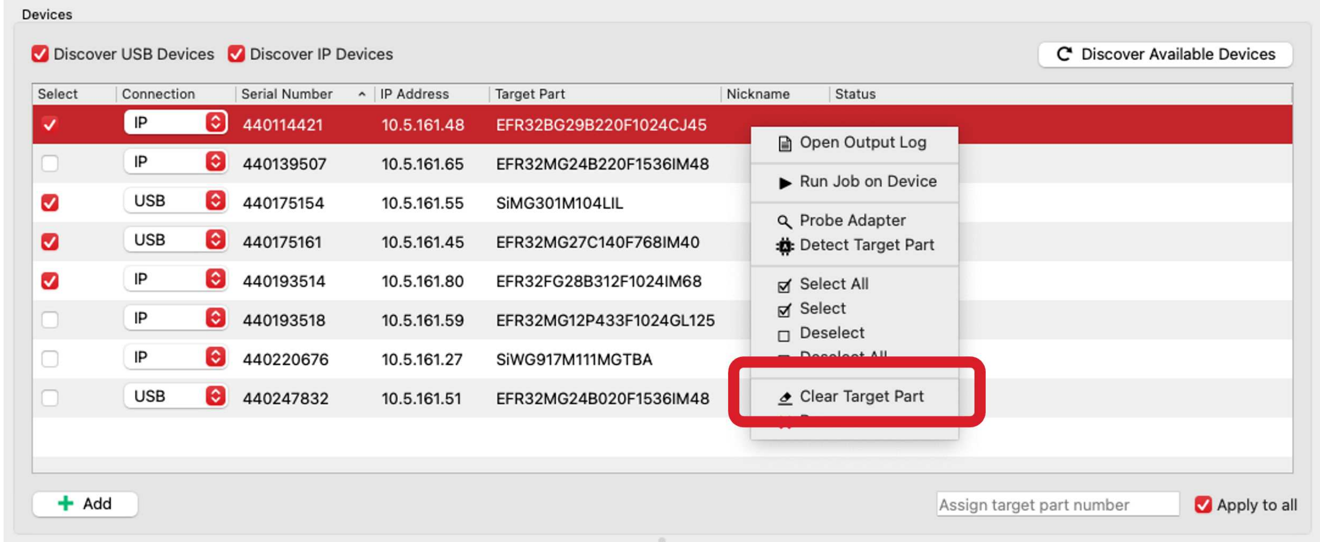
## Manually Assigning Target Parts




Below the adapter list is a text box labeled Assign target part number. Use this field to assign a target part number to multiple adapters at once.

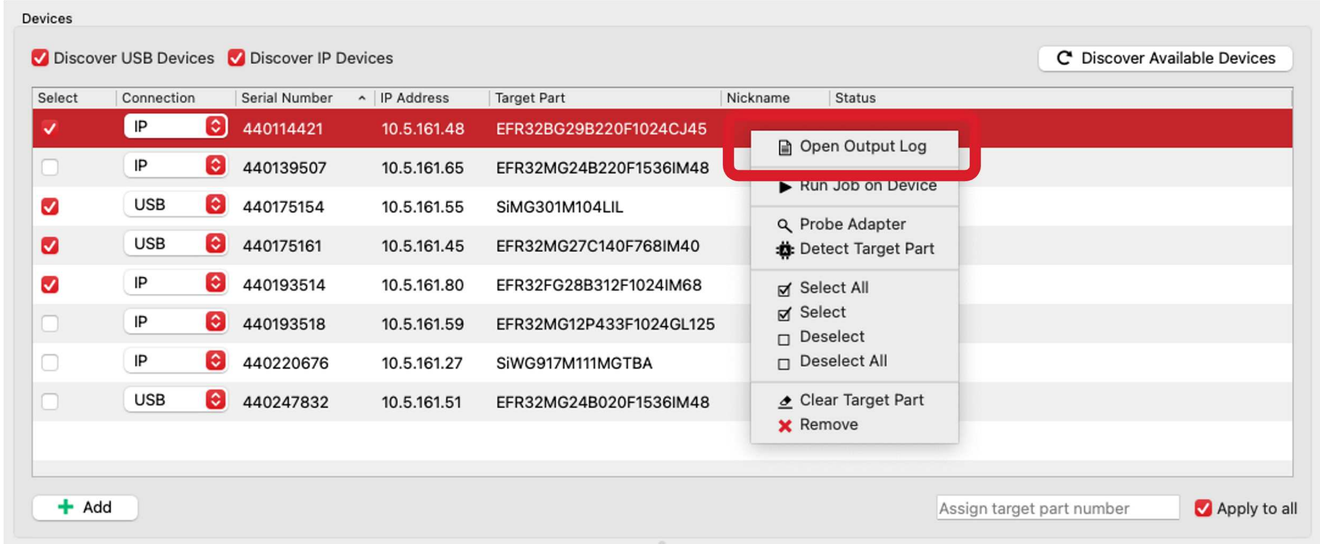
If the Apply to all checkbox is selected, the specified part number is assigned to all adapters in the list. If the checkbox is not selected, the part number is assigned only to the currently highlighted adapters.

The text box supports auto-completion and suggests matching part numbers based on the part numbers known to Simplicity Commander.



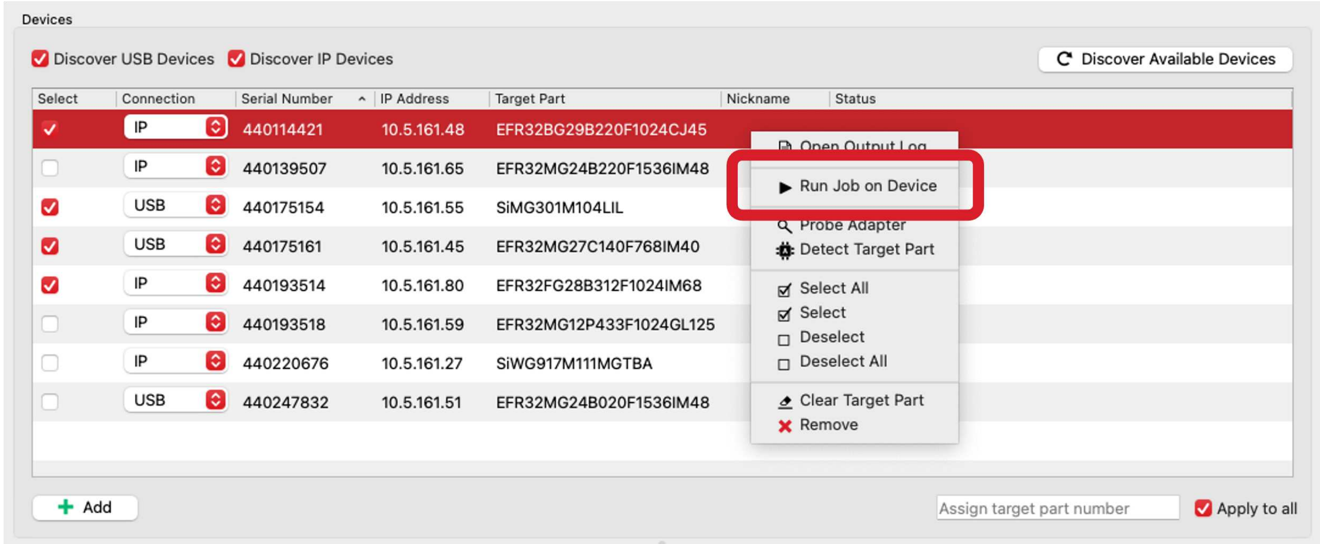
You can also right-click an adapter and select Clear Target Part to remove the assigned part number from that adapter.

## Open Output Log




If logging is enabled in the MultiCommander settings and a log file exists, you can open the output log for a specific adapter by right-clicking the adapter and selecting Open Output Log. The log opens in your default text editor, allowing you to review the detailed output of the actions performed on that adapter.

## Run Job on Singular Devices




Right-clicking an adapter provides the Run Job on Device option. This option runs the currently defined batch job on that adapter only, regardless of its selection state.

To run the job on multiple adapters, select them by holding Ctrl (Windows/Linux), Cmd (macOS), or Shift, and then right-click to run the job on all selected devices.

Copyright © 2026 Silicon Laboratories. All rights reserved.

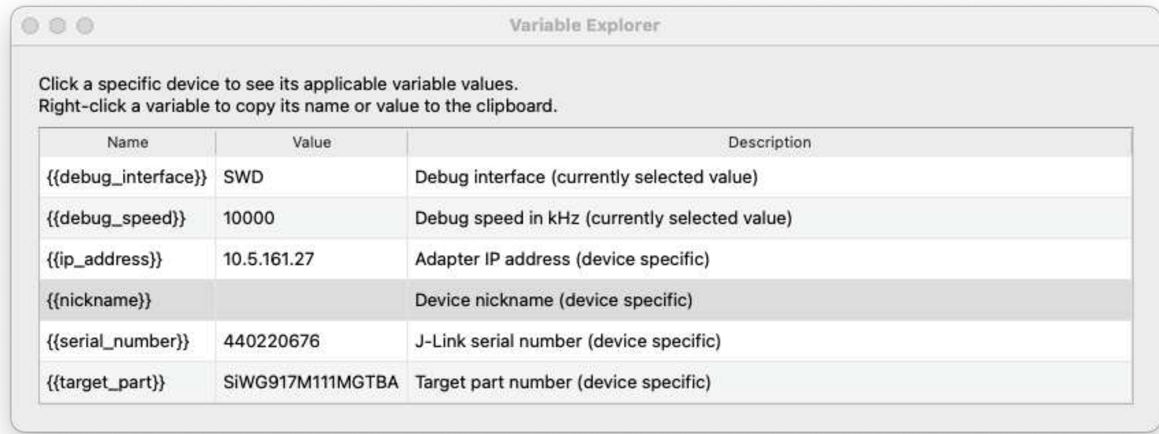
# Defining Actions

# Defining Actions

The lower portion of the MultiCommander user interface is used to define the sequence of actions to perform on the selected adapters. In this section, you can add, configure, and manage actions that are executed in order on each selected adapter.

## Variables

Copyright © 2026 Silicon Laboratories. All rights reserved.




MultiCommander supports a set of device-specific variables that you can use in actions. These variables allow you to create flexible action sequences that adapt to each adapter based on properties such as serial number, IP address, target device, or nickname.

The supported variables include:

- `{{serial_number}}` : The adapter serial number.
- `{{ip_address}}` : The adapter IP address.
- `{{target_part}}` : The part number of the target device connected to the adapter.
- `{{nickname}}` : The nickname assigned to the adapter.
- `{{debug_interface}}` : The debug interface type of the adapter (for example, SWD or JTAG).
- `{{debug_speed}}` : The debug speed configured for the adapter.

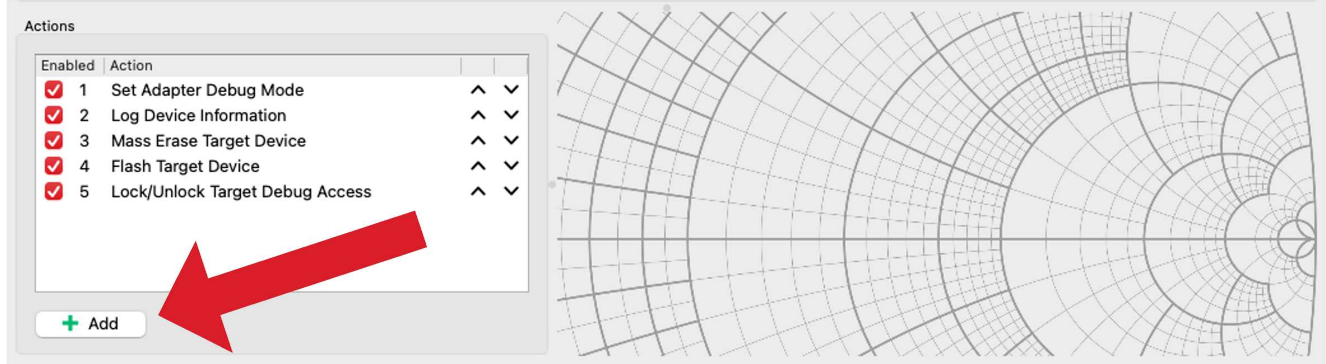
You can view all available variables in the Variable Explorer, which can be opened by clicking Variable Explorer... located next to the action selection drop-down. When you select an adapter in the device list, the explorer shows the variable values for that adapter.

Right-clicking a variable in the explorer lets you copy either the variable name or the variable value for the selected adapter to the clipboard.

Any actions that support text input fields will allow you to use these variables by including them in double curly braces (for example, `{{serial_number}}` ). During execution, MultiCommander replaces each variable with the corresponding values for each adapter.

# Adding Actions

Copyright © 2026 Silicon Laboratories. All rights reserved.




To add an action, click Add below the actions list. A box opens to the right, where you can choose an action from the drop-down list.

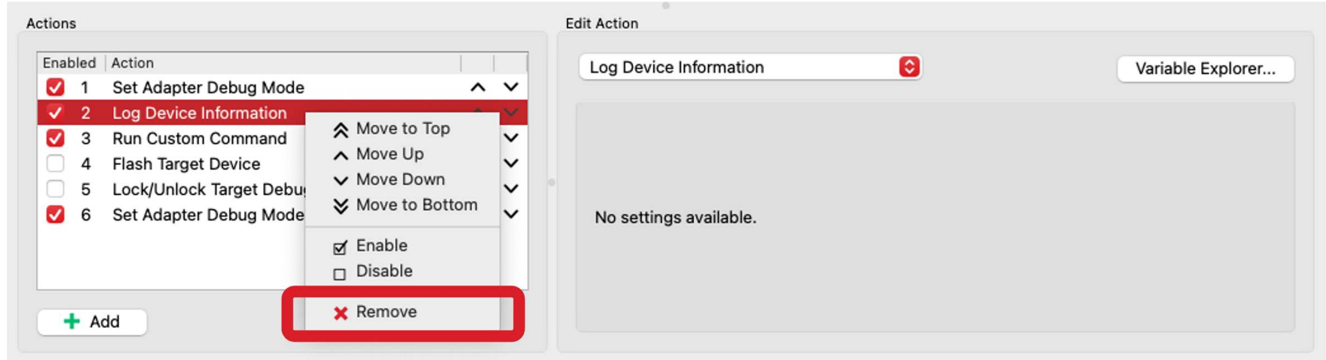
Available actions include:

- Log Device Information
- Flash Target Device
- Mass Erase Target Device
- Lock/Unlock Target Device
- Recover Target Device
- Wait
- Run Custom Command
- Install Adapter Firmware
- Set Adapter Debug Mode

Each action has its own configuration options. When you select an action from the list, its options appear in the box on the right.

You can add an action to the list multiple times, and there is no limit to the number of actions you can define.

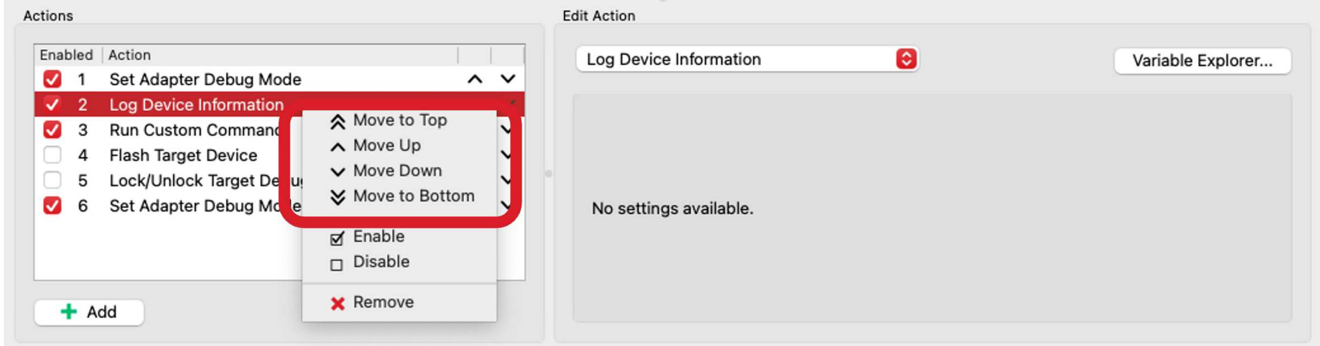
## Removing Actions




To remove actions from the list, right-click one or more highlighted actions and select Remove from the context menu. You can highlight multiple actions by holding down the Ctrl/Cmd or Shift keys while clicking rows in the list.

## Reordering Actions

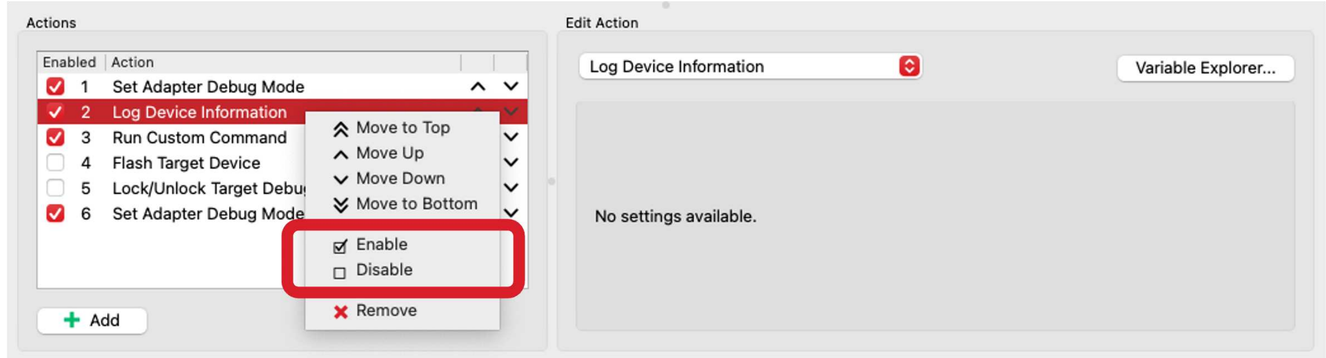
Copyright © 2026 Silicon Laboratories. All rights reserved.




You can reorder actions by selecting an action and using the arrow buttons next to each list item. The up arrow moves the selected action one position higher, and the down arrow moves it one position lower.

You can also right-click an action to view additional options to move the action to the top or bottom of the list. Multiple actions can be selected and moved at the same time.

## Enabling/Disabling Actions

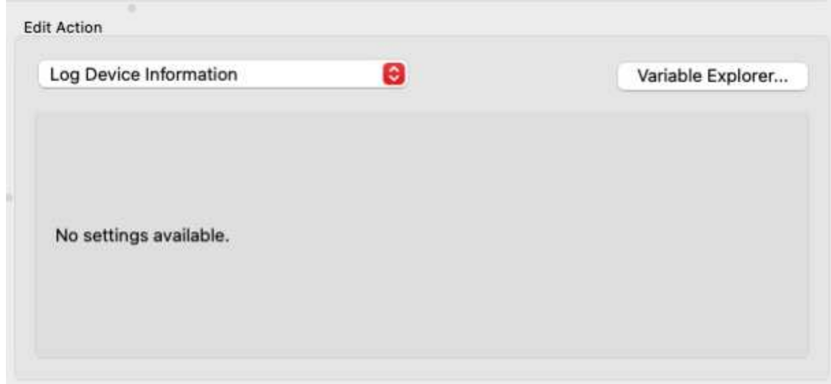



Each action includes a checkbox that indicates whether it is enabled. You can enable or disable an action by clicking the checkbox or by right-clicking the action and selecting Enable or Disable.

Disabled actions are skipped during execution, which allows you to modify the sequence of operations without removing actions.

## Action: Log Device Information

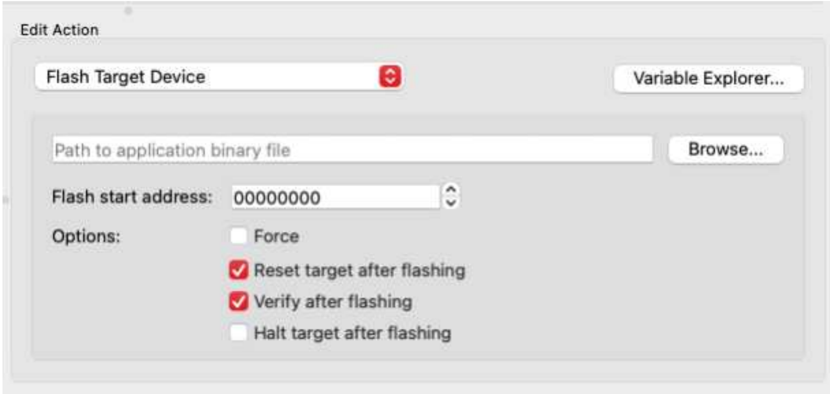
Copyright © 2026 Silicon Laboratories. All rights reserved.




The Log Device Information action retrieves and logs information about the connected target device. Logged information includes the part number, flash size, RAM size, unique ID, and other details.

This action requires no additional configuration.

## Action: Flash Target Device



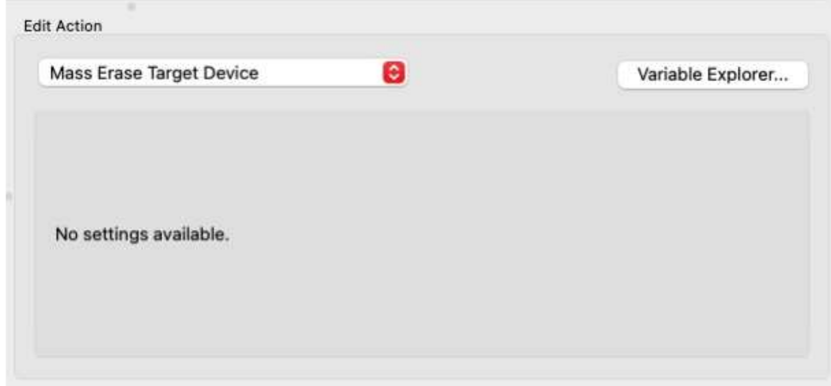

The Flash Target Device action allows you to program the connected target device with a specified application file.

You must provide the path to the application file. For binary files, you must also specify the flash address where the file is written. Additional options are available, such as resetting the device after flashing or verifying the flash contents.

This action supports variables in the file path field.

## Action: Mass Erase Target Device

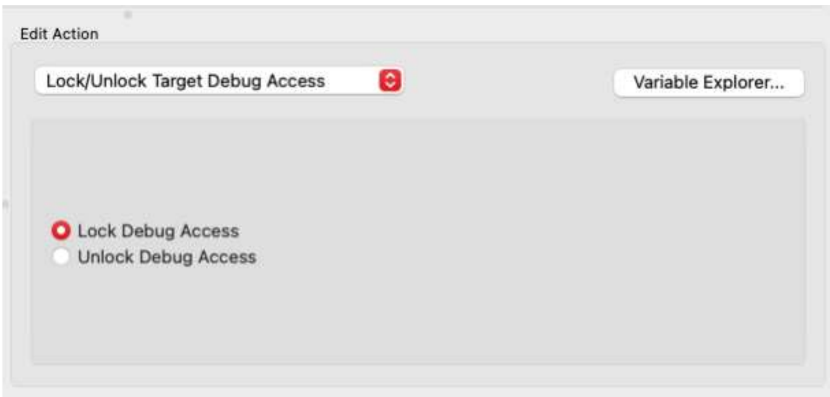
Copyright © 2026 Silicon Laboratories. All rights reserved.




The Mass Erase Target Device action performs a mass erase on the target device connected to the adapter.

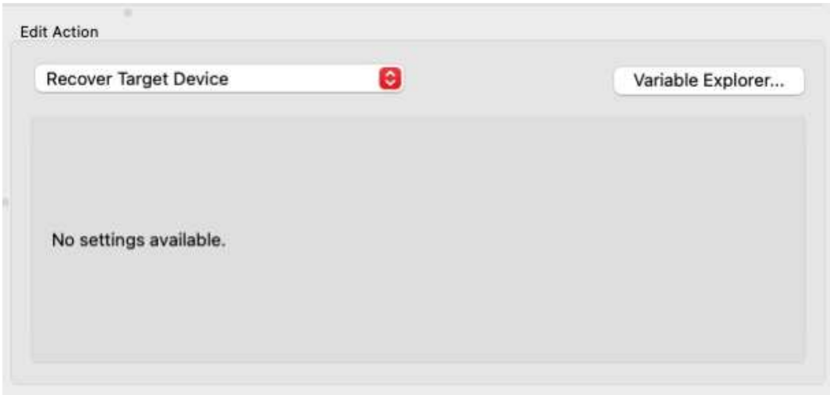
This action requires no additional configuration.

## Action: Lock/Unlock Target Device




The Lock/Unlock Target Device action lets you enable or disable debug access on the target device connected to the adapter. Use the radio buttons to choose whether to lock or unlock the device.

## Action: Recover Target Device

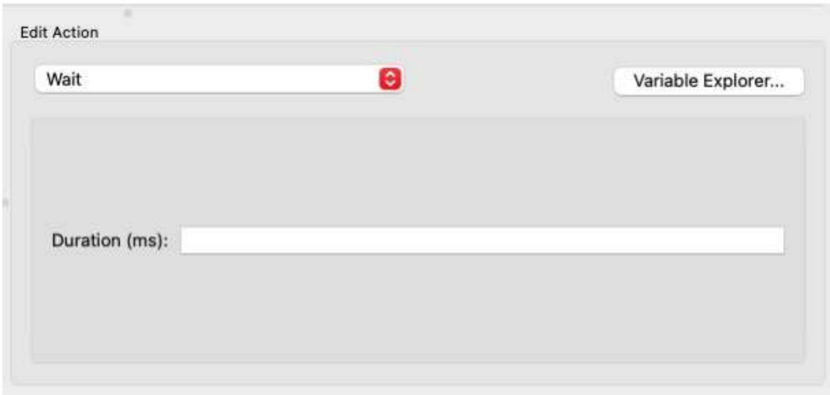



Copyright © 2026 Silicon Laboratories. All rights reserved.

The Recover Target Device action attempts to recover a target device that is in a locked or *bricked* state.

This action requires no additional configuration.

## Action: Wait

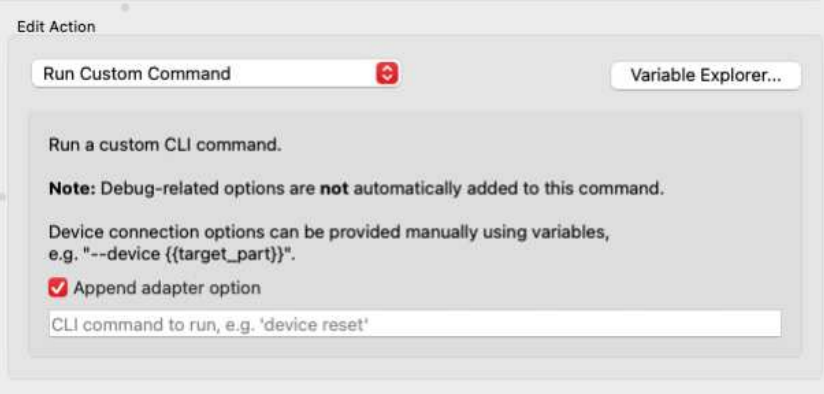



The Wait action inserts a delay into the action sequence. You specify the wait duration in milliseconds.

This can be useful when an operation requires additional time to complete before proceeding to the next action.

The maximum allowed wait time is 10 minutes (600,000 milliseconds).

## Action: Run Custom Command




The Run Custom Command action executes a custom Simplicity Commander command on the adapter. You specify the full command line, excluding the `commander` executable, including all options and arguments.

If the custom command is to be run on an adapter, it is recommended to leave Append adapter option enabled, as MultiCommander automatically appends the appropriate adapter connection option (such as `--serialno <serial number>` or `--ip <ip address>`), based on the selected connection type for each adapter.

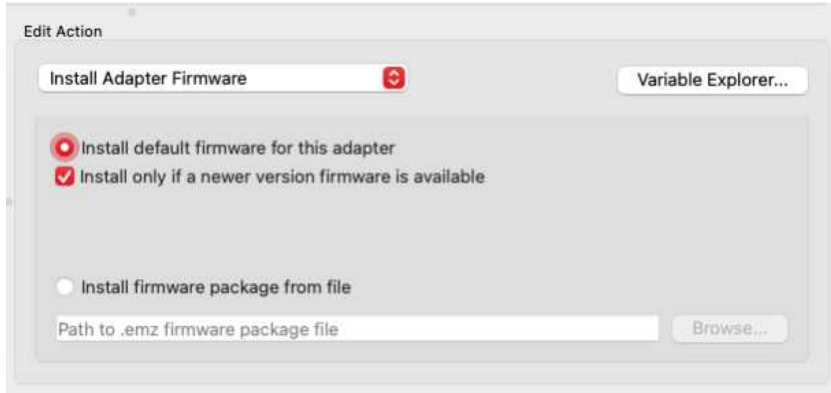
For host-only commands that do not require an adapter, you can disable this option.

Copyright © 2026 Silicon Laboratories. All rights reserved.

> Note: When running custom commands, debug connection options are not automatically applied. If needed, you must include them manually in the command line.

This action supports variables in the command line field.

## Action: Install Adapter Firmware



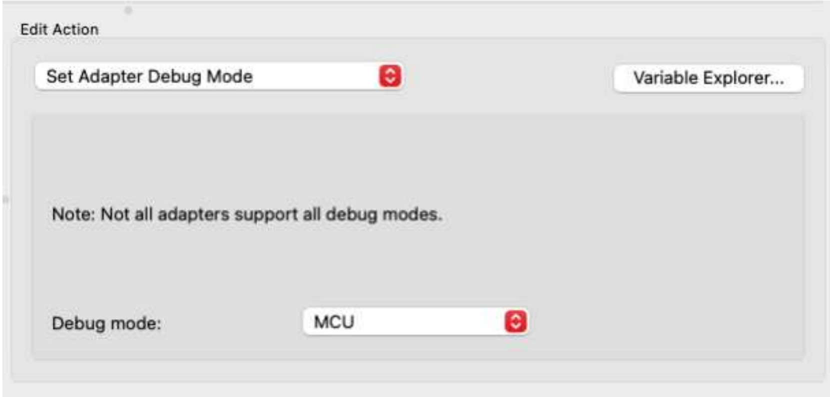

> Note: This action only applies to Silicon Labs adapters.

The Install Adapter Firmware action updates the adapter firmware. By default, MultiCommander installs the latest available firmware version only if the installed version is older.

You can force installation regardless of the current firmware version or specify a custom firmware file by providing a file path.

This action supports variables in the firmware file path field.

## Action: Set Adapter Debug Mode




Copyright © 2026 Silicon Laboratories. All rights reserved.

> Note: This action only applies to Silicon Labs adapters.

The Set Adapter Debug Mode action configures the adapter debug mode. Available options include:

- MCU
- IN
- OUT
- MINI
- OFF

Not all adapters support all debug modes. See Simplicity Link Adapter documentation for more information.

Copyright © 2026 Silicon Laboratories. All rights reserved.

# Applying Debug Connection Options

# Applying Debug Connection Options

The lower portion of the MultiCommander user interface includes options for configuring debug connection settings. You can specify the debug interface type, such as SWD or JTAG, and the debug speed in kHz. If you select the Add Debug Connection Options checkbox, these settings are applied to all adapters during command execution.

The currently set debug interface and speed are available as variables ( `{{debug_interface}}` and `{{debug_speed}}` ) that can be used in actions. These variables are available regardless of whether the Add Debug Connection Options checkbox is selected.

Debug connection options are *never* applied to the Run Custom Command action. If you want to use debug connection options in a custom command, you must include them manually by adding the appropriate Simplicity Commander command-line options and variable names, for example:

Copyright © 2026 Silicon Laboratories. All rights reserved.

`--tif {{debug_interface}} --speed {{debug_speed}}` .

Copyright © 2026 Silicon Laboratories. All rights reserved. 

Executing a Batch Job

# Executing a Batch Job

After you select the desired adapters and define the sequence of actions, you can execute the batch job by clicking Run in the lower-right corner of the MultiCommander user interface.

While the batch job is running, the status of each adapter is updated in real time to reflect the progress of the executing actions. Each adapter displays its current action as the job progresses, and a progress bar at the bottom of the interface shows the overall completion status across all selected adapters.

If logging is enabled in the MultiCommander settings, the output of each action for each adapter is written to a separate log file. After the batch job completes, you can review these logs by right-clicking an adapter in the list and selecting Open Output Log from the context menu.

Jobs that have not yet started can be aborted by clicking Abort, which replaces the Run button during execution. Aborting stops any further actions from running on adapters that have not started processing. Adapters that are already executing actions continue until completion or until an error occurs.

Copyright © 2026 Silicon Laboratories. All rights reserved.

MultiCommander

Devices

Discover USB Devices | Discover IP Devices | Discover Available Devices

| Select | Connection | Serial Number | IP Address | Target Part | Nickname | Status |
|---|---|---|---|---|---|---|
| ☐ | IP | 440114421 | 10.5.161.48 | EFR32BG29B220F1024CJ45 | | |
| ☑ | IP | 440139507 | 10.5.161.65 | EFR32MG24B220F1536IM48 | | DONE (elapsed: 1.56 s) |
| ☐ | USB | 440175154 | 10.5.161.55 | SiMG301M104LIL | | |
| ☐ | USB | 440175161 | 10.5.161.45 | EFR32MG27C140F768IM40 | | |
| ☐ | IP | 440193514 | 10.5.161.80 | EFR32FG28B312F1024IM68 | | |
| ☑ | IP | 440193518 | 10.5.161.59 | EFR32MG12P433F1024GL125 | | DONE (elapsed: 1.47 s) |
| ☑ | IP | 440220676 | 10.5.161.27 | SiWG917M111MGTBA | | DONE (elapsed: 1.64 s) |
| ☑ | USB | 440247832 | 10.5.161.51 | EFR32MG24B020F1536IM48 | | DONE (elapsed: 1.37 s) |

+ Add | Assign target part number | Apply to all

Actions

| Enabled | | Action |
|---|---|---|
| ☑ | 1 | Set Adapter Debug Mode |
| ☑ | 2 | Log Device Information |
| ☑ | 3 | Run Custom Command |
| ☐ | 4 | Flash Target Device |
| ☐ | 5 | Lock/Unlock Target Debug Access |
| ☑ | 6 | Set Adapter Debug Mode |

+ Add

Edit Action

Set Adapter Debug Mode | Variable Explorer...

Note: Not all adapters support all debug modes.

Debug mode: MCU

Add Debug Connection Options

Debug Interface: SWD | Speed: 10000 kHz

0v1p0b1952-preview | Run

After the batch job finishes, the status of each adapter indicates whether the operations completed successfully or if an error occurred. The elapsed time for each adapter is also displayed in the status column.

Copyright © 2026 Silicon Laboratories. All rights reserved.

If an error occurs, the Status field identifies the action that failed. Hovering over the Status field displays a tooltip with additional error details. You can also open the output log for the adapter to review more detailed information about the failure.

## Configuring Number of Concurrent Jobs

You can configure the maximum number of concurrent jobs that MultiCommander will execute in the settings, under Settings > Set Max Concurrent Processes.... By default, this value is set to the total number of available CPU cores, including both physical and logical cores. You can adjust this value based on your system capabilities and performance requirements.

## Configuring Process Timeouts

You can configure the timeout for individual actions in the settings under Settings > Set Process Timeout.... By default, the timeout is set to 2 minutes and can be increased to a maximum of 60 minutes.

## Configuring Logging

Logging is enabled by default in MultiCommander. When logging is enabled, output from actions executed on each adapter is saved to a log file. You can disable logging by clearing the checkbox under Logging > Enable

Copyright © 2026 Silicon Laboratories. All rights reserved.

Logging.

From the Logging menu, you can also open the current log directory by selecting Logging > Open Log File Directory or change the directory by selecting Logging > Set Log File Directory....

Log files are named using the adapter serial number when available, or the IP address if no serial number is present. Logs include timestamps for the start and end of each job, as well as the full output from each executed action. Log files are appended, so repeated batch jobs on the same adapter are included in the same log file.

Copyright © 2026 Silicon Laboratories. All rights reserved.

Importing and Exporting a Batch Job

# Importing and Exporting a Batch Job

MultiCommander job files ( `.mcj` ) can be imported (loaded) and exported (saved) using the File menu in the MultiCommander user interface. The `.mcj` files contain the complete list of adapters and the defined sequence of actions, making it easy to reuse or share batch job configurations.

The exported file includes adapter connection details, such as serial numbers, IP addresses, and nicknames. Use caution when sharing `.mcj` files that may contain sensitive information.

Note: Files referenced by actions, such as firmware images used for programming, are *not* included in the exported `.mcj` file.

## Exporting a Batch Job

To export a batch job, select File > Save Job... from the menu. This will open a file dialog where you can choose the location and name for the `.mcj` file. This action saves the current configuration of adapters and actions to a `.mcj` file.

## Importing a Batch Job

To import a batch job, select File > Load Job.... A file dialog opens, allowing you to select an existing `.mcj` file. Loading a job replaces the current adapter and action configuration with the configuration defined in the selected file.

Copyright © 2026 Silicon Laboratories. All rights reserved.