tiflash Documentation

Release 1.3.0

Cameron Webb

Jul 10, 2022

Contents

An *unofficial* Python flash programmer for Texas Instruments Launchpads.

CHAPTER 1

Overview

TIFlash is a python/command-line interface for Texas Instruments' Code Composer Studio. It allows you to perform many of the operations possible in the CCS GUI via python or command-line. This can be extremely useful for automating tasks involving Texas Instruments microcontrollers or embedded processors. In addition, TIFlash makes it easier to perform simple actions like flashing, erasing or resetting a device without having to spin up an entire CCS GUI session.

1.1 Under the Hood

Under the hood TIFlash uses CCS's scripting interface (Debug Server Scripting) to interact with devices.

CHAPTER 2

Contents

2.1 Getting Started

2.1.1 Prerequisites

You will need to have Code Composer Studio installed along with drivers for any devices you plan to use (offered during installation of CCS)

You'll also need Python installed on your computer, either 2.7 or 3.6+ (preferred) will work.

2.1.2 Installing

Install TIFlash with `pip install tiflash`.

You can then do a quick test of your installation on the command line running: `tiflash info`

```
🚀 :~ cjwebb$ tiflash info
tiflash version:    1.2.7
release date:       2019.03.25
python version:     3.6.6
ccs version:        8.1.0.00011
ccs prefix:         /Applications/ti
ccs location:       /Applications/ti/ccsv8
device drivers:     CC2XXX,CC3XXX,MSP430,MSP432,TIVAC
```

Warning: You should see the path to your Code Composer Studio installation (if not see *Custom CCS Install Path*)

For more examples of running tiflash please see the *Examples page*

2.1.3 Custom Configurations

Custom CCS Install Path

Note: If you have CCS installed in the default directory, TIFlash should work out of box with no additional configurations.

If you installed CCS in a custom location, you'll need to provide the path of the CCS installation to TIFlash. You can do this one of two ways:

Option #1 Set an environment variable `CCS_PREFIX` to the directory of your installation(s):

Example:

If you have a specific version of CCS installed you want to use, set `CCS_PREFIX` to the full path of the installation: `CCS_PREFIX=/opt/ti/ccsv8`

If you have multiple CCS versions installed in the same directory, set `CCS_PREFIX` to the parent directory `CCS_PREFIX=/opt/ti` (TIFlash will automatically choose the latest)

Hint: The `CCS_PREFIX` environment variable option is nice when you have multiple CCS installations because it allows you to still use the `ccs` session arg for choosing a version of CCS to use.

```
tiflash.get_info(ccs="7")       # finds ccs version 7 in directory set by CCS_
↪PREFIX
tiflash.get_info(ccs="8.1")     # finds ccs version 8.1 in directory set by CCS_
↪PREFIX
```

Option #2 Pass the full path of the ccs folder as the `ccs` argument for every TIFlash command called.

Example:

```
tiflash.get_info(ccs="/opt/ti")          # TIFlash will automatically␣
↪select latest
tiflash.get_info(ccs="/opt/ti/ccsv8")    # Force TIFlash to use CCSv8
```

2.2 API Reference

Below is the TIFlash Python API.

2.2.1 Session Arguments

`**session_args`

`**session_args` are a set of keyword arguments (**kwargs) that specify how to connect to a device when running a command.

Each argument can be provided in any function that takes `**session_args` as a parameter. Just provide the particular argument(s) in the function call as a keyword argument:

```
# Example of providing session args: 'serno' and 'ccs'
function_name(serno="LXXXXXX", ccs="7")
```

Name	Type	Description	Default
serno	str	serial number of device	
devicetype	str	full devicetype name	
ccs	str	full path to ccs installation or version number	latest
chip	str	cpu/chip of device to connect to	first chip found for device
connection	str	full name of connection to use	default connection for device
ccxml	str	full path to ccxml file to use	
fresh	boolean	force a new ccxml file to be created and used	False
debug	boolean	output debug information when running	False
timeout	int	amount of time (seconds) for tiflash to execute	60

Note:

Name	Tips
device-type	you can see a list of devicetypes with the get_devicetypes() function
chip	you can see a list of chips/cpus with the get_cpus() function
connection	you can see a list of connections with the get_connections() function
ccxml	providing an existing ccxml file to use will eliminate any requirement of providing a serno, devicetype, and/or connection type

---

Warning: *At the very minimum you'll need to provide the device's serial number (serno) or device-type.*

TIFlash will try to determine the rest of the information from there. If a piece of information cannot be determined, an error will be raised and you'll need to provide this information as a session argument.

2.2.2 TIFlash

This module contains the core functions used for interacting with CCS. These functions are not specific to any particular device family and thus can be used on any device.

```
# All core functions are provided in the tiflash module
import tiflash
```

exception TIFlashAPIError
Bases: `tiflash.core.core.TIFlashError`

Generic TIFlash API Error

attach(*ccs=None*, ***session_args*)
Attach command; opens a CCS session and attaches to device.

Parameters

- **ccs** (*str*) – version number of CCS to use or path to custom installation
- **session_args** (**dict) – keyword arguments containing settings for the device connection

Raises `TIFlashError` – raises error when expression error is raised

detect_devices(*ccs=None*, ***session_args*)
Detect devices connected to machine.

Returns list of dictionaries describing connected devices

Return type list

erase(*options=None*, *ccs=None*, ***session_args*)
Erases device; setting 'options' before erasing device

Parameters

- **options** (*dict*) – dictionary of options in the format {option_id: option_val}; These options are set first before calling erase function.
- **ccs** (*str*) – version number of CCS to use or path to custom installation
- **session_args** (**dict) – keyword arguments containing settings for the device connection

Returns Result of erase operation (success/failure)

Return type bool

Raises `TIFlashError` – raises error if option invalid

evaluate(*expr*, *symbol_file=None*, *ccs=None*, ***session_args*)
Evaluates the given C/GEL expression

Parameters

- **expr** (*str*) – C or GEL expression
- **symbol_file** (*str*) – .out or GEL symbol file to load before evaluating

- **ccs** (*str*) – version number of CCS to use or path to custom installation
- **session_args** (**dict) – keyword arguments containing settings for the device connection

Returns String result from evaluating expression

Return type str

Raises TIFlashError – raises error when expression error is raised

flash(*image*, *binary=False*, *address=None*, *options=None*, *ccs=None*, ***session_args*)
Flashes device; setting 'options' before flashing device

Parameters

- **image** (*str*) – path to image to use for flashing
- **binary** (*bool*) – flashes image as binary if True
- **address** (*int*) – offset address to flash image
- **options** (*dict*) – dictionary of options in the format {option_id: option_val}; These options are set first before calling flash function.
- **ccs** (*str*) – version number of CCS to use or path to custom installation
- **session_args** (**dict) – keyword arguments containing settings for the device connection

Returns Result of flash operation (success/failure)

Return type bool

Raises TIFlashError – raises error if option invalid

get_connections(*ccs=None*, *search=None*)
Gets list of all connections installed on machine (ccs installation)

Parameters

- **search** (*str*) – String to filter connections by
- **ccs** (*str*) – version number of CCS to use or path to custom installation

Returns list of connection types installed in ccs

Return type list

Raises FindCCSError – raises exception if cannot find ccs installation

get_cpus(*ccs=None*, *search=None*)
Gets list of all cpus installed on machine (ccs installation)

Parameters

- **search** (*str*) – String to filter cpus by
- **ccs** (*str*) – version number of CCS to use or path to custom installation

Returns list of cpus types installed in ccs

Return type list

Raises FindCCSError – raises exception if cannot find ccs installation

get_devicetypes(*ccs=None*, *search=None*)
Gets list of all devicetypes installed on machine (ccs installation)

Parameters

- **search** (*str*) – String to filter devices by
- **ccs** (*str*) – version number of CCS to use or path to custom installation

Returns list of device types installed in ccs

Return type list

Raises `FindCCSError` – raises exception if cannot find ccs installation

get_info(*ccs=None*, ***session_args*)
Returns dict of information regarding tiflash environment

Parameters **ccs** (*str*) – version number of CCS to use or path to custom installation

Returns dictionary of information regarding tiflash environment

Return type dict

get_option(*option_id*, *pre_operation=None*, *ccs=None*, ***session_args*)
Reads and returns the value of the option_id.

Parameters

- **option_id** (*str*) – Option ID to request the value of. These ids are device specific and can viewed using TIFlash.print_options().
- **pre_operation** (*str*) – Operation to run prior to reading option_id.
- **ccs** (*str*) – version number of CCS to use or path to custom installation
- **session_args** (**dict) – keyword arguments containing settings for the device connection

Returns Value of option_id

Return type str

Raises `TIFlashError` – raises error if option does not exist

list_options(*option_id=None*, *ccs=None*, ***session_args*)
"Gets all options for the session device.

Parameters

- **option_id** (*str, optional*) – string used to filter options returned
- **ccs** (*str*) – version number of CCS to use or path to custom installation

Returns list of option dictionaries

Return type list(dict)

memory_read(*address*, *num_bytes=1*, *page=0*, *ccs=None*, ***session_args*)
Reads specified bytes from memory

Parameters

- **address** (*long*) – memory address to read from
- **num_bytes** (*int*) – number of bytes to read
- **page** (*int, optional*) – page number to read memory from
- **ccs** (*str*) – version number of CCS to use or path to custom installation

- **session_args** (**dict) – keyword arguments containing settings for the device connection

Returns Returns list of bytes read from memory

Return type list

memory_write(*address*, *data*, *page=0*, *ccs=None*, ***session_args*)
Writes specified data to memory

Parameters

- **address** (*long*) – memory address to read from
- **data** (*list*) – list of bytes to write to memory
- **page** (*int, optional*) – page number to read memory from
- **ccs** (*str*) – version number of CCS to use or path to custom installation
- **session_args** (**dict) – keyword arguments containing settings for the device connection

Raises TIFlashError – raises error when memory read error received

register_read(*regname*, *ccs=None*, ***session_args*)
Reads specified register of device

Parameters

- **regname** (*str*) – register name to read from
- **ccs** (*str*) – version number of CCS to use or path to custom installation
- **session_args** (**dict) – keyword arguments containing settings for the device connection

Returns value of register

Return type int

Raises TIFlashError – raised if regname is invalid

register_write(*regname*, *value*, *ccs=None*, ***session_args*)
Writes a value to specified register of device

Parameters

- **regname** (*str*) – register name to read from
- **value** (*int*) – value to write to register
- **ccs** (*str*) – version number of CCS to use or path to custom installation
- **session_args** (**dict) – keyword arguments containing settings for the device connection

Raises TIFlashError – raised if regname is invalid

reset(*options=None*, *ccs=None*, ***session_args*)
Performs a Board Reset on device

Parameters

- **options** (*dict*) – dictionary of options in the format {option_id: option_val}; These options are set first before calling reset function.
- **ccs** (*str*) – version number of CCS to use or path to custom installation

- **session_args** (**dict) – keyword arguments containing settings for the device connection

Returns True if reset was successful; False otherwise

Return type bool

set_option(*option_id*, *option_val*, *post_operation=None*, *ccs=None*, ***session_args*)
Sets the value of the option_id.

Parameters

- **option_id** (*str*) – Option ID to set value of. These ids are device specific and can viewed using TIFlash.print_options().
- **option_val** (*str or int*) – Value to set option to.
- **post_operation** (*str*) – Operation to run after to setting option_id.
- **ccs** (*str*) – version number of CCS to use or path to custom installation
- **session_args** (**dict) – keyword arguments containing settings for the device connection

Raises TIFlashError – raises error if option does not exist

verify(*image*, *binary=False*, *address=None*, *options=None*, *ccs=None*, ***session_args*)
Verifies device; setting 'options' before erasing device

Parameters

- **image** (*str*) – path to image to use for verifying
- **binary** (*bool*) – verifies image as binary if True
- **address** (*int*) – offset address to verify image
- **options** (*dict*) – dictionary of options in the format {option_id: option_val}; These options are set first before calling verify function.
- **ccs** (*str*) – version number of CCS to use or path to custom installation
- **session_args** (**dict) – keyword arguments containing settings for the device connection

Returns Result of verify operation (success/failure)

Return type bool

Raises TIFlashError – raises error if option invalid

xds110_list(*ccs=None*, ***session_args*)
Returns list of sernos and xds110 version numbers of connected XDS110 devices.

Parameters

- **ccs** (*str*) – version number of CCS to use or path to custom installation
- **session_args** (**dict) – keyword arguments containing settings for the device connection

Returns list of tuples (serno, version) of connected XDS110 devices

Return type list

Raises XDS110Error – raises if xdsdfu does not exist or fails

xds110_reset(*ccs=None*, ***session_args*)
Calls xds110reset command on specified serno.

Parameters

- **ccs** (*str*) – version number of CCS to use or path to custom installation
- **session_args** (**dict) – keyword arguments containing settings for the device connection

Returns True if xds110reset was successful

Return type bool

Raises

- TIFlashError – raises if serno not set
- XDS110Error – raises if xds110_reset fails

xds110_upgrade(*ccs=None*, ***session_args*)
Upgrades/Flashes XDS110 firmware on board.

Firmware flashed is found in xds110 directory (firmware.bin). This function uses the 'xdsdfu' executable to put device in DFU mode. Then performs the flash + reset functions of xdsdfu to flash the firmware.bin image

Parameters

- **ccs** (*str*) – version number of CCS to use or path to custom installation
- **session_args** (**dict) – keyword arguments containing settings for the device connection

Returns True if successful

Return type bool

Raises XDS110Error – raises if xds110 firmware update fails

Session Args

the session args are a common set of arguments used for most functions

TIFlash

the tiflash module contains all the core functionality of TIFlash.

2.3 Command-Line Interface

The command-line interface is a quick and easy way to interact with a device.

Session Arguments Arguments for specifying how to connect to a device

Commands Arguments for specifying actions to perform on a device

---

Note: Before getting into the various command-line arguments, it's important to understand the general format for commands.

```
# The typical format of a tiflash command is of the following:
tiflash [session arguments] <command> [command arguments]
```

tiflash Command for invoking the command-line tool.

session arguments These are the arguments provided to specify which device to connect to and how to connect to it. *At the very least you'll need to provide the device's serial number.*

See Session Arguments for a complete list of session arguments.

command This is the command or action to perform on the device.

See Commands for a complete list of commands.

command arguments These arguments are specific to each command. To see all possible options you can run the command with the -h help option.

You can view each command's specific arguments here Commands.

2.3.1 Session Arguments

There are a variety of ways for specifying a device to connect to. Below are a list of arguments you can provide for specifying how to connect to a device.

Note: You should always specify all session arguments *before* specifying the command you wish to execute.

```
usage: tiflash [session arguments] <command> [command arguments]
```

commands

cmd	Possible choices: options-get, options-set, options-list, list, reset, erase, verify, flash, memory-read, memory-write, register-read, register-write, evaluate, attach, xds110-reset, xds110-upgrade, xds110-list, detect, info

session arguments

-s, --serno	Serial number of device
-d, --devicetype	Devicetype of device
--ccs	Version of ccs to use (default is the latest)
--ccxml	CCXML (full path) file to use
--connection	Connection type to use for device
--chip	Device core to use
-t, --timeout	Timeout to use for command (seconds)
-F, --fresh	Generate new (fresh) ccxml
	Default: False
-D, --debug	Display debugging output
	Default: False
-A, --attach	Attach CCS to Device after performing action
	Default: False

-v, --version	print tiflash version
-V, --VERSION	print tiflash & python version

Note: Most often you'll only need to supply the serial number and TIFlash will try to automatically determine the default configurations (device type, connection, chip, etc.) to use when connecting to the device.

2.3.2 Commands

Reset

Reset a device. (Board Reset)

```
usage: tiflash [Session Arguments] reset [optionals]
```

Named Arguments

-o, --option	Sets an option before running reset cmd

Flash

Flash a device with an image(s).

```
usage: tiflash [Session Arguments] flash [optionals]
```

Positional Arguments

image	Image to flash.

Named Arguments

-b, --bin	Specify if image(s) are binary images Default: False
-a, --address	Address to begin flashing image(s)
-o, --option	sets an option before running flash cmd

Erase

Erase a device's flash.

```
usage: tiflash [Session Arguments] erase [optionals]
```

Named Arguments

-o, --option	Sets an option before running erase cmd

Verify

Verify an image on a device's flash.

```
usage: tiflash [Session Arguments] verify [optionals]
```

Positional Arguments

image Image to verify.

Named Arguments

-b, --bin Specify if image is a binary image

Default: False

-o, --option Sets an option before running verify cmd

Memory

Commands for reading/writing to a device's memory.

memory-read

Read from memory location on a device.

```
usage: tiflash [Session Arguments] memory-read <address> [optionals]
```

Positional Arguments

address Address in memory to read from

Named Arguments

-p, --page Page number in memory to access address

Default: 0

-n, --num Number of bytes to read

Default: 1

--hex Displays output in hex

Default: False

memory-write

Write to memory location on a device.

```
usage: tiflash [Session Arguments] memory-write <address> [optionals]
```

Positional Arguments

address Address in memory to write to

Named Arguments

-p, --page Page number in memory to access address

Default: 0

-d, --data **Bytes (hex) to write to memory.** Each byte separated by a space

memory-read

read from memory location in device's flash

memory-write

write to memory location in device's flash

Register

Commands for reading/writing to a device's register.

register-read

Read from register on a device.

```
usage: tiflash [Session Arguments] register-read <regname> [optionals]
```

Positional Arguments

regname Name of register to read.

Named Arguments

--hex Displays output in hex

Default: False

register-write

Write value to register on a device.

```
usage: tiflash [Session Arguments] register-write <reganame> <value>
```

Positional Arguments

regname Name of register to read.

value Value (32bit hex) to write to register.

register-read

read from register of device

register-write

write to register of device

Evaluate

Evaluate a C/GEL expression on a device.

```
usage: tiflash [Session Arguments] evaluate <expression> [optionals]
```

Positional Arguments

expression C or GEL expression to execute

Named Arguments

--symbols **.out or GEL symbol file to load before** evaluating expression.

XDS110

XDS110 connection commands

xds110-reset

Calls xds110reset on specified device

```
usage: tiflash [Session Arguments] xds110-reset
```

xds110-upgrade

Upgrades XDS110 firmware on device

```
usage: tiflash [Session Arguments] xds110-upgrade
```

xds110-list

Lists sernos of connected XDS110 devices

```
usage: tiflash [Session Arguments] xds110-list
```

xds110-reset

run xds110-reset command

xds110-upgrade

run xds110-upgrade command

xds110-list

run xds110-list command

Detect

Detect devices connected to machine

```
usage: tiflash [Session Arguments] detect
```

Options

Get/Set/List device specific options

Options-get

Get value of a device option.

```
usage: tiflash [Session Arguments] options-get <optionID> [optionals]
```

Positional Arguments

optionID Option ID to get value of.

Named Arguments

-op, --operation **Specify an operation to perform prior to** getting option

Options-set

Set value of a device option.

```
usage: tiflash [Session Arguments] options-set <optionID> <optionVal> [optionals]
```

Positional Arguments

optionID Option ID to set value of.

optionVal Value to set option to.

Named Arguments

-op, --operation **Specify an operation to perform after** setting option

Options-list

List device options.

```
usage: tiflash [Session Arguments] options-list [optionID]
```

Positional Arguments

optionID Option ID to get info on.

options-get

get an option on a device

options-set

set an option on a device

options-list

list options for a device

Attach

Open up CCS session & attach to device

```
usage: tiflash [Session Arguments] attach
```

List

List device/environment information.

```
usage: tiflash [Session Arguments] list [optionals]
```

Named Arguments

--devicetypes	Prints list of installed devicetypes
	Default: False
--connections	Prints list of installed connections
	Default: False
--cpus	Prints list of installed cpus
	Default: False
--options	Prints list of target options
	Default: False
-f, --filter	String to filter results by

Info

Prints out information of tiflash environment

```
usage: tiflash [Session Arguments] info
```

Below is a list of commands you can call with TIFlash.

Note: You can only provide one command at a time. All session arguments should be specified before specifying the command to use.

Reset

board reset a device

Flash

flash image(s) on to a device

Erase

erase a device's flash

Verify

verify an image in a device's flash

Memory

read/write to memory location in device's flash

Register

read/write to register of device

Evaluate

evaluate a C/GEL expression on a device

XDS110

run an xds110 command

Detect

run an detect command

Options

get/set/list device specific options

Attach

attach a CCS session to device

List

list environment/device information

Info

list tiflash environment information

2.4 Device Examples

Each device type will have specific options you can specify when performing an action in tiflash. These options are the same options available through the CCS GUI.

This page attempts to cover some of the more common options used for a few of these device types.

Note: Note that tiflash will work with any device that can be used in CCS! The devices listed below are just a few examples.

You can view the available options for a particular device by running:

```
tiflash -d DEVICETYPE options-list
```

2.4.1 MSP432

Below are common example commands specific for Texas Instruments' MSP432 devices

Reset After Flash

Reset the device after flashing

Python

```
>>> opts = {"ResetOnRestart" : True}
>>> tiflash.flash("/path/to/image.hex", options=opts, serno="M4321005")

True
```

CLI

```
$ tiflash -s M4321005 flash "/path/to/image.hex" -o "ResetOnRestart" "True"

True
```

Flash Reset Type - Hard Reset

Set Flash Reset Type to Hard Reset when flashing device (default option)

Python

```
>>> opts = {"FlashResetType" : "Hard reset"}
>>> tiflash.flash("/path/to/image.hex", options=opts, serno="M4321005")

True
```

CLI

```
$ tiflash -s M4321005 flash "/path/to/image.hex" -o "FlashResetType" "Hard reset"

True
```

Flash Reset Type - Soft Reset

Set Flash Reset Type to Soft Reset when flashing device

Python

```
>>> opts = {"FlashResetType" : "Soft reset"}
>>> tiflash.flash("/path/to/image.hex", options=opts, serno="M4321005")

True
```

CLI

```
$ tiflash -s M4321005 flash "/path/to/image.hex" -o "FlashResetType" "Soft reset"

True
```

Flash Erase - Main Memory Only

Erase main memory only (default option)

Python

```
>>> opts = {"FlashEraseSelection" : "Erase main memory only"}
>>> tiflash.flash("/path/to/image.hex", options=opts, serno="M4321005")

True
```

CLI

```
$ tiflash -s M4321005 flash "/path/to/image.hex" -o "FlashEraseSelection" "Erase main␣
↪memory only"

True
```

Flash Erase - Main and Information Memory

Erase main and information memory

Python

```
>>> opts = {"FlashEraseSelection" : "Erase main and information memory"}
>>> tiflash.flash("/path/to/image.hex", options=opts, serno="M4321005")

True
```

CLI

```
$ tiflash -s M4321005 flash "/path/to/image.hex" -o "FlashEraseSelection" "Erase main␣
↪and information memory"

True
```

Flash Erase - Factory Reset

Reset device to Factory default

Python

```
>>> opts = {"FlashEraseSelection" : "Factory Reset"}
>>> tiflash.flash("/path/to/image.hex", options=opts, serno="M4321005")

True
```

CLI

```
$ tiflash -s M4321005 flash "/path/to/image.hex" -o "FlashEraseSelection" "Factory␣
↪Reset"

True
```

Flash Erase - Necessary Segments Only

Erase and download necessary segments only

Python

```
>>> opts = {"FlashEraseSelection" : "Erase and download necessary segments only"}
>>> tiflash.flash("/path/to/image.hex", options=opts, serno="M4321005")

True
```

CLI

```
$ tiflash -s M4321005 flash "/path/to/image.hex" -o "FlashEraseSelection" "Erase and␣
↪download necessary segments only"

True
```

Flash Erase - Do Not Erase Flash Memory

Do not erase Flash memory

Python

```
>>> opts = {"FlashEraseSelection" : "Do not erase Flash memory"}
>>> tiflash.flash("/path/to/image.hex", options=opts, serno="M4321005")

True
```

CLI

```
$ tiflash -s M4321005 flash "/path/to/image.hex" -o "FlashEraseSelection" "Do not␣
↪erase Flash memory"

True
```

2.4.2 CC3220/S/SF

Below are common example commands specific for Texas Instruments' CC3220/S/SF devices

Reset After Flash

Reset the device after flashing

Python

```
>>> opts = {"ResetOnRestart" : True}
>>> tiflash.flash("/path/to/image.hex", options=opts, serno="E0071009")

True
```

CLI

```
$ tiflash -s E0071009 flash "/path/to/image.hex" -o "ResetOnRestart" "True"

True
```

Flash Program Option - Necessary Pages Only

Erase necessary pages only (default option)

Python

```
>>> opts = {"FlashProgramOption" : "Necessary Pages Only"}
>>> tiflash.flash("/path/to/image.hex", options=opts, serno="E0071009")

True
```

CLI

```
$ tiflash -s E0071009 flash "/path/to/image.hex" -o "FlashProgramOption" "Necessary␣
↪Pages Only"

True
```

Flash Program Option - Erase Options Specified in FlashEraseType

Use Erase Options specified in FlashEraseType

Python

```
>>> opts = {"FlashProgramOption" : "Use the Erase Options Specified Below"}
>>> tiflash.flash("/path/to/image.hex", options=opts, serno="E0071009")

True
```

CLI

```
$ tiflash -s E0071009 flash "/path/to/image.hex" -o "FlashProgramOption" "Use the␣
↪Erase Options Specified Below"

True
```

Flash Program Option - Do Not Erase Flash Memory

Do not erase Flash Memory

Python

```
>>> opts = {"FlashProgramOption" : "Do Not Erase Flash Memory"}
>>> tiflash.flash("/path/to/image.hex", options=opts, serno="E0071009")

True
```

CLI

```
$ tiflash -s E0071009 flash "/path/to/image.hex" -o "FlashProgramOption" "Do Not␣
↪Erase Flash Memory"

True
```

Flash Erase Type - Entire Flash

Erase the Entire Flash (default option)

> **Warning:** *FlashProgramOption* must be set to *"Use the Erase Options Specified Below"* in order for this setting to be used

Python

```
>>> opts = {"FlashEraseType" : "Entire Flash"}
>>> opts["FlashProgramOption"] = "Use the Erase Options Specified Below"
>>> tiflash.flash("/path/to/image.hex", options=opts, serno="E0071009")

True
```

CLI

```
$ tiflash -s E0071009 flash "/path/to/image.hex" -o "FlashEraseType" "Entire Flash" -
↪o "FlashProgramOption" "Use the Erase Options Specified Below"

True
```

Flash Erase Type - By Address Range

Erase Flash by specified Address Range

> **Warning:** *FlashProgramOption* must be set to *"Use the Erase Options Specified Below"* in order for this setting to be used

---

Note: Address Range is set by the *FlashEraseEndAddr* and *FlashEraseStartAddr* options

---

Python

```
>>> opts = {"FlashEraseType" : "By Address Range"} : }
>>> opts["FlashProgramOption"] = "Use the Erase Options Specified Below"
>>> opts["FlashEraseStartAddr"] = 0
>>> opts["FlashEraseEndAddr"] = 0xFFFF
```

(continues on next page)

(continued from previous page)

```
>>> tiflash.flash("/path/to/image.hex", options=opts, serno="E0071009")

True
```

CLI

```
$ tiflash -s E0071009 flash "/path/to/image.hex" -o "FlashEraseType" "By Address Range
↪" -o "FlashProgramOption" "Use the Erase Options Specified Below" -o
↪"FlashEraseStartAddr" 0 -o "FlashEraseEndAddr" 0xFFFF

True
```

Flash Crystal Frequency

Set the Flash Crystal Frequency

Python

```
>>> opts = {"FlashCrystalFreq" : "8"}
>>> tiflash.flash("/path/to/image.hex", options=opts, serno="E0071009")

True
```

CLI

```
$ tiflash -s E0071009 flash "/path/to/image.hex" -o "FlashCrystalFreq" "8"

True
```

2.4.3 CC13XX + CC26XX

Below are common example commands specific for Texas Instruments' CC13XX and CC26XX devices.

- *IEEE Address (Read)*
- *IEEE Address (Write)*
- *BLE Address (Read)*
- *BLE Address (Write)*
- *Flash Erase - All Unprotected Sectors*
- *Flash Erase - Necessary Sectors Only*
- *Flash Erase - Program Load Only*
- *Reset After Flash*
- *Device Revision*
- *Device RAM Size*
- *Device Flash Size*

IEEE Address (Read)

Obtaining a device's IEEE Address

Python

```
>>> tiflash.get_option("DeviceIeeePrimary", pre_operation="ReadPriIeee", serno=
↪"L4000CE")

00:12:4B:00:11:22:33:44
```

CLI

```
$ tiflash -s L4000CE options-get DeviceIeeePrimary --operation "ReadPriIeee"

00:12:4B:00:11:22:33:44
```

IEEE Address (Write)

Writing a device's Secondary IEEE Address

Python

```
>>> tiflash.set_option("DeviceIeeeSecondary", "12:34:56:78:9A:BC:DE:F0", serno=
↪"L4000CE", post_operation="WriteSecIeee")
```

CLI

```
$ tiflash -s L4000CE options-set "DeviceIeeeSecondary" "12:34:56:78:9A:BC:DE:F0" -op
↪"WriteSecIeee"
```

Writing a device's Secondary IEEE Address (when flashing)

Python

```
>>> opts = {"DeviceIeeeSecondary": "12:34:56:78:9A:BC:DE:F0",
↪"OverrideIeeeSecondaryAddr": True}
>>> tiflash.flash("/path/to/image.hex", serno="L4000CE", options=opts)

True
```

CLI

```
$ tiflash -s L4000CE flash "/path/to/image.hex" -o DeviceIeeeSecondary
↪"12:34:56:78:9A:BC:DE:F0" -o OverrideIeeeSecondaryAddr True

True
```

BLE Address (Read)

Obtaining a device's BLE Address

Python

```
>>> tiflash.get_option("DeviceBlePrimary", pre_operation="ReadPriBle", serno="L4000CE
↪")

00:81:F9:11:22:33
```

CLI

```
$ tiflash -s L4000CE options-get DeviceBlePrimary --operation "ReadPriBle"

00:81:F9:11:22:33
```

BLE Address (Write)

Writing a device's Secondary BLE Address

Python

```
>>> tiflash.set_option("DeviceBleSecondary", "12:34:56:78:9A:BC:DE:F0", serno="L4000CE
↪", post_operation="WriteSecBle")
```

CLI

```
$ tiflash -s L4000CE options-set "DeviceBleSecondary" "12:34:56:78:9A:BC:DE:F0" -op
↪"WriteSecBle"
```

Writing a device's Secondary BLE Address (when flashing)

Python

```
>>> opts = {"DeviceBleSecondary": "12:34:56:78:9A:BC", "OverrideBleSecondaryAddr":␣
↪True}
>>> tiflash.flash("/path/to/image.hex", serno="L4000CE", options=opts)

True
```

CLI

```
$ tiflash -s L4000CE flash "/path/to/image.hex" -o DeviceBleSecondary
↪"12:34:56:78:9A:BC" -o OverrideBleSecondaryAddr True

True
```

Flash Erase - All Unprotected Sectors

Erase entire Flash on device before flashing image

Python

```
>>> opts = {"FlashEraseSetting" : "All Unprotected Sectors"}
>>> tiflash.flash("/path/to/image.hex", options=opts, serno="L4000CE")

True
```

CLI

```
$ tiflash -s L4000CE flash "/path/to/image.hex" -o "FlashEraseSetting" "All␣
↪Unprotected Sectors"

True
```

Flash Erase - Necessary Sectors Only

Erase Necessary Sectors Only of Flash on device before flashing image (default option)

Python

```
>>> opts = {"FlashEraseSetting" : "Necessary Sectors Only"}
>>> tiflash.flash("/path/to/image.hex", options=opts, serno="L4000CE")

True
```

CLI

```
$ tiflash -s L4000CE flash "/path/to/image.hex" -o "FlashEraseSetting" "Necessary␣
↪Sectors Only"

True
```

Flash Erase - Program Load Only

Program Load Only (do not erase any sectors of flash) when flashing image on to device

Python

```
>>> opts = {"FlashEraseSetting" : "Program Load Only (do not erase sectors)"}
>>> tiflash.flash("/path/to/image.hex", options=opts, serno="L4000CE")

True
```

CLI

```
$ tiflash -s L4000CE flash "/path/to/image.hex" -o "FlashEraseSetting" "Program Load␣
↪Only (do not erase sectors)"

True
```

Reset After Flash

Reset the device after flashing

Python

```
>>> opts = {"ResetOnRestart" : True}
>>> tiflash.flash("/path/to/image.hex", options=opts, serno="L4000CE")

True
```

CLI

```
$ tiflash -s L4000CE flash "/path/to/image.hex" -o "ResetOnRestart" "True"

True
```

Device Revision

Get device's Revision Number

Python

```
>>> tiflash.get_option("DeviceInfoRevision", serno="L4000CE")

"2.1"
```

CLI

```
$ tiflash -s L4000CE options-get DeviceInfoRevision

2.1
```

Device RAM Size

Get RAM size on device

Python

```
>>> tiflash.get_option("DeviceInfoRAMSize", serno="L4000CE")

"80 KB"
```

CLI

```
$ tiflash -s L4000CE options-get DeviceInfoRAMSize

80 KB
```

Device Flash Size

Get Flash size on device

Python

```
>>> tiflash.get_option("DeviceInfoFlashSize", serno="L4000CE")

"352 KB"
```

CLI

```
$ tiflash -s L4000CE options-get DeviceInfoFlashSize

352 KB
```

2.5 Contributing Guidelines

Contributions are very welcome! This includes not only code, but bug reports and documentation. Please follow the guidelines laid out below.

When contributing to this repository, please first discuss the change you wish to make via an "issue". The issue will be used as a forum of discussion for the bug, feature or update before merging the changes.

This repo follows the Git Feature Branch Workflow for any new features/bugs/updates.

2.5.1 Table of Contents

2.5.2 Setting Up Development Environment

Install via git repo

```
# clone repo
git install https://github.com/webbcam/tiflash.git

# install required 3rd party modules
cd tiflash
pip install -r requirements.txt

# install tiflash via pip in develop mode
pip install -e .

# setup pre-commit hooks (style guide testing)
cd hooks
chmod +x install-hooks.sh pre-commit.sh
./install-hooks.sh
```

2.5.3 Running Tests

Before creating any pull requests you should be sure to run the tests (located in tests directory) locally.

Prerequisites

You'll need the pytest module to run the tests.

```
# install pytest
pip install -U pytest
```

You'll also need at least one device connected to your PC to run the tests on. You'll need to add the device(s) information to the tests/devices.cfg file. (Also include a valid image in this configuration file for flash tests).

Running the tests

All tests are located in the tests directory. Please see the README.md in the tests directory for detailed information on running the tests.

You are able to run any tests ranging from an entire test suite to sub test suites to just a particular test. *(Note: upon submitting a pull request, we will run the entire test suite before merging).*

```
cd tiflash/tests

# Entire Test Suite
py.test ./

# Sub Test Suite (i.e. core)
py.test core/

# Particular Test
py.test utils/ccsfinder.py
```

**Please see the README.md in the tests directory for more information*

2.5.4 Raising an Issue

Please raise an issue for any bug, new feature or updates. You should use the ISSUE_TEMPLATE.md as a starting place for your issue.

2.5.5 Pull Requests

When preparing a pull request you should run through the following steps:

1. Run entire test suite on your local PC with the new changes and include report.html (generated with pytest-html plugin) with pull request.
2. Update the appropriate README.md with details of changes to the project. This includes any new feature added, any new tests added (and expected result) for fixed bug, etc.
3. Increase the version numbers in any examples files and the README.md to the new version that this Pull Request would represent. The versioning scheme we use is SemVer.
4. You may merge the Pull Request in once you have the sign-off of two other developers, or if you do not have permission to do that, you may request the second reviewer to merge it for you. Note all tests must pass on our test setup before Pull Request can be merged.

2.6 License

TIFlash is released under the *MIT license*:

```
Copyright (c) 2018 Cameron Webb

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is␣
↪furnished
```

(continues on next page)

(continued from previous page)

```
to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in␣
↪all
copies or substantial portions of the Software.

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

2.7 Disclaimer

```
This project is NOT supported by (nor affiliated with) Texas Instruments Inc.

Code Composer Studio is a trademark of Texas Instruments Inc.
Any and all other trademarks are the property of their respective owners.
```

2.8 Release Change Log

2.8.1 1.2.9

Date: 09.17.2019

- fix default board_ids.json issue (#86)

2.8.2 1.2.8

Date: 04.11.2019

- fix list_options() issue with CCS9 (#77)

2.8.3 1.2.7

Date: 03.25.2019

- fix trailing path separator issue (#72)

2.8.4 1.2.6

Date: 02.17.2019

- fix xds110-list issue (#69)
- removed module ascii art

2.8.5 1.2.5

Date: 02.09.2019

- fixed bug dealing with Python installation paths having a space (#65)
- added .tiflash folder to be default workspace (#66)

2.8.6 1.2.4

Date: 02.08.2019

- added 'info' command (#63)
- improved CCS version choice (#60)
- improved CCS installation path discovery (#61)

Python Module Index

t

Index