

HOW-TO Guide for Beluga

Installing the Toolchain.....	2
Tools to Install.....	2
Installing nRF Command Line Tools and nRF Connect.....	2
Additional Steps for nRF Connect (Linux).....	4
Installing the NCS toolchain.....	4
Cannot find v2.9.0? No Problem!.....	6
(Optional) Generate Environment Script.....	7
Installing VS code extension.....	7
Configuring Builds.....	9
Adding a build configuration.....	9
Decawave DWM1001 Dev Kit.....	9
Custom Beluga Hardware.....	10
Building for Custom Beluga Hardware with External Flash.....	11
Signing the image with a custom key.....	12
Setting up generation environment.....	12
Generating a key.....	13
Incorporating the key into firmware.....	13
Flashing the firmware.....	14
Flashing onto a Decawave DWM1001 Dev.....	14
Flashing onto the Custom Beluga Hardware.....	14
Using JTAG.....	14
Using SEGGER.....	14
Using nRF52 devkit.....	15
Using DFU.....	16
DFU Gotcha.....	22
imgflash.....	23
Installing.....	23
Commands.....	23
Ports.....	24
Options.....	24
Example output.....	24
Images.....	24
Options.....	25
Example output.....	25
Flash.....	26
Arguments.....	26
Options.....	27
Example output.....	27
Reset.....	29

Arguments.....	29
Options.....	29
Example output.....	29
Notes.....	30
Confirm.....	30
Arguments.....	30
Options.....	30
Example output.....	31
Swap.....	31
Arguments.....	32
Options.....	32
Example Output.....	32

Installing the Toolchain

This section will detail the steps for installing the toolchain and setting up VS code to build and flash the Beluga firmware to hardware.

Tools to Install

- [nRF Command Line Tools](#)
- [nRF Connect](#)
- [SEGGER J-Link](#)
- [VS code](#)

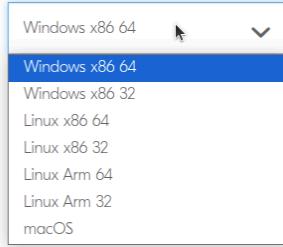
Installing nRF Command Line Tools and nRF Connect

This section details how to install the nRF tools since their website is non-intuitive.

1. Navigate to the link specified in [Tools to Install](#)
2. Select the platform you are running on

Choose platform and version

Choose your Desktop platform and select version (latest released version recommended)



3. Select the version you want (latest version should be fine)

Changelog:

10.24.2 Linux x86 64

HIGHLIGHTS:

- (nrfjprogexe) Calling '--program --recover' on a protected device would display errors stating it is protected. These errors are no longer displayed.

10.24.1 Linux x86 64
10.24.0 Linux x86 64
10.23.5 Linux x86 64
10.23.4 Linux x86 64
10.23.2 Linux x86 64
10.23.1 Linux x86 64
10.23.0 Linux x86 64
10.22.1 Linux x86 64
10.22.0 Linux x86 64
10.21.0 Linux x86 64

4. Click on the download link

Selected version

10.24.2 Linux x86 64

- [nrf-command-line-tools_10.24.2_amd64.deb](#)
- [nrf-command-line-tools-10.24.2-1.x86_64.rpm](#)
- [nrf-command-line-tools-10.24.2_Linux-amd64.tar.gz](#)

Additional Steps for nRF Connect (Linux)

When you download nRF Connect for linux, it downloads as an app image. Before running it, you need to install libfuse2 and mark the file as executable.

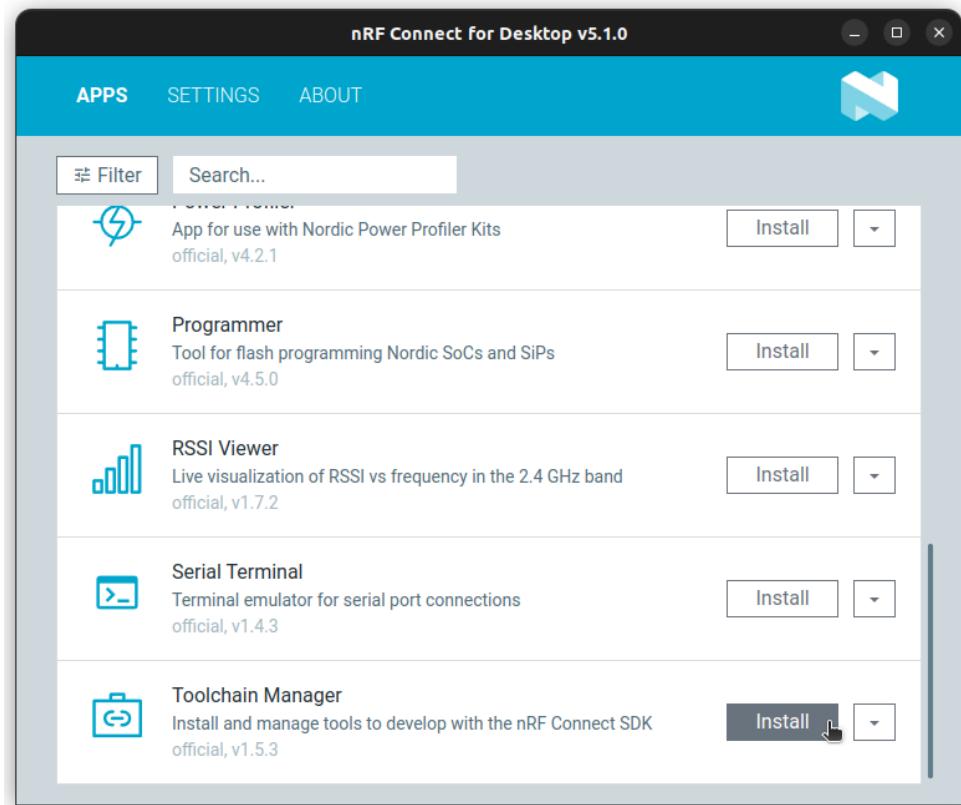
Unset

```
sudo apt install libfuse2
sudo chmod +x nrfconnect-5.1.0-x86_64.appimage # Make sure to change image name
```

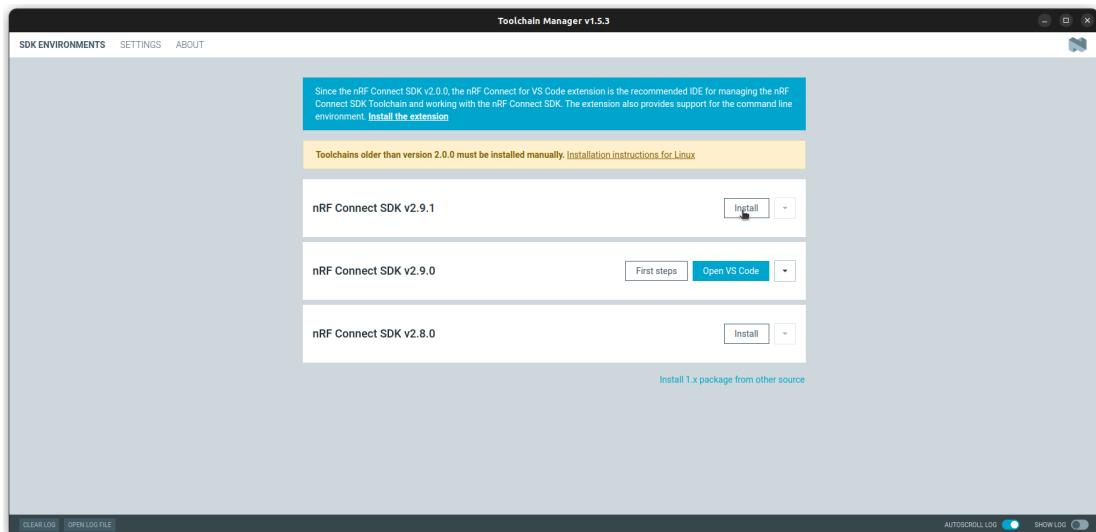
Installing the NCS toolchain

Even though the Nordic extension in VS code has the option to install the toolchain from there, it does not work. This method is proven to work.

1. Open nRF Connect
2. Install the “Toolchain Manager”



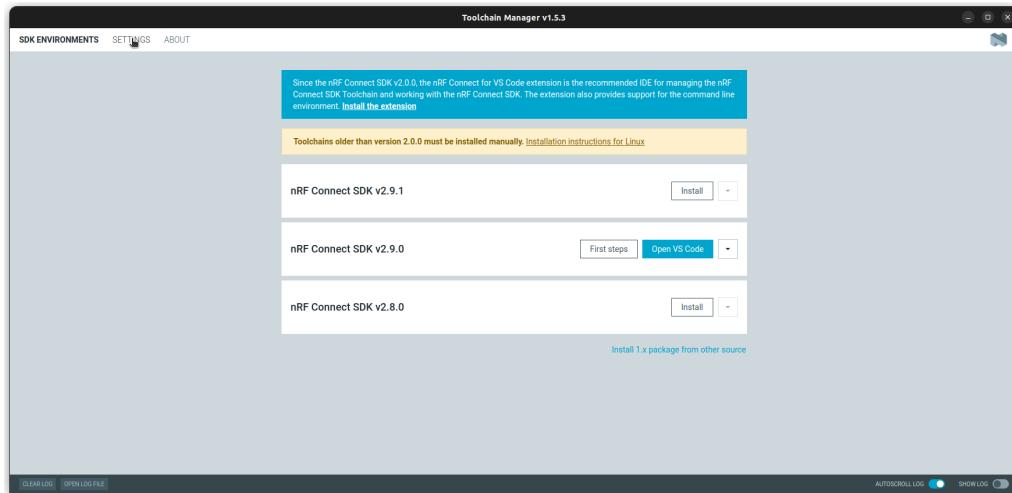
3. Open the Toolchain Manager and install nRF Connect SDK v2.9.0



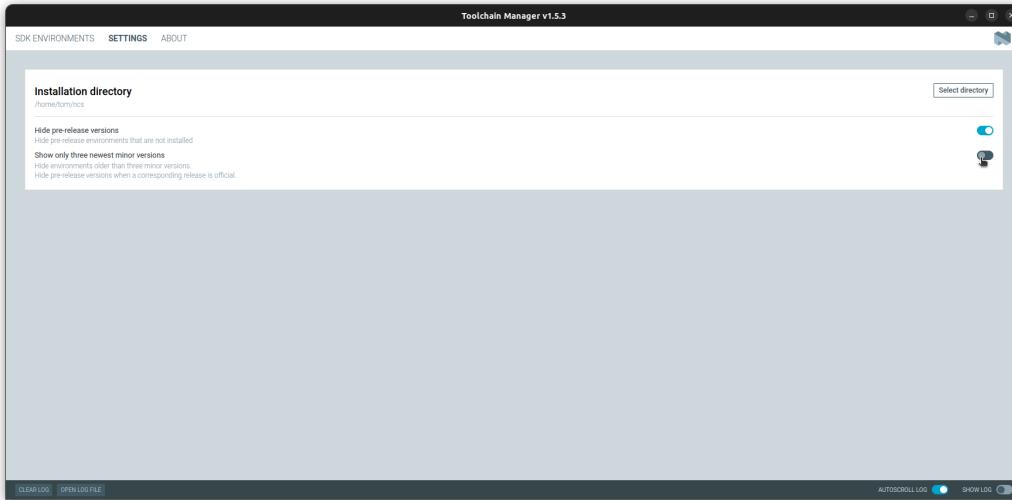
Cannot find v2.9.0? No Problem!

The nRF Connect Toolchain Manager shows the latest 3 versions of the SDK by default. To get around this, follow the steps below:

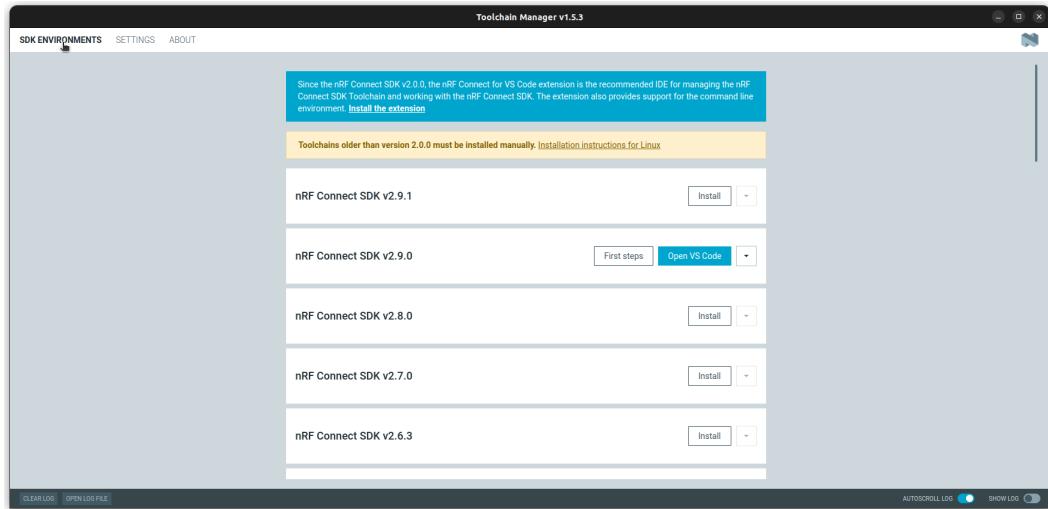
1. Click on the settings tab



2. Toggle the “Show only three newest minor versions” option



3. Navigate back to the “SDK Environments” tab

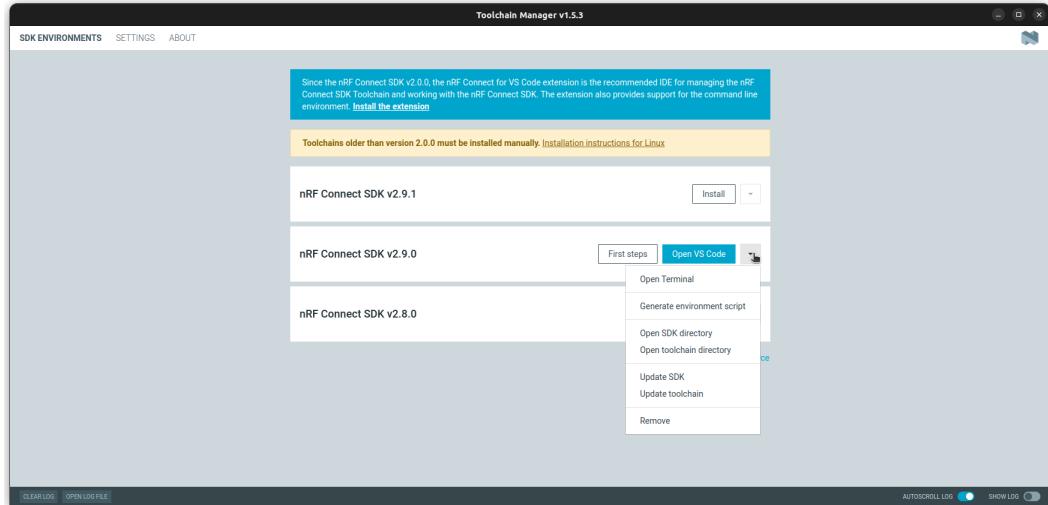


Now you can access versions no older than v2.0.0.

(Optional) Generate Environment Script

Sometimes, it is necessary to generate the environment script. For instance, if you want to use a Jetbrains IDE, this is a necessary step. Another use case for the environment script is using the NCS environment in a terminal.

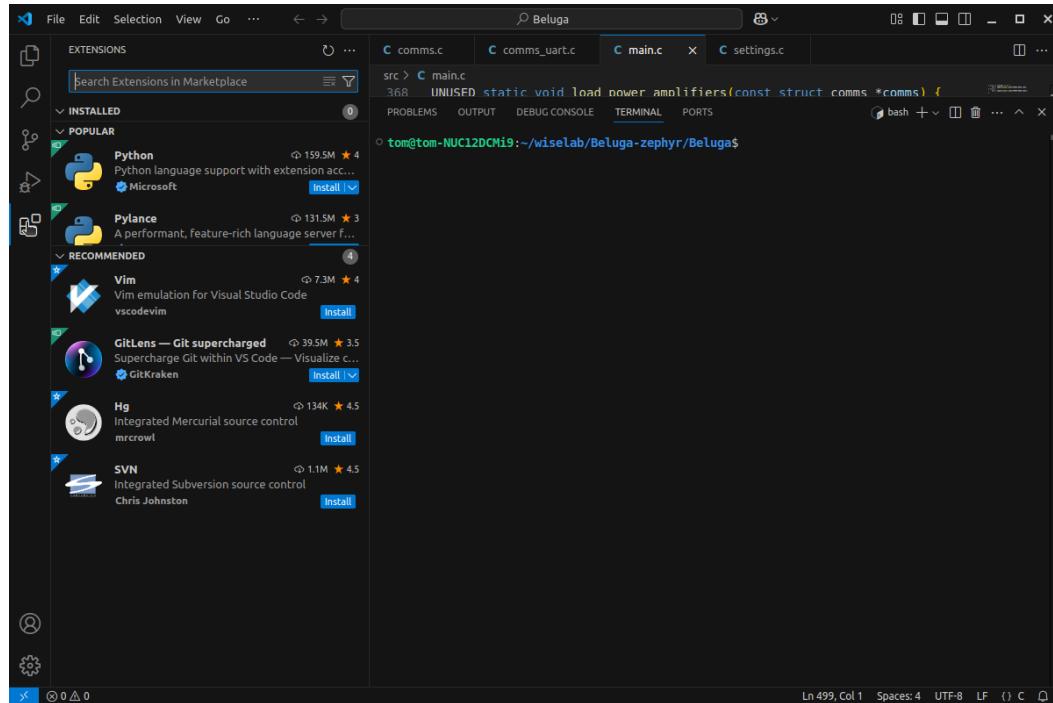
1. Click on the Down Arrow



2. Click “Generate environment script”
3. Select the save location for the environment script

Installing VS code extension

1. Open VS code
2. Open “Extensions”



3. Search ‘nRF Connect’ in the marketplace
4. Install the “nRF Connect for VS Code Extension”

nRF Connect for VS Code Extension

Nordic Semiconductor | nordicsemi.com | 97,758 | ★★★★☆

Recommended extensions for development with the nRF Connect S...

Extension Pack (7)

- C/C++** C/C++ IntelliSense, debugging, and code br... Microsoft [Install](#)
- CMake** CMake language support for Visual Studio C... twxs [Install](#)
- GNU Linker Map files** Syntax highlighting and symbol listing for ... Trond Snekvik [Install](#)

nRF Connect Visual Studio Code Extension Pack

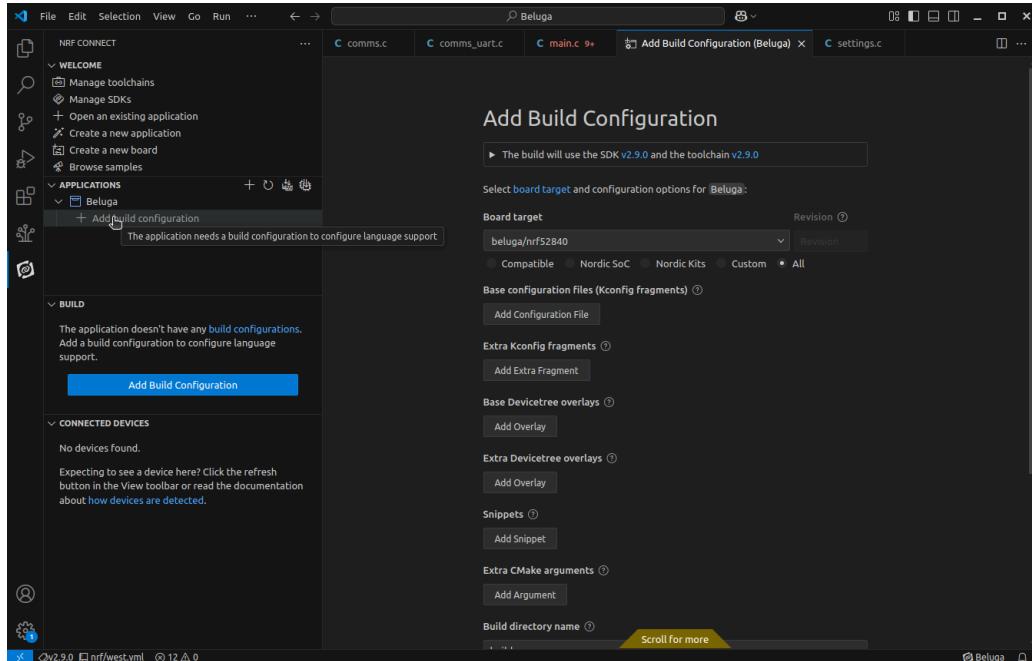
Nordic Semiconductor's nRF Connect Visual Studio Code Extension Pack provides an easier, user-friendly GUI to build and configure your embedded systems projects based on the nRF Connect SD...

Configuring Builds

This section will detail the steps for creating a board configuration in VS code. It will also detail the requirements for both the decawave dev kit as well as the custom Beluga board.

Adding a build configuration

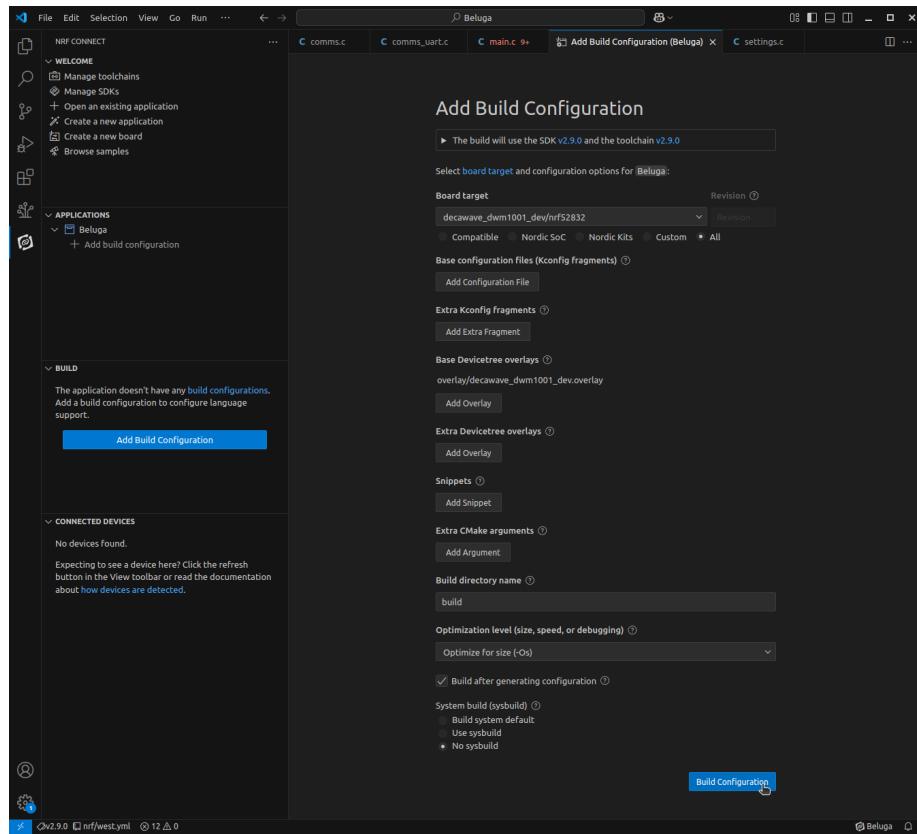
1. Open VS code
2. Switch to the nRF Connect tab (CTRL+ALT+N)



3. Click “Add Build Configuration”
4. Follow steps for the specific board you are configuring for

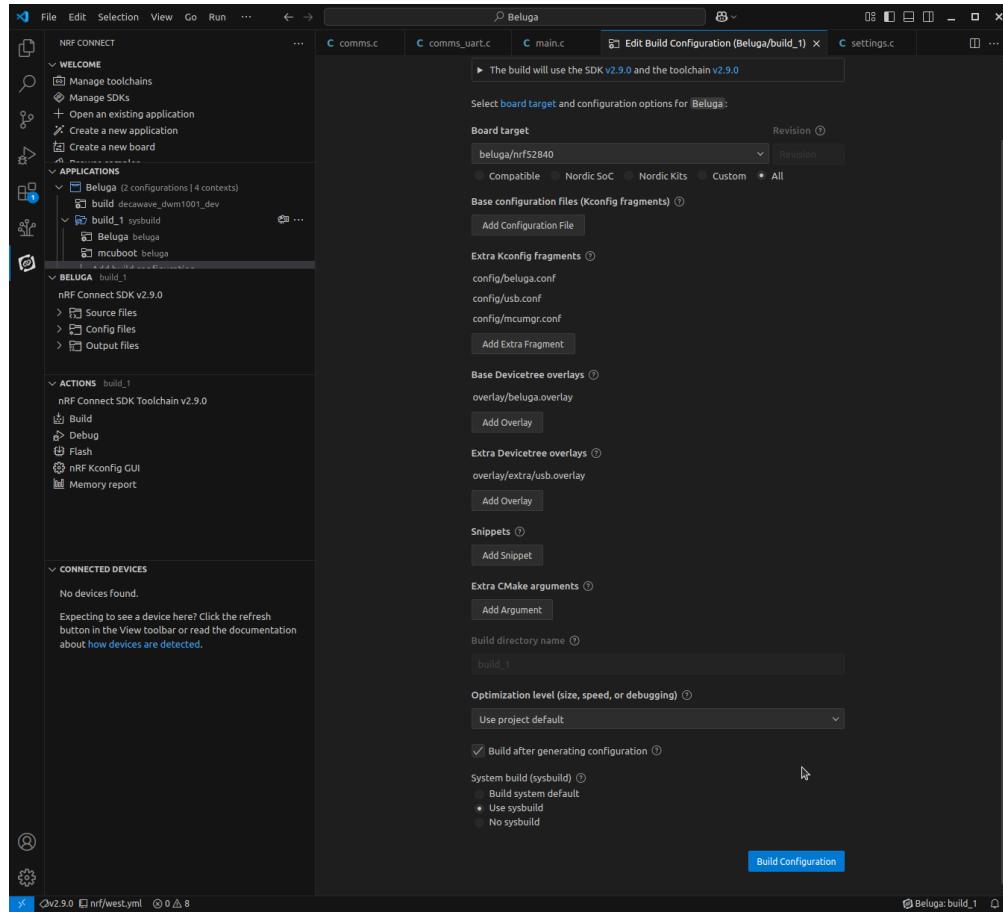
Decawave DWM1001 Dev Kit

1. Select **Board target** to be “decawave_dwm1001_dev/nrf52832” (You may have to select “All” under board target)
2. (Optional) Add “prj.conf” as the **Base configuration files (Kconfig fragments)**
 - a. This is not needed, but if there is a weird issue with the build, then it is worth a try
3. Add “overlay/decawave_dwm1001_dev.overlay” to the **Base Devicetree overlays**
4. Select “Optimize for size (-Os) for **Optimization level (size, speed, or debugging)**
5. Select “No sysbuild” for **System build (sysbuild)**
6. Confirm your settings match the ones in the image



Custom Beluga Hardware

1. Select **Board target** to be “beluga/nrf52840” (You may have to select “All” under board target)
2. (Optional) Add “prj.conf” as the **Base configuration files (Kconfig fragments)**
 - a. This is not needed, but if there is a weird issue with the build, then it is worth a try
3. Add the following files to **Extra Kconfig fragments**
 - a. conf/beluga.conf
 - b. conf/mcumgr.conf
 - c. conf/usb.conf
4. Add “overlay/beluga.overlay” to the **Base Devicetree overlays**
5. Add “overlay/extra/usb.overlay” to the **Extra Devicetree overlays**
6. Select whatever **Optimization level** you want
7. Select “Use sysbuild” under **System build (sysbuild)**
8. Confirm that you have the correct settings



Building for Custom Beluga Hardware with External Flash

1. Add the following configurations to **sysbuild/mcuboot.conf**

```
Unset
CONFIG_NORDIC_QSPI_NOR=y
CONFIG_BOOT_MAX_IMG_SECTORS=256
```

2. Add the following to **sysbuild/mcuboot.overlay**

```
Unset
&mx25r64 {
    status = "okay";
};
```

```
/ {  
    chosen {  
        nordic_pm-ext-flash = &mx25r64;  
    };  
};
```

3. Add the following to **sysbuild.conf**

```
Unset  
SB_CONFIG_PM_EXTERNAL_FLASH_MCUBOOT_SECONDARY=y
```

4. Follow the steps in [Custom Beluga Hardware](#), but also add the following files to the specified sections below.

- Add “config/flash.conf” to **Extra Kconfig fragments**
- Add “overlay/extra/flash.overlay” to **Extra Devicetree overlays**

Signing the image with a custom key

When deploying in the world, it is highly recommended to sign the image with a custom key. This key can easily be generated and prevents anyone from uploading their own firmware.

Setting up generation environment

Before generating a key, a python environment needs to be set up. Run the following commands to get a Python environment set up with *imgtool* installed.

```
Unset  
mkdir -p keys && cd keys  
python3 -m venv .venv  
source .venv/bin/activate  
pip install imgtool
```

Note that the commands may be a little different for Windows (Why would you even run Windows?).

Before proceeding, ensure that the tool got installed correctly by running `imgtool --help`. If it shows usage information, then it installed correctly. If it shows a `Module not found error` then you need to run the following command:

```
Unset  
pip install cryptography intelhex click cbor2 pyyaml
```

Generating a key

Once the environment is set up, a new key can be generated by running one of the commands below:

```
Unset  
imgtool keygen -t ecdsa-p256 -k private_key.pem  
imgtool keygen -t rsa-2048 -k private_key.pem  
imgtool keygen -t rsa-3072 -k private_key.pem  
imgtool keygen -t ed25519 -k private_key.pem
```

Remember which algorithm you used to generate the key as it will be important for the firmware. Additionally, backup the key somewhere safe. It is not uncommon to lose the key and thus be unable to ever do DFU on the device again (until the device is flashed again over JTAG).

Incorporating the key into firmware

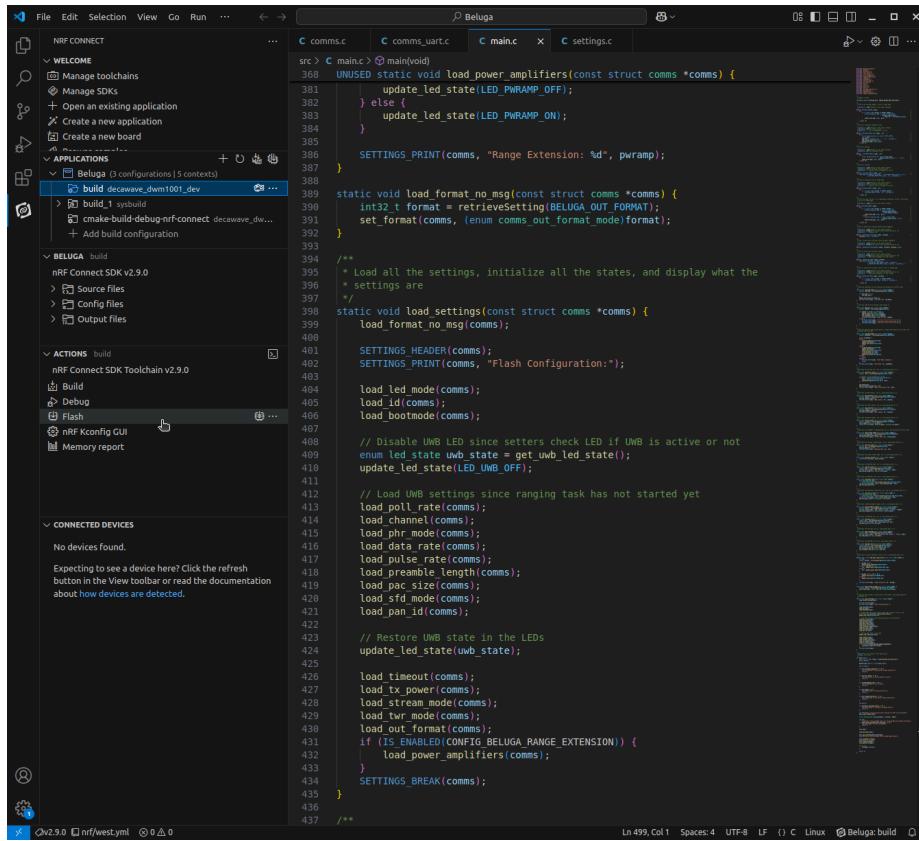
Once the key is generated, it needs to be incorporated into firmware. This is relatively easy as it requires you to update `sysbuild.conf`. For example, if an `ecdsa-p256` key was generated in `Beluga/keys`, the the following lines would have to be added to `sysbuild.conf`:

```
Unset  
SB_CONFIG_BOOT_SIGNATURE_KEY_FILE="\${APP_DIR}/keys/private_key.pem"  
SB_CONFIG_BOOT_SIGNATURE_TYPE_ECDSA_P256=y
```

Flashing the firmware

Flashing onto a Decawave DWM1001 Dev

1. Plug the decawave DWM1001 Dev into your computer
2. Select the DWM1001 build
3. Press flash



```
File Edit Selection View Go Run ... ← → Beluga
src > C main.c C comms_uart.c C main.c x C settings.c
368 UNUSED static void load_power_amplifiers(const struct comms *comms) {
381     update_led_state(LED_PWRAMP_OFF);
382 } else {
383     update_led_state(LED_PWRAMP_ON);
384 }
385
386 SETTINGS_PRINT(comms, "Range Extension: %d", pwramp);
387 }

388 static void load_format_no_msg(const struct comms *comms) {
389     int32_t format = retrieveSetting(BELUGA_OUT_FORMAT);
390     set_format(comms, (enum comms_out_format_mode)format);
391 }
392 }

393 /**
394 * Load all the settings, initialize all the states, and display what the
395 * settings are
396 */
397 static void load_settings(const struct comms *comms) {
398     load_format_no_msg(comms);

399     SETTINGS_HEADER(comms);
400     SETTINGS_PRINT(comms, "Flash Configuration:");

401     load_led_mode(comms);
402     load_id(comms);
403     load_bootmode(comms);

404     // Disable UWB LED since setters check LED if UWB is active or not
405     enum led_state uwb_state = get_uwb_led_state();
406     update_led_state(LED_UWB_OFF);

407     // Load UWB settings since ranging task has not started yet
408     load_poll_rate(comms);
409     load_channel(comms);
410     load_preamble(comms);
411     load_tx_power(comms);
412     load_rx_rate(comms);
413     load_pulse_rate(comms);
414     load_preamble_length(comms);
415     load_pac_size(comms);
416     load_sf0_mode(comms);
417     load_sf1_mode(comms);
418     load_sf2_mode(comms);
419     load_sf3_mode(comms);
420     load_sf4_mode(comms);
421     load_sf5_mode(comms);
422     load_sf6_mode(comms);
423     load_sf7_mode(comms);

424     // Restore UWB state in the LEDs
425     update_led_state(uwb_state);

426     load_timeout(comms);
427     load_tx_power(comms);
428     load_stream_mode(comms);
429     load_twr_mode(comms);
430     load_sf0_format(comms);
431     if (IS_ENABLED(CONFIG_BELUGA_RANGE_EXTENSION)) {
432         load_power_amplifiers(comms);
433     }
434 }
435 }

436 SETTINGS_BREAK(comms);

437 /**
Ln 499, Col 1 Spaces:4 UTF-8 LF () C Linux Beluga:build
```

Flashing onto the Custom Beluga Hardware

Flashing using JTAG is required for the first time. Any time after that can be done over DFU.

Using JTAG

Before flashing, make sure to select the Beluga image.

Using SEGGER

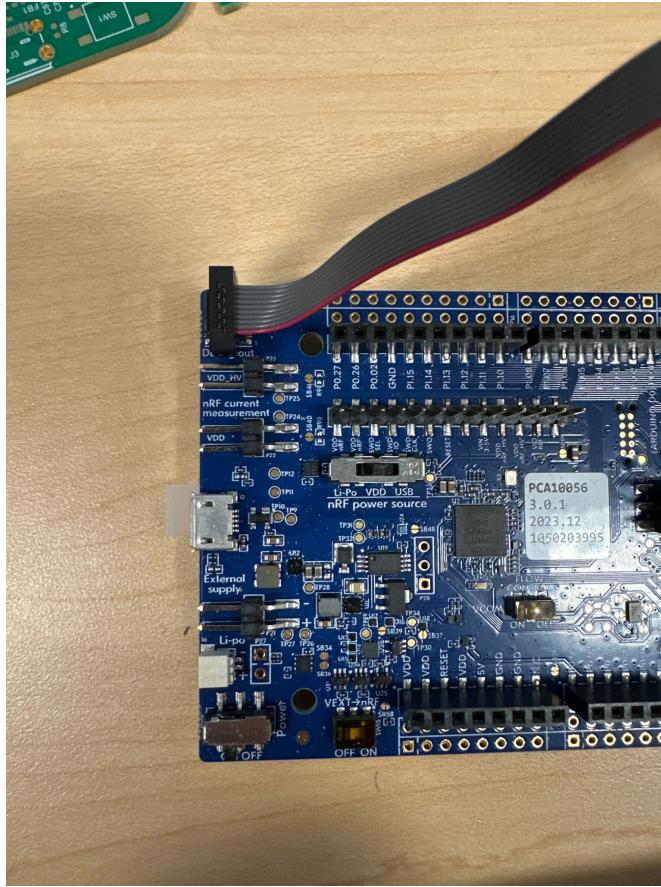
1. Plug the [SEGGER J-Link](#) into your computer
2. Connect the J-Link to the board using the [Tag-Connect TC2050-IDC](#)



3. Select Beluga build
4. Press flash

Using nRF52 devkit

1. Connect a ribbon cable to the Debug Out connector on an [nRF52 dev kit](#)



2. Connect the ribbon cable to an adaptor
3. Connect adaptor to the board using the [Tag-Connect TC2050-IDC](#)
4. Connect dev kit to computer
5. Select beluga build
6. Press flash

Using DFU

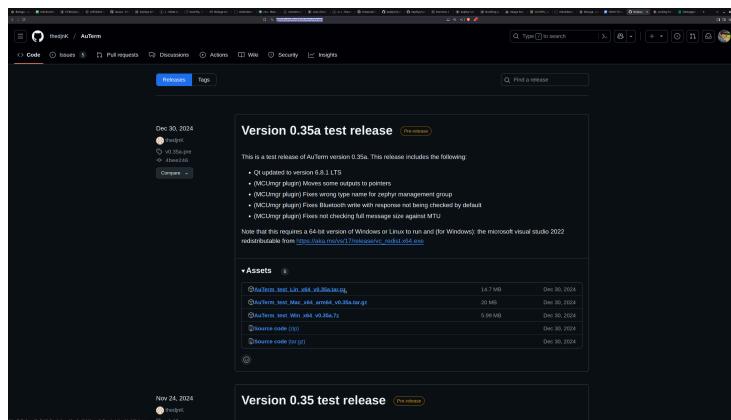
After flashing using JTAG, the custom Beluga hardware will be able to be flashed over USB. This will require an SMP client to do. There are a few tools and libraries that Zephyr recommends, all of which can be found [here](#). If the web page is no longer working, reference the table below:

Name	OS support			Type	Language	License
	Windows	Linux	Mac			
AuTerm	Yes	Yes	Yes	Application	C++	GPL-3.0
mcumgr-client	Yes	Yes	Yes	Application	Rust	Apache-2.0

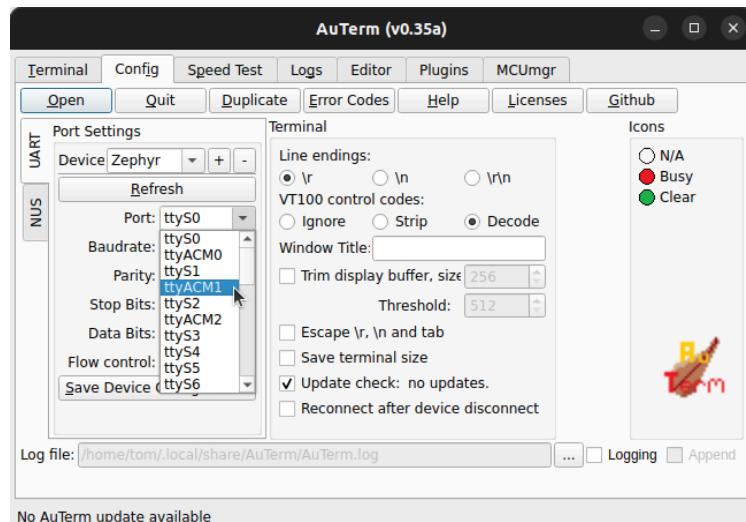
smclient	Yes	Yes	Yes	Library	Python	Apache-2.0
Zephyr MCUmgr client (in-tree)	No	Yes	No	Library	C	Apache-2.0

For this guide, AuTerm and imgflash will be used (because I find it incredibly entertaining to annoy the rust community). The usage for imgflash can be found later in this guide.

1. Download the latest release of AuTerm from the [releases page](#) (At the time of writing, there were no releases, only pre-releases).
 - a. Download the build for your operating system

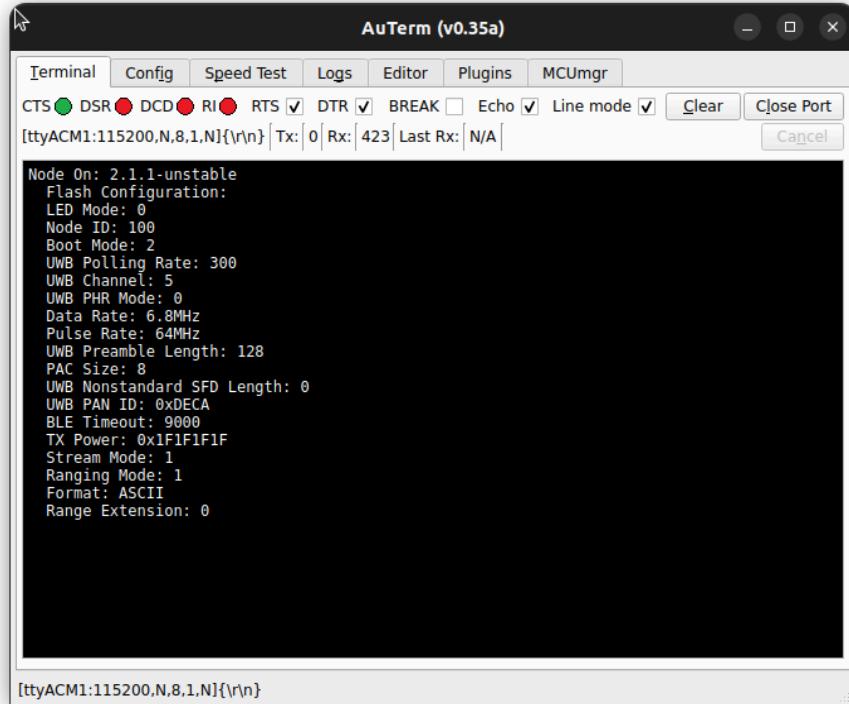


2. Extract the binary
3. Launch AuTerm
 - a. For Linux and maybe macOS, you need to mark the binary as executable
4. Press refresh and click on the dropdown list to find the serial port to connect to
 - a. On Linux, it should be one of the ttyACM ports
 - b. On MacOS, it should be one of the <insert port here> ports



5. Before opening, make sure you have no parity, 115200 baud, 1 stop bit, 8 data bits, and no flow control. Press open once you have those settings.

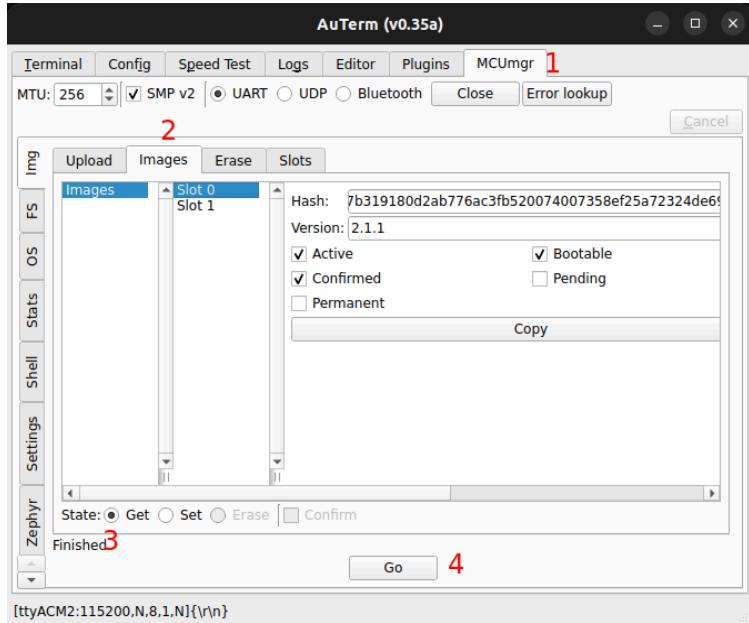
- a. If you see something like this in the terminal, it means you have the wrong port



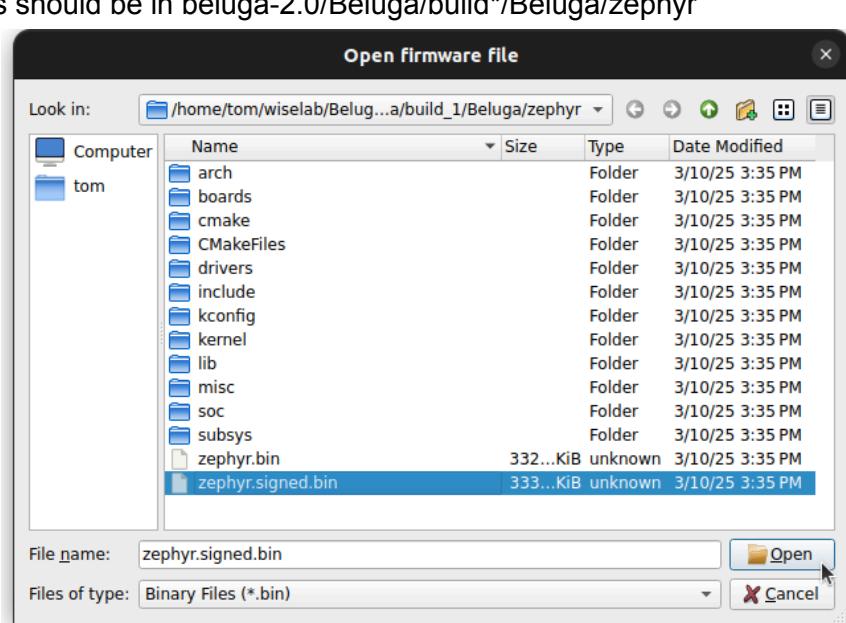
The screenshot shows the AuTerm (v0.35a) terminal window. The title bar says "AuTerm (v0.35a)". The menu bar includes Terminal, Config, Speed Test, Logs, Editor, Plugins, and MCUmgr. Below the menu is a toolbar with buttons for CTS (green), DSR (red), DCD (red), RI (red), RTS (checkmark), DTR (checkmark), BREAK (unchecked), Echo (checkmark), Line mode (checkmark), Clear (button), and Close Port (button). The status bar at the bottom shows the port as [ttyACM1:115200,N,8,1,N]{\r\n} and statistics Tx: 0 Rx: 423 Last Rx: N/A. The main terminal window displays the following configuration parameters:

```
Node On: 2.1.1-unstable
Flash Configuration:
LED Mode: 0
Node ID: 100
Boot Mode: 2
UWB Polling Rate: 300
UWB Channel: 5
UWB PHR Mode: 0
Data Rate: 6.8MHz
Pulse Rate: 64MHz
UWB Preamble Length: 128
PAC Size: 8
UWB Nonstandard SFD Length: 0
UWB PAN ID: 0xDECA
BLE Timeout: 9000
TX Power: 0x1F1F1F1F
Stream Mode: 1
Ranging Mode: 1
Format: ASCII
Range Extension: 0
```

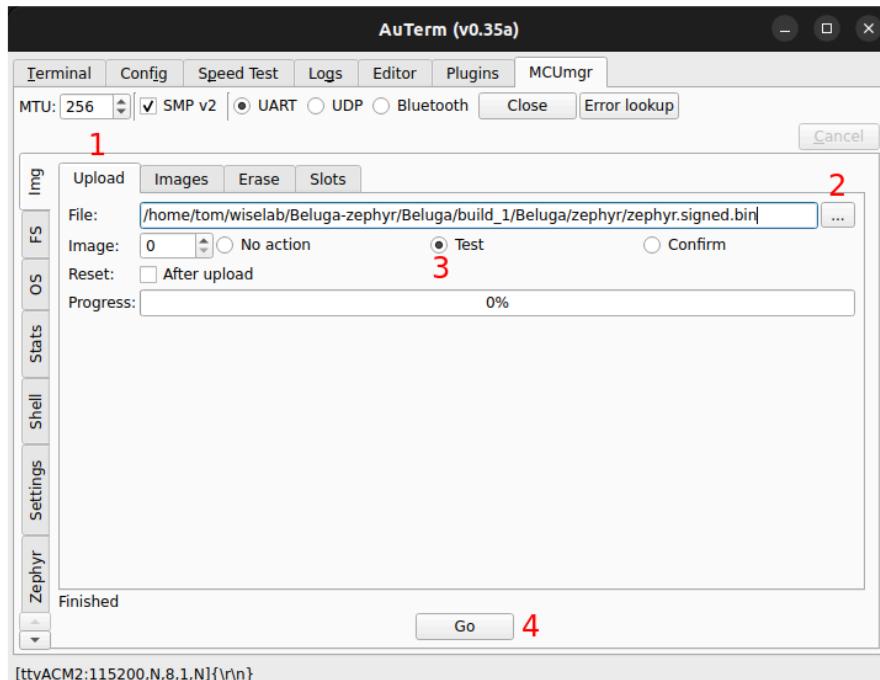
- b. If you see nothing in the terminal, then you can try running “AT+ID” to see if the settings have already been printed somewhere else. If no response is seen, then you may have the correct port.
6. Once you have a potentially correct port follow the instructions below:
- Open the MCUmgr tab
 - Open the images tab
 - Select state “Get”
 - Press “Go”



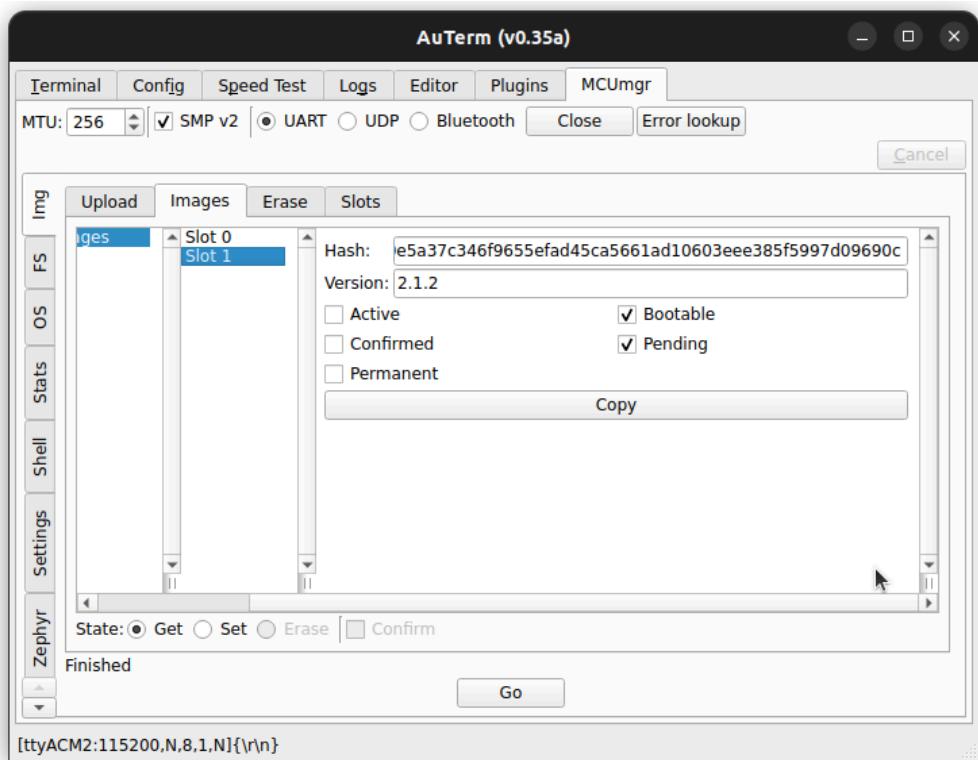
- e. If you see images (like the figure above), then you have the correct port
7. Switch to the upload tab and do the following:
- Select a binary file to upload
 - Make sure it is the signed file
 - This should be in beluga-2.0/Beluga/build*/Beluga/zephyr



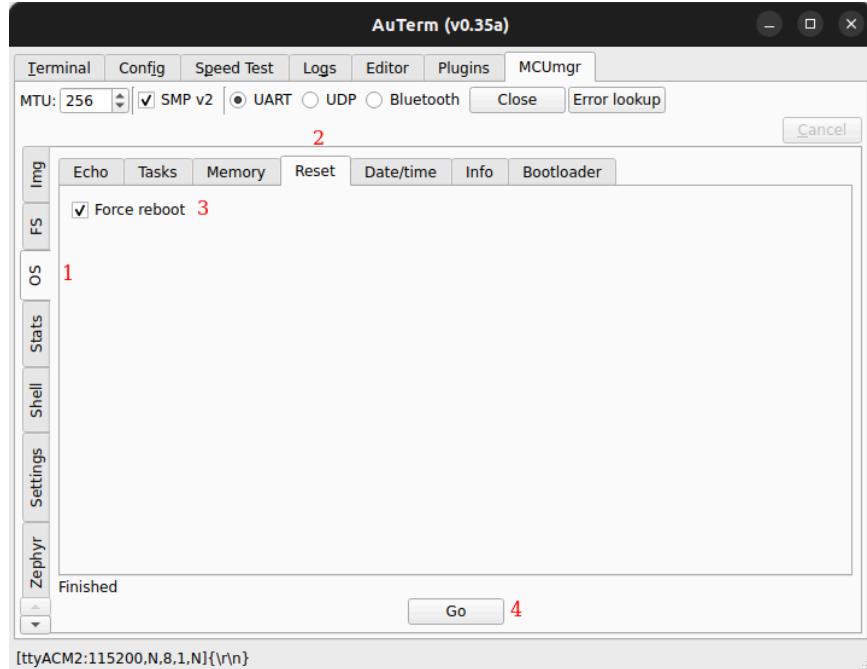
- Select "Test"
- Press "Go"
 - Sometimes it will time out. If it does, just retry.



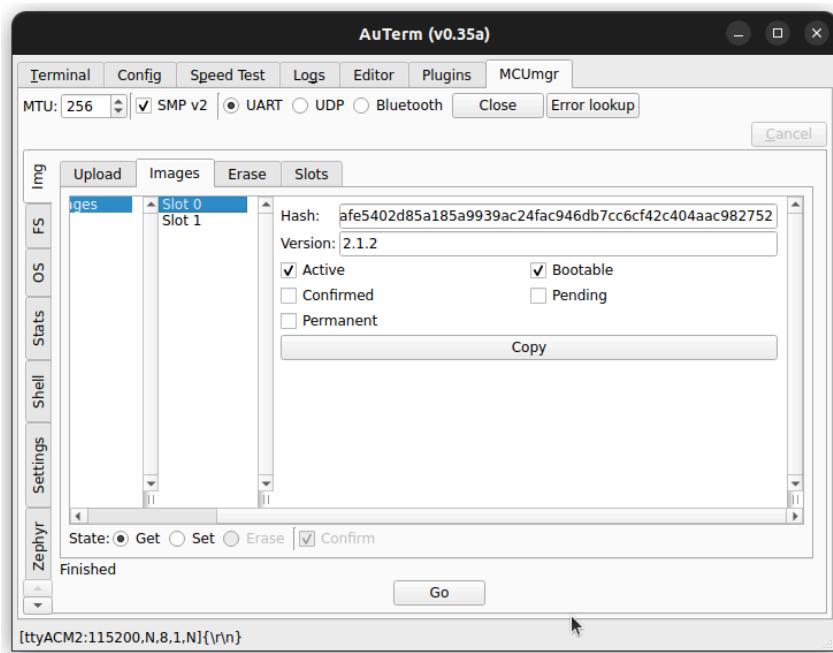
8. Go back to the images tab and rerun the steps in step 6. This time, you should see a pending image.



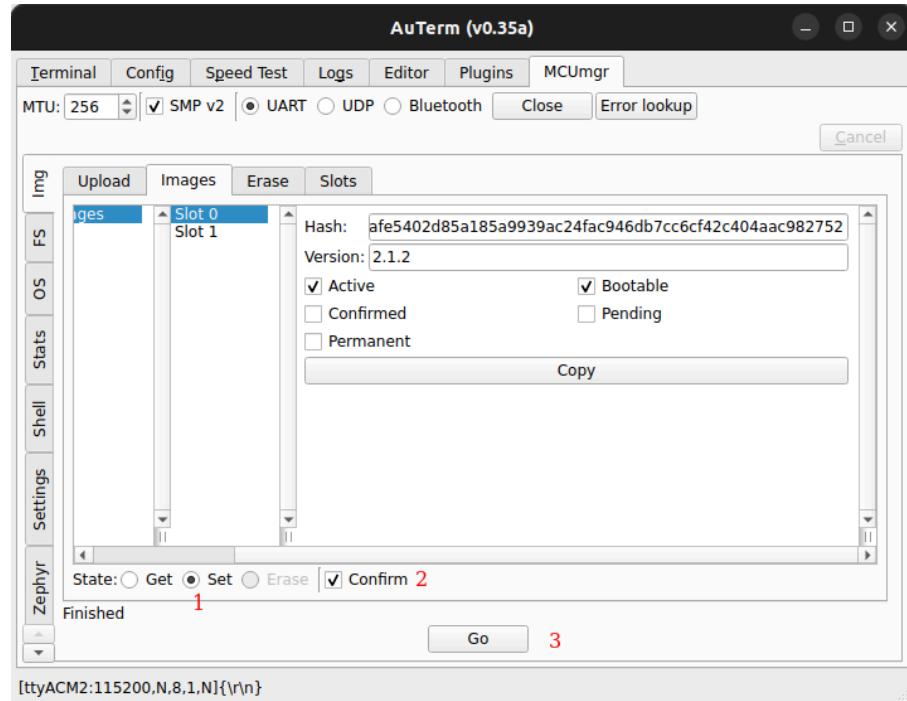
9. Switch to the OS tab
 - Select the Reset tab
 - Tick the “Force Reboot” box
 - Press “Go”



- d. The other option is to press the “RESET” button on the hardware
10. Once the device reboots, repeat the steps in step 6 and ensure the new version of the image is running.



11. Once you confirm the image is working correctly, you can confirm it by:
- Selecting the “Set” state
 - Tick the “Confirm” box
 - Clicking “Go”



- d. If the image is faulty, then repeat step 9 to revert back to the original image
 - i. If it continues to fail, try using the reset button

DFU Gotcha

The above method was tested and confirmed to work with the custom Beluga hardware. If you do not force a reboot in step 9, then it will ignore the test image. Also, if you use "AT+REBOOT", it will also ignore the test image.

imgflash

Imgflash is a CLI application developed for flashing devices over USB. It is located within the Beluga tree and can be installed with pip.

Installing

Before installing imgflash, ensure that you meet the following requirements:

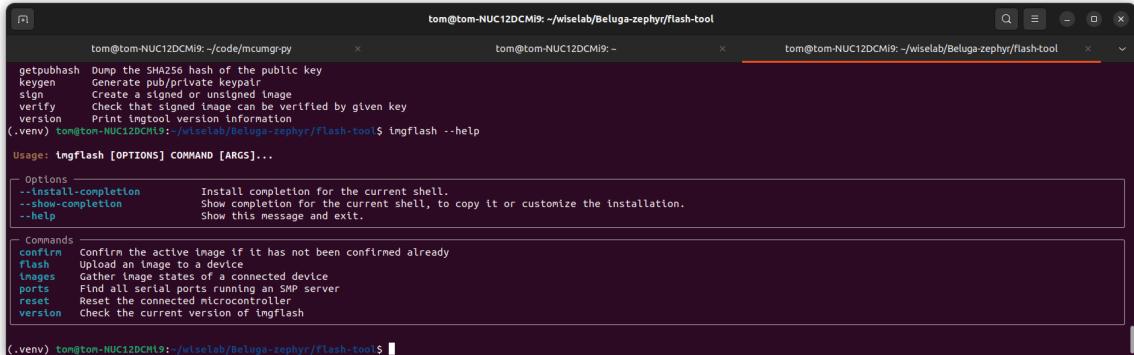
1. Python version >=3.9
2. Beluga repository is cloned

Once you have confirmed the requirements, follow the steps below

1. Open a terminal to the Beluga repository
2. Run the following commands
 - a. If on Windows, just google the Windows equivalents to these commands

```
Unset
cd flash-tool
python3 -m venv .venv
source .venv/bin/activate
pip install .
```

3. Confirm that imgflash successfully installed by running “imgflash --help”



The screenshot shows a terminal window with three tabs. The active tab displays the help output for the imgflash command. The output includes usage information and detailed descriptions for various options and commands. The options section lists --install-completion, --show-completion, and --help. The commands section lists confirm, flash, images, ports, reset, and version.

```
tom@tom-NUC12DCM19:~/code/mcumgr-py
tom@tom-NUC12DCM19:~/wiselab/Beluga-zephyr/flash-tool
tom@tom-NUC12DCM19:~/wiselab/Beluga-zephyr/flash-tool$ imgflash --help

Usage: imgflash [OPTIONS] COMMAND [ARGS]...

Options:
  --install-completion           Install completion for the current shell.
  --show-completion              Show completion for the current shell, to copy it or customize the installation.
  --help                          Show this message and exit.

Commands:
  confirm                         Confirm the active image if it has not been confirmed already
  flash                           Upload an image to a device
  images                          Gather image states of a connected device
  ports                           Find all serial ports running an SMP server
  reset                           Reset the connected microcontroller
  version                         Check the current version of imgflash

(.venv) tom@tom-NUC12DCM19:~/wiselab/Beluga-zephyr/flash-tool$
```

Commands

Imgflash has a few commands built already. The simplest command is “imgflash version” which just shows the version of the Python package. The other commands are a little more complex and are described below.

Ports

The ports command scans all of the serial ports on your computer and indicates which ports are running SMP servers. This is important because this application can only communicate with microcontrollers running SMP servers and it makes it easier to identify which ports are running SMP servers.

Options

Option	Type	Description	Default
--targets	TEXT	List of targets to search for. Targets are described as "Manufacturer Product". The list must be wrapped with quotes and each entry is separated by a comma. If no targets are specified, the application will test each port.	None
--log/--no-log		Turns on/off the logger output	--no-log
--help		Shows the help menu	

Example output

```
tom@tom-NUC12DCM19: ~/wiselab/Beluga-zephyr/flash-tool
Find all serial ports running an SMP server

Options --targets TEXT List of targets to search for. Targets are described as "Manufacturer Product". Separate targets with commas (,) [default: None]
--log --no-log Allow logger output [default: no-log]
--help Show this message and exit.

(.venv) tom@tom-NUC12DCM19:~/wiselab/Beluga-zephyr/flash-tool$ imgflash ports
This operation may take a while. Are you sure you want to proceed without specifying targets? [y/N]: y
Testing ports... 100% (/dev/ttyACM0 [33/33]) 0:02:40
(.venv) tom@tom-NUC12DCM19:~/wiselab/Beluga-zephyr/flash-tool$ imgflash ports
This operation may take a while. Are you sure you want to proceed without specifying targets? [y/N]: y
Testing ports... 100% (/dev/ttyACM2 [35/35]) 0:02:45
    CMU Beluga
/dev/ttyACM2

(.venv) tom@tom-NUC12DCM19:~/wiselab/Beluga-zephyr/flash-tool$ imgflash ports --targets "CMU Beluga"
Testing ports... 100% (/dev/ttyACM4 [4/4]) 0:00:05
    CMU Beluga
/dev/ttyACM2
/dev/ttyACM4

(.venv) tom@tom-NUC12DCM19:~/wiselab/Beluga-zephyr/flash-tool$
```

Images

The images command displays the image information on the connected devices. This can be for a specific device, a specific device type, or for all connected devices.

Options

Option	Type	Description	Default
--port/-p	TEXT	Serial port that communicates with the SMP server	None
--all		Display image information for all connected devices if specified	
--log/--no-log		Display logging	--no-log
--targets	TEXT	List of targets to search for. Targets are described as “Manufacturer Product”. The list must be wrapped with quotes and each entry is separated by a comma. If no targets are specified, the application will test each port.	None
--help		Shows help menu	

Example output

```
tom@tom-NUC12DCM19:~/code/mcumgr-py          tom@tom-NUC12DCM19:~          tom@tom-NUC12DCM19:~/wiselab/Beluga-zephyr/flash-tool
(.venv) tom@tom-NUC12DCM19:~/wiselab/Beluga-zephyr/flash-tool$ imgflash images --help
Usage: imgflash images [OPTIONS]

Gather image states of a connected device

Options
--port      -p          TEXT  Port communicates with the SMP server [default: None]
--all       --all        TEXT  Display the image information for all the connected devices
--log       --no-log     TEXT  Display logger information [default: no-log]
--targets   --targets    TEXT  List of targets to search for. Targets are described as "Manufacturer Product". Separate targets with commas (,) [default: None]
--help      --help       TEXT  Show this message and exit.

(.venv) tom@tom-NUC12DCM19:~/wiselab/Beluga-zephyr/flash-tool$ imgflash images -p /dev/ttyACM2
└─ CMU Beluga - /dev/ttyACM2


| Image Info | Slot 0                                                            | Slot 1                                                            |
|------------|-------------------------------------------------------------------|-------------------------------------------------------------------|
| Hash       | dde69c0ab1969b4229c65f88f434a37d49f3ffaa649cd2f2e877e6cf5c535c1ad | dde69c0ab1969b4229c65f88f434a37d49f3ffaa649cd2f2e877e6cf5c535c1ad |
| Version    | 2.1.5                                                             | 2.1.5                                                             |
| Active     | ✓                                                                 | ✗                                                                 |
| Bootable   | ✓                                                                 | ✓                                                                 |
| Confirmed  | ✓                                                                 | ✗                                                                 |
| Pending    | ✗                                                                 | ✗                                                                 |
| Permanent  | ✗                                                                 | ✗                                                                 |


(.venv) tom@tom-NUC12DCM19:~/wiselab/Beluga-zephyr/flash-tool$ imgflash images --targets "CMU Beluga"
Testing ports...
100% (/dev/ttyACM4 [4/4]) 0:00:05
Error: Please specify port with --port or -p or run with the --all flag
(.venv) tom@tom-NUC12DCM19:~/wiselab/Beluga-zephyr/flash-tools
```

```

tom@tom-NUC12DCM19:~/code/mcumgr-py
tom@tom-NUC12DCM19:~/wiselab/Beluga-zephyr/flash-tool
(.venv) tom@tom-NUC12DCM19:~/wiselab/Beluga-zephyr/flash-tool$ imgFlash images --targets "CMU Beluga" --all
Testing ports...
CMU Beluga - /dev/ttyACM4

```

Image Info	Slot 0	Slot 1
Hash	2ba6a0e20cce1f1458e1559764ec85d3190fe04fbf3cc9163b0813a9487c1dc5	dde69c0ab1969b4229c65f88f434a37d49f3ffaa649cd2f2e877e6fc535c1ad
Version	2.1.4	2.1.5
Active	✓	✗
Bootable	✓	✓
Confirmed	✓	✗
Pending	✗	✗
Permanent	✗	✗

Image Info	Slot 0	Slot 1
Hash	dde69c0ab1969b4229c65f88f434a37d49f3ffaa649cd2f2e877e6fc535c1ad	dde69c0ab1969b4229c65f88f434a37d49f3ffaa649cd2f2e877e6fc535c1ad
Version	2.1.5	2.1.5
Active	✓	✗
Bootable	✓	✓
Confirmed	✓	✗
Pending	✗	✗
Permanent	✗	✗

Flash

The flash command will upload an image onto the hardware running the SMP server. Unlike the previous commands, this command **requires** a serial port to be specified. Additionally, it requires a build directory and an application name to be specified (Application name is needed because it assumes that sysbuild was used to build multiple applications). If this command fails, make sure that the confirmed image is also the active image. The SMP server will not overwrite the confirmed image.

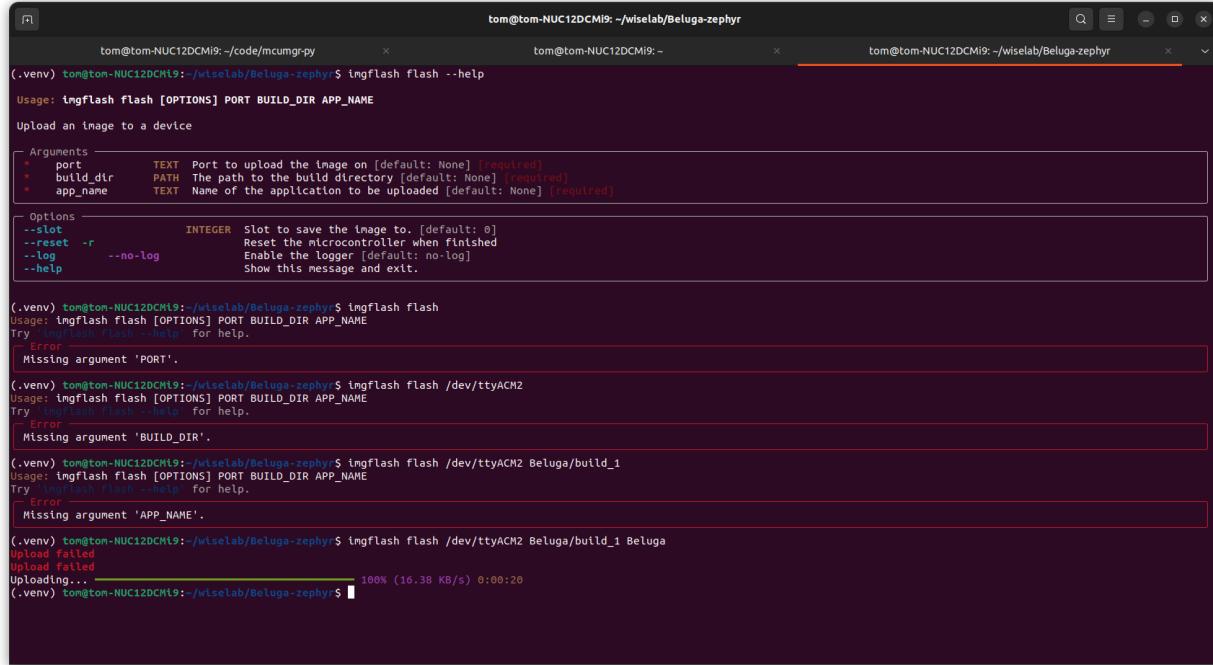
Arguments

Argument	Type	Description	Position
port	TEXT	The port to upload the image on.	1
Build directory	PATH	The file path to the build directory. This can be relative or absolute.	2
Application name	TEXT	The name of the application being uploaded.	3

Options

Option	Type	Description	Default
--slot	INTEGER	The slot to save the image to. This is only needed if the hardware needs to run more than 1 image.	0
--reset/-r		Reset the microcontroller after uploading the image.	
--log/--no-log		Enable or disable logging messages	--no-log
--help		Show help message	

Example output



The screenshot shows a terminal window with three tabs. The active tab displays the usage information for the `imgflash flash` command. It includes arguments for port, build directory, and application name, as well as options for slot, reset, log level, and help. Below this, several examples demonstrate the command's behavior with missing arguments, leading to errors. The other tabs show previous commands related to mcumgr-py and Beluga-zephyr.

```

tom@tom-NUC12DCM19:~/code/mcumgr-py
(.venv) tom@tom-NUC12DCM19:~/wiselab/Beluga-zephyr$ imgflash flash --help
Usage: imgflash flash [OPTIONS] PORT BUILD_DIR APP_NAME

Upload an image to a device

  Arguments
    * port      TEXT  Port to upload the image on [default: None] [required]
    * build_dir PATH  The path to the build directory [default: None] [required]
    * app_name   TEXT  Name of the application to be uploaded [default: None] [required]

  Options
    -slot      INTEGER Slot to save the image to. [default: 0]
    --reset   -r      Reset the microcontroller when finished
    --log     --no-log Enable the logger [default: no-log]
    --help

(.venv) tom@tom-NUC12DCM19:~/wiselab/Beluga-zephyr$ imgflash flash
Usage: imgflash flash [OPTIONS] PORT BUILD_DIR APP_NAME
Try 'imgflash flash --help' for help.
Error
Missing argument 'PORT'.

(.venv) tom@tom-NUC12DCM19:~/wiselab/Beluga-zephyr$ imgflash flash /dev/ttyACM2
Usage: imgflash flash [OPTIONS] PORT BUILD_DIR APP_NAME
Try 'imgflash flash --help' for help.
Error
Missing argument 'BUILD_DIR'.

(.venv) tom@tom-NUC12DCM19:~/wiselab/Beluga-zephyr$ imgflash flash /dev/ttyACM2 Beluga/build_1
Usage: imgflash flash [OPTIONS] PORT BUILD_DIR APP_NAME
Try 'imgflash flash --help' for help.
Error
Missing argument 'APP_NAME'.

(.venv) tom@tom-NUC12DCM19:~/wiselab/Beluga-zephyr$ imgflash flash /dev/ttyACM2 Beluga/build_1 Beluga
Upload failed
Upload failed
Uploading, ██████████ 100% (16.38 KB/s) 0:00:20
(.venv) tom@tom-NUC12DCM19:~/wiselab/Beluga-zephyr$ 

```

Images after uploading:

```
tom@tom-NUC12DCMi9:~/code/mcumgr-py          tom@tom-NUC12DCMi9:~/wiselab/Beluga-zephyr          tom@tom-NUC12DCMi9:~/wiselab/Beluga-zephyr
(.venv) tom@tom-NUC12DCMi9:~/wiselab/Beluga-zephyr$ imgflash images -p /dev/ttyACM2
CMU Beluga - /dev/ttyACM2
```

Image Info	Slot 0	Slot 1
Hash	2ba6a0e20cce1f1458e1559764ec85d3190fe04fbf3cc9163b0813a9487c1dc5	dde69c0ab1969b4229c65f88f434a37d49f3ffaa649cd2f2e877e6fc535c1ad
Version	2.1.4	2.1.5
Active	✓	✗
Bootable	✓	✓
Confirmed	✓	✗
Pending	✗	✓
Permanent	✗	✗

```
(.venv) tom@tom-NUC12DCMi9:~/wiselab/Beluga-zephyr$
```

Images after uploading if reset option is specified

```
tom@tom-NUC12DCMi9:~/code/mcumgr-py          tom@tom-NUC12DCMi9:~/wiselab/Beluga-zephyr          tom@tom-NUC12DCMi9:~/wiselab/Beluga-zephyr
(.venv) tom@tom-NUC12DCMi9:~/wiselab/Beluga-zephyr$ imgflash images -p /dev/ttyACM2
CMU Beluga - /dev/ttyACM2
```

Image Info	Slot 0	Slot 1
Hash	dde69c0ab1969b4229c65f88f434a37d49f3ffaa649cd2f2e877e6fc535c1ad	2ba6a0e20cce1f1458e1559764ec85d3190fe04fbf3cc9163b0813a9487c1dc5
Version	2.1.5	2.1.4
Active	✓	✗
Bootable	✓	✓
Confirmed	✗	✓
Pending	✗	✗
Permanent	✗	✗

```
(.venv) tom@tom-NUC12DCMi9:~/wiselab/Beluga-zephyr$
```

Reset

The reset command will reset the microcontroller through the SMP server. This is useful for cases where the reset flag was not specified/forgotten in the flash command or if the test image failed and the old image needs to be restored.

Arguments

Argument	Type	Description	Position
port	TEXT	Port that the SMP server is running on	1

Options

Options	Type	Description	Default
--force/-f		Force reboot	
--log/--no-log		Show logger output	--no-log
--help		Show help menu	

Example output

The screenshot shows a terminal window with three tabs. The active tab displays the usage information for the `imgflash reset` command:

```
Usage: imgflash reset [OPTIONS] PORT
Reset the connected microcontroller
```

Below the usage, the terminal shows the command being run with an invalid port:

```
(.venv) tom@tom-NUC12DCM19:/~/wiselab/Beluga-zephyr$ imgflash reset /dev/ttyACM1
Error
Given port is not running an SMP server
```

Then, it shows the command being run with a valid port, resulting in a success message:

```
(.venv) tom@tom-NUC12DCM19:/~/wiselab/Beluga-zephyr$ imgflash reset /dev/ttyACM2
Success
Reset complete
```

Notes

Sometimes, reset will not work. If this is the case, then 2 options are available:

1. Press the reset button instead
 - a. This is less ideal for situations where there is no human present
2. Confirm the image in slot 1 and rerun the reset command
 - a. This will guarantee that the test image will run on the next reboot, however, it risks bricking the microcontroller if the image is faulty

Confirm

The confirm command will confirm the test image that is currently running. If an image has already been confirmed, then it will have no effect.

Arguments

Argument	Type	Description	Position
port	TEXT	Port to confirm the image on	1

Options

Options	Type	Description	Default
--show/-l		Shows the image states after confirming	
--slot	INTEGER	Specifies which image to confirm via the image slot	0
--force/-f		Ignore the running and confirmed checks when marking an image for confirmation. This will not work for non-bootable images	
--log/--no-log		Show logger output	--no-log
--help		Show help menu	

Example output

```
tom@tom-NUC12DCM19:~/code/mcumgr-py
(.venv) tom@tom-NUC12DCM19:~/wiselab/BeLuga-zephyr$ imgflash confirm --help
Usage: imgflash confirm [OPTIONS] PORT
Confirm the active image if it has not been confirmed already

Arguments
  * port    TEXT Port to confirm the image on [default: None] [required]

Options
  --show -l      Show image slots after confirming
  --log        Show logger output [default: no-log]
  --help       Show this message and exit.

(.venv) tom@tom-NUC12DCM19:~/wiselab/BeLuga-zephyr$ imgflash confirm /dev/ttyACM2
Error
Cannot confirm image that has been confirmed already
(.venv) tom@tom-NUC12DCM19:~/wiselab/BeLuga-zephyr$
```

```
tom@tom-NUC12DCM19:~/code/mcumgr-py
(.venv) tom@tom-NUC12DCM19:~/wiselab/BeLuga-zephyr$ imgflash confirm /dev/ttyACM2
Error
Cannot confirm image that has been confirmed already
(.venv) tom@tom-NUC12DCM19:~/wiselab/BeLuga-zephyr$ imgflash flash /dev/ttyACM2 Beluga/build_1 Beluga
Upload failed
Upload failed
Uploading... 100% (16.48 KB/s) 0:00:20
(.venv) tom@tom-NUC12DCM19:~/wiselab/BeLuga-zephyr$ imgflash reset /dev/ttyACM2
Success
Reset complete
(.venv) tom@tom-NUC12DCM19:~/wiselab/BeLuga-zephyr$ imgflash confirm /dev/ttyACM2 -l
Success
Confirmed the image
CMU BeLuga - /dev/ttyACM2
Image Info Slot 0 Slot 1
Hash dde69cdab1969b4229c65f88f434a37d49f3ffa649cd2f2e877e6fcf535c1ad Zba6a0e20cce1f1458e1559764ec85d3196fe04fbf3cc9163b0813a9487c1dc5
Version 2.1.5 2.1.4
Active ✓ ✗
Bootable ✓ ✗
Confirmed ✓ ✗
Pending ✗ ✗
Permanent ✗ ✗
(.venv) tom@tom-NUC12DCM19:~/wiselab/BeLuga-zephyr$
```

Swap

The swap command will mark the image in slot 1 as pending. This is useful for instances where the test image fails to boot (this usually happens when the test image is larger than the active image). Instead of reuploading the test image, one can simply mark it as pending again.

Arguments

Argument	Type	Description	Position
port	TEXT	Port the SMP server is running on	1

Options

Options	Type	Description	Default
--show/-l		Shows the image states after confirming	
--log/--no-log		Show logger output	--no-log
--help		Show help menu	

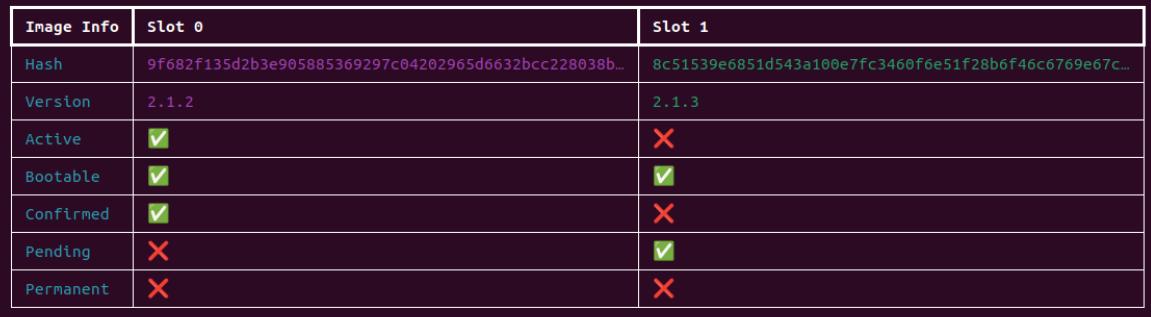
Example Output

```
tom@tom-NUC12DCM19:~/wiselab/Beluga-zephyr/flash-tool
(.venv) tom@tom-NUC12DCM19:~/wiselab/Beluga-zephyr/flash-tool$ imgflash swap --help
Usage: imgflash swap [OPTIONS] PORT
Mark the secondary image as pending

Arguments
* port      TEXT  The port the SMP server is running on [default: None] [required]

Options
--show -l      Show image slots after confirming
--log   --no-log  Show logger output [default: no-log]
--help            Show this message and exit.

(.venv) tom@tom-NUC12DCM19:~/wiselab/Beluga-zephyr/flash-tool$ imgflash swap /dev/ttyACM2
Success
Successfully marked the image as pending

(.venv) tom@tom-NUC12DCM19:~/wiselab/Beluga-zephyr/flash-tool$ imgflash swap /dev/ttyACM2 -l
CMU Beluga - /dev/ttyACM2

```