Beluga User Manual

Revision History

Revision	Date	Changes
v1.0	10/10/2025	Initial write

Firmware

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.

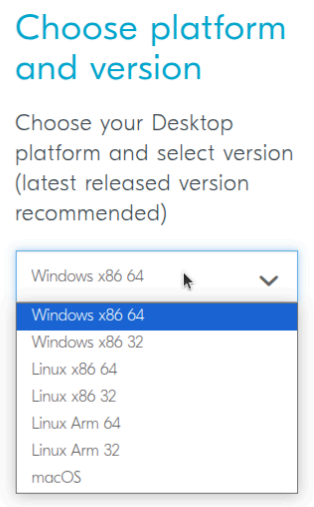
Tools to Install

- nRF Command Line Tools
- nRF Connect
- SEGGER J-Link
- VS code

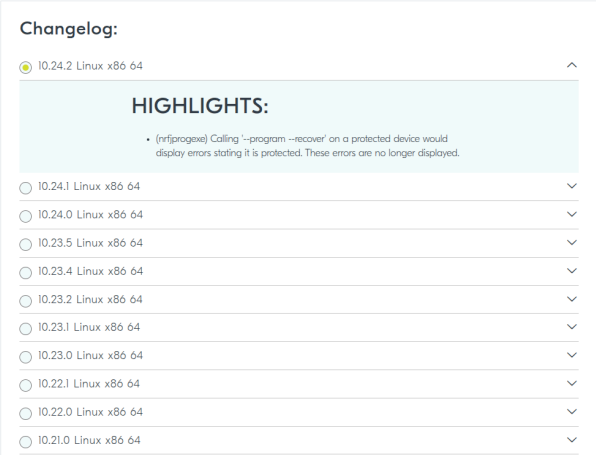
Installing nRF Command Line Tools and nRF Connect

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



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



4. Click on the download link

Additional Steps for nRF Connect (Linux)

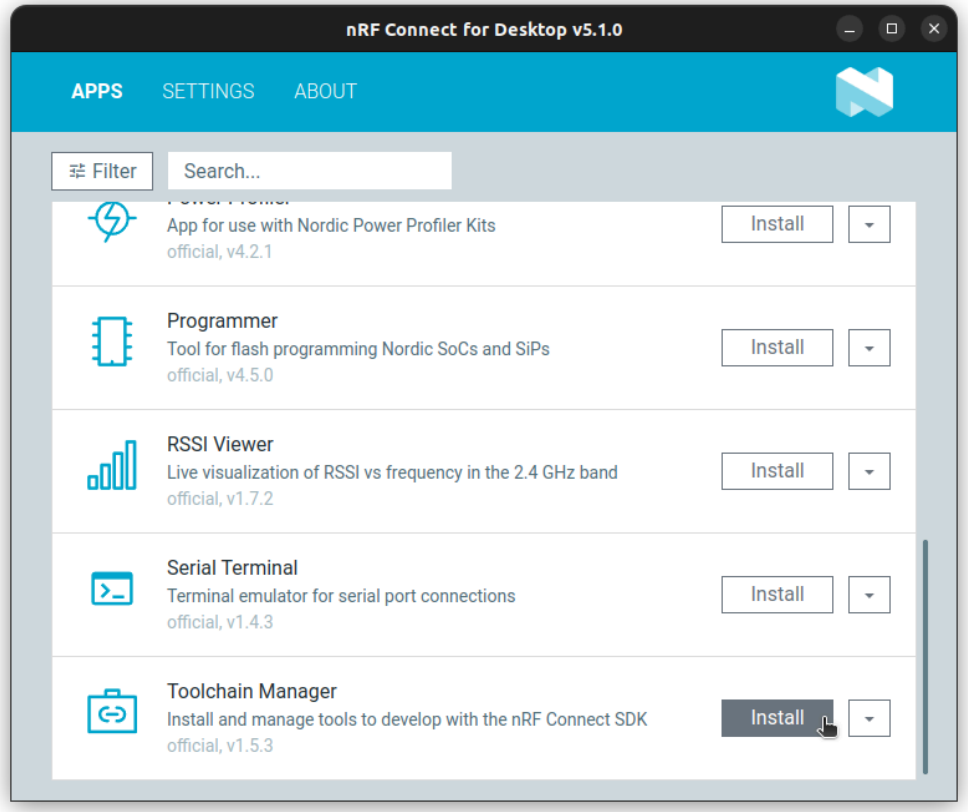
When you download nRF Connect for linux, it downloads as an app image. Before running it, you need to install libfuse2 and mark the file as executable.

```
None
sudo apt install libfuse2
sudo chmod +x nrfconnect-5.1.0-x86_64.appimage # Make sure to change image name
```

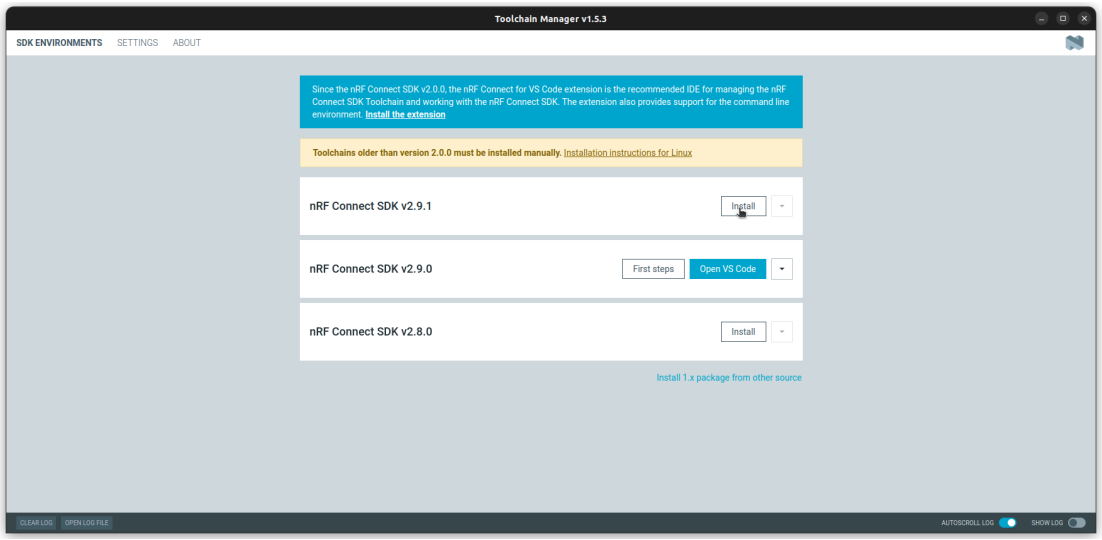
Installing the NCS toolchain

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”



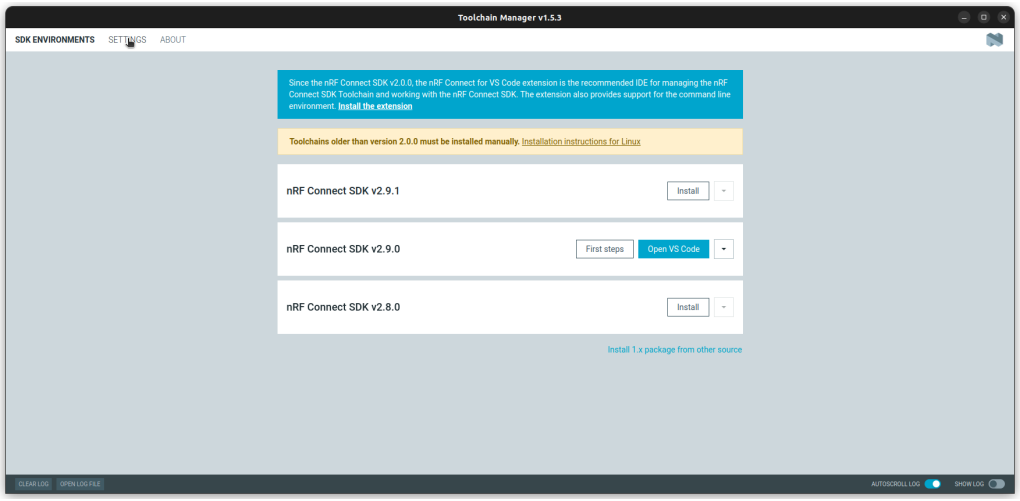
3. Open the Toolchain Manager and install nRF Connect SDK v2.9.0



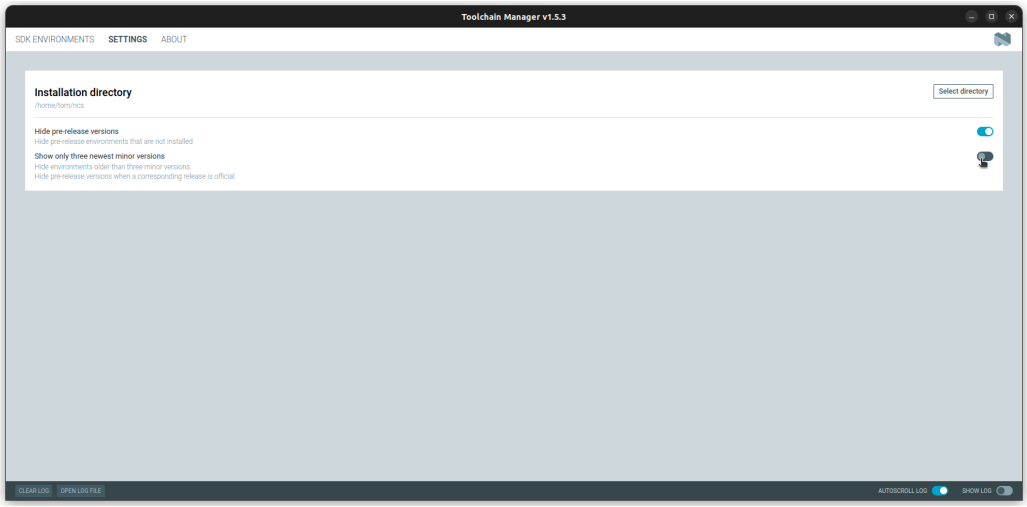
Cannot find v2.9.0? No Problem!

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

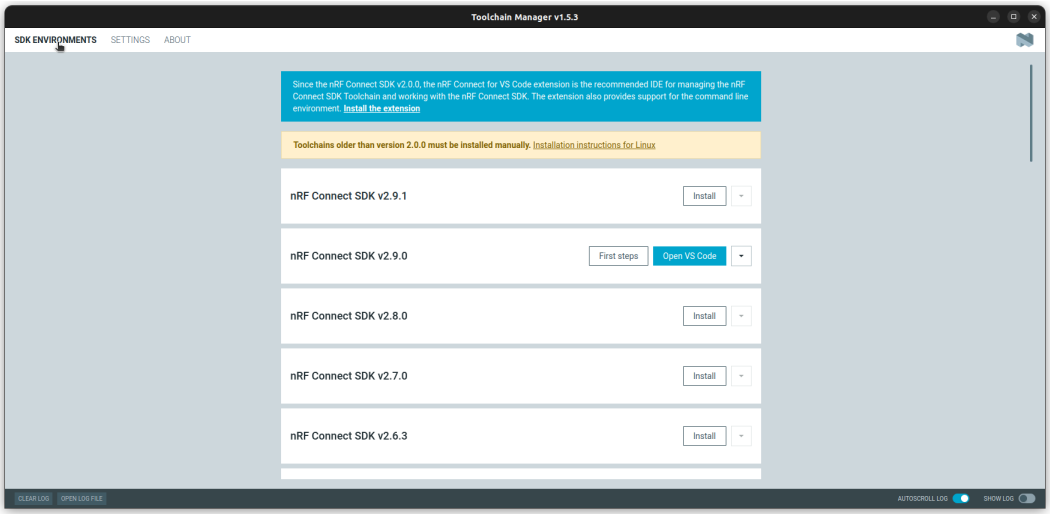
1. Click on the settings tab



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



3. Navigate back to the “SDK Environments” tab

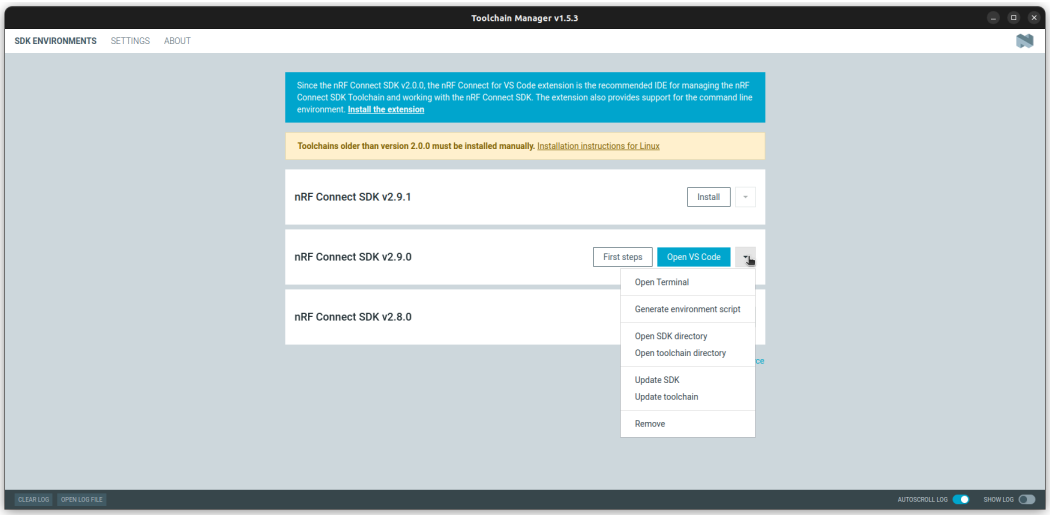


Now you can access versions no older than v2.0.0.

(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

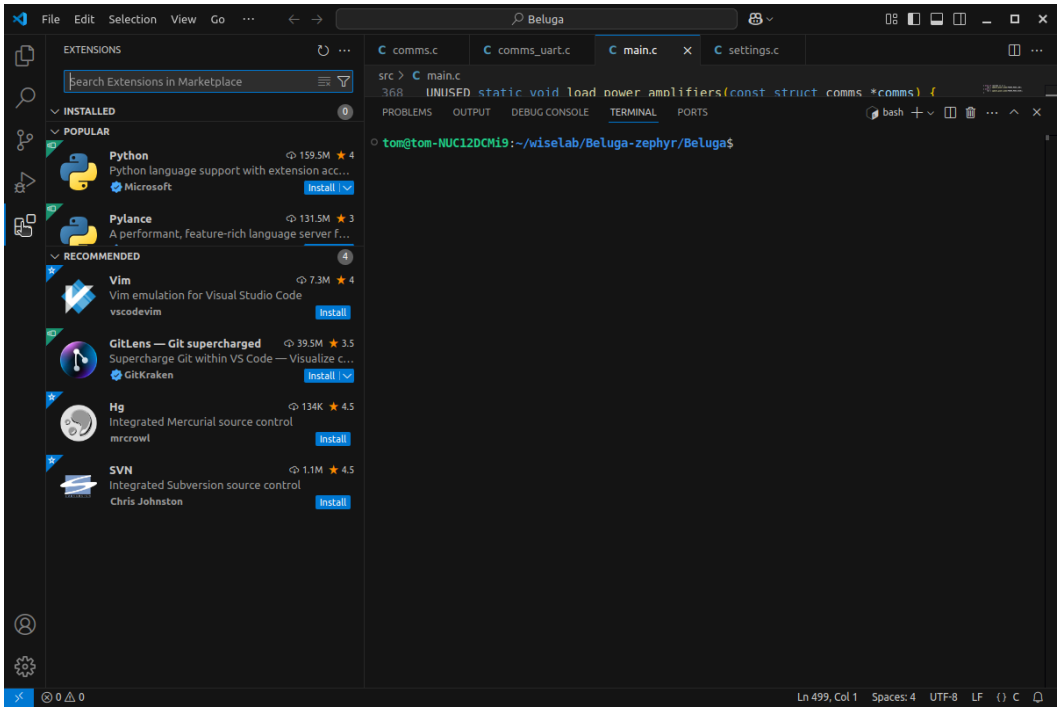


2. Click “Generate environment script”
3. Select the save location for the environment script

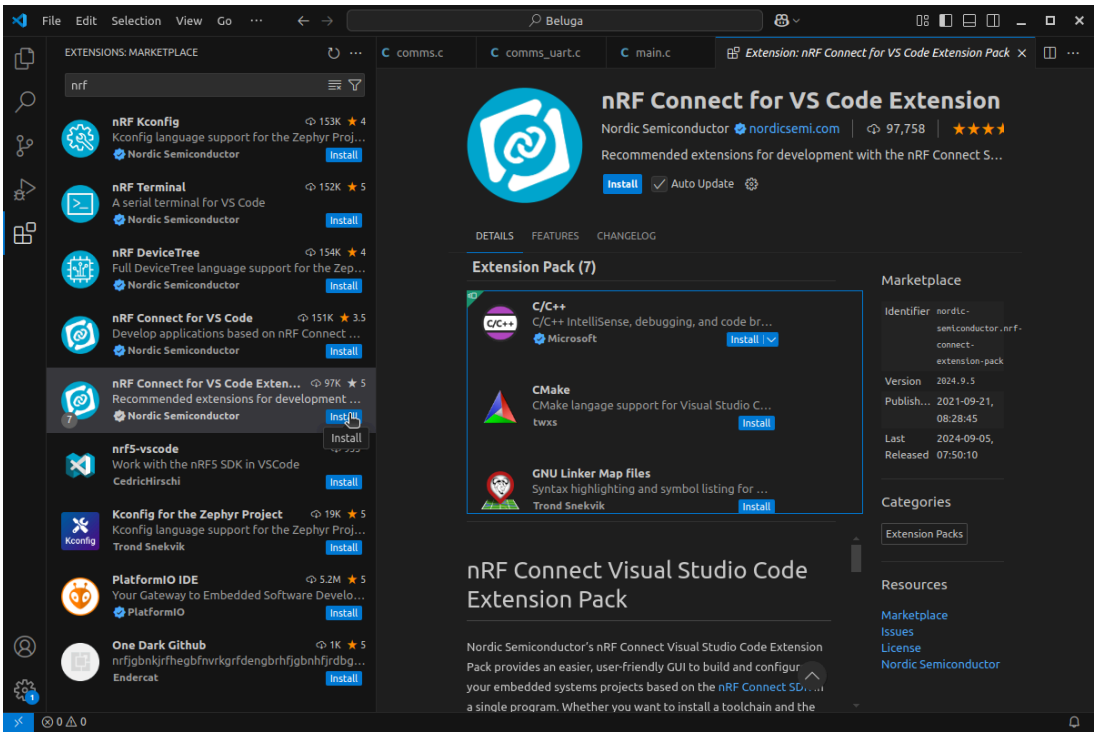
Installing VS code extension

1. Open VS code

2. Open “Extensions”



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

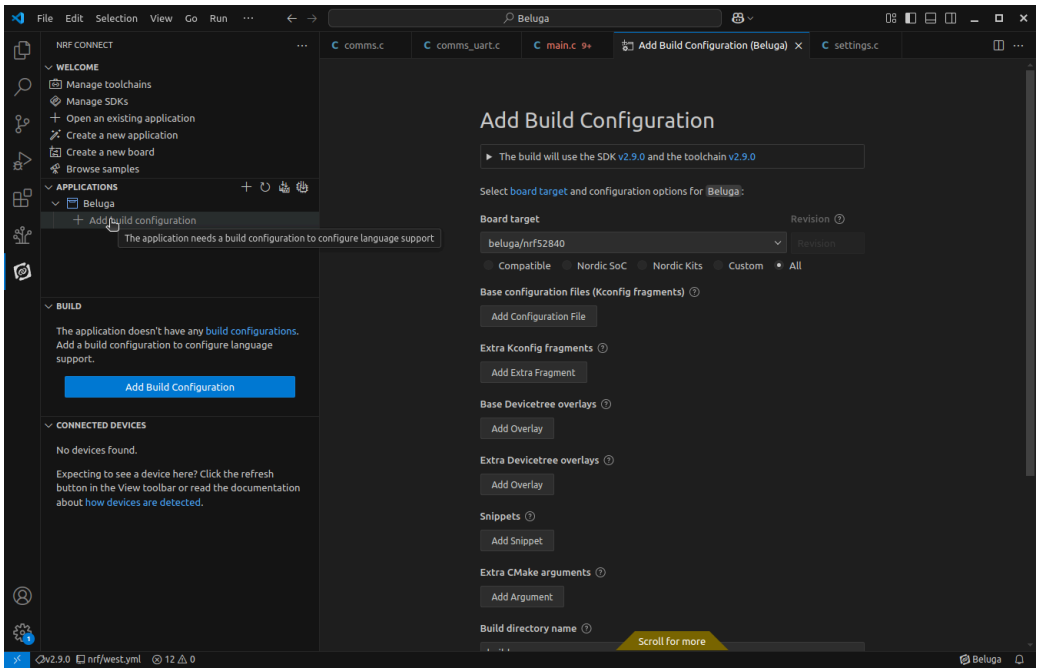


Configuring Builds

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

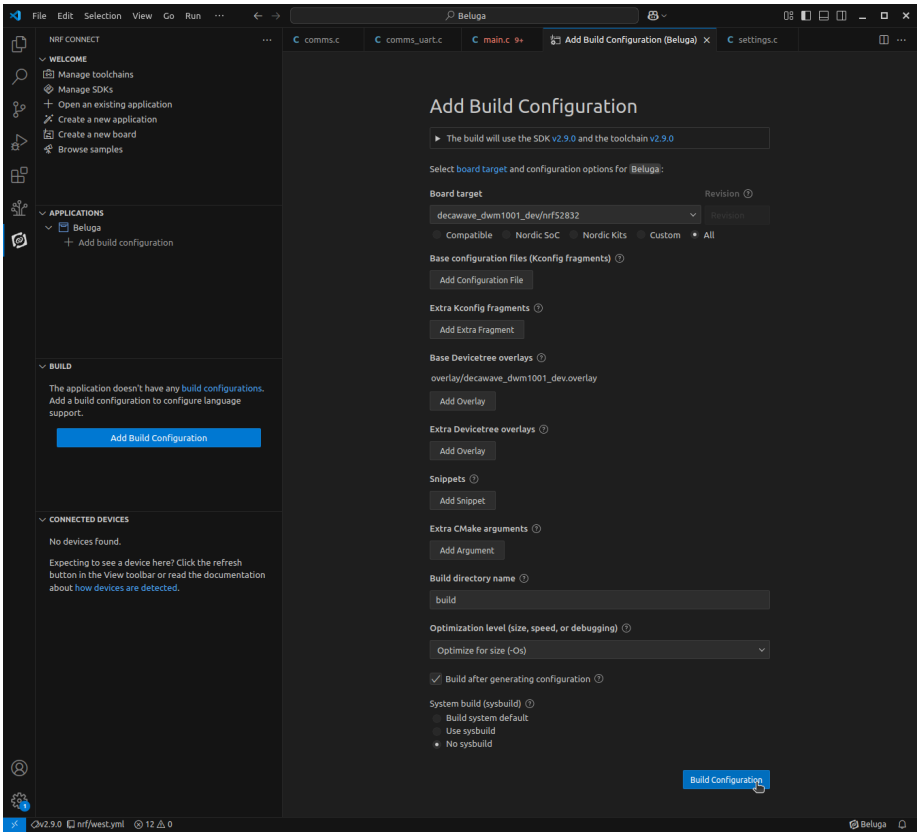
1. Open VS code
2. Switch to the nRF Connect tab (CTRL+ALT+N)



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

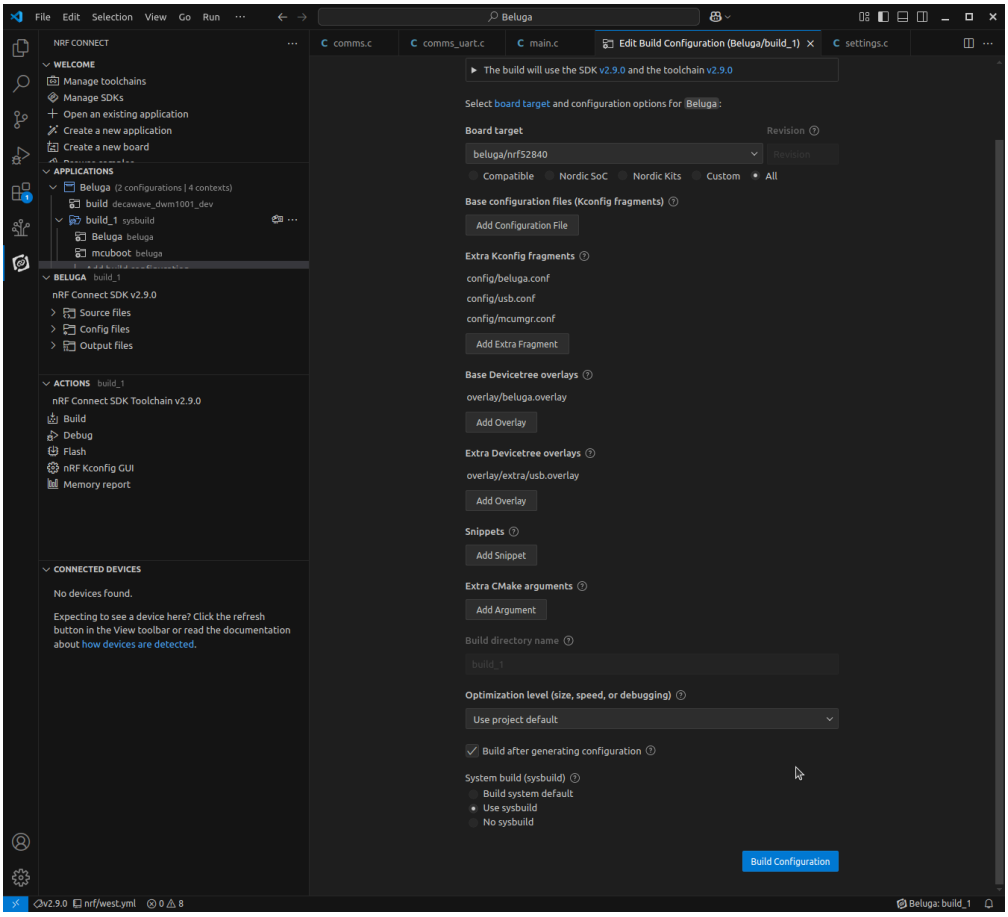
Decawave DWM1001 Dev Kit

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**

```
None
CONFIG_NORDIC_QSPI_NOR=y
CONFIG_BOOT_MAX_IMG_SECTORS=256
```

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

```
None
&mx25r64 {
        status = "okay";
};
```

```
/ {
        chosen {
                nordic,pm-ext-flash = &mx25r64;
        };
    };
```

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

```
None
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.
    a. Add “config/flash.conf” to **Extra Kconfig fragments**
    b. 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.

```
None
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:

```
None
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:

```
None
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 kay 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**:

```
None
SB_CONFIG_BOOT_SIGNATURE_KEY_FILE="\${APP_DIR}/keys/private_key.pem"
SB_CONFIG_BOOT_SIGNATURE_TYPE_ECDSA_P256=y
```

Configuring the firmware

The Beluga firmware can be configured during both build time and during runtime. This chapter will outline both the build time configurations and the runtime configurations.

Build Configurations

There are many build configurations for Beluga. Since there are many configurations, they won't be listed here, however, there are a few ways to view the build configurations which will be outlined here.

View the raw Kconfig files

This is the simplest way of viewing the different build configurations as there is no need to generate a build. However, it is necessary to know how Kconfig works. In Beluga, there is a main Kconfig file, and a few sub Kconfig files.

Menuconfig

Zephyr comes with a terminal-based interactive configuration editor called “menuconfig.” Unlike the previous method, you have to build the project, however, you get a decent looking interface that can be used to view and edit configurations. Follow the steps below:

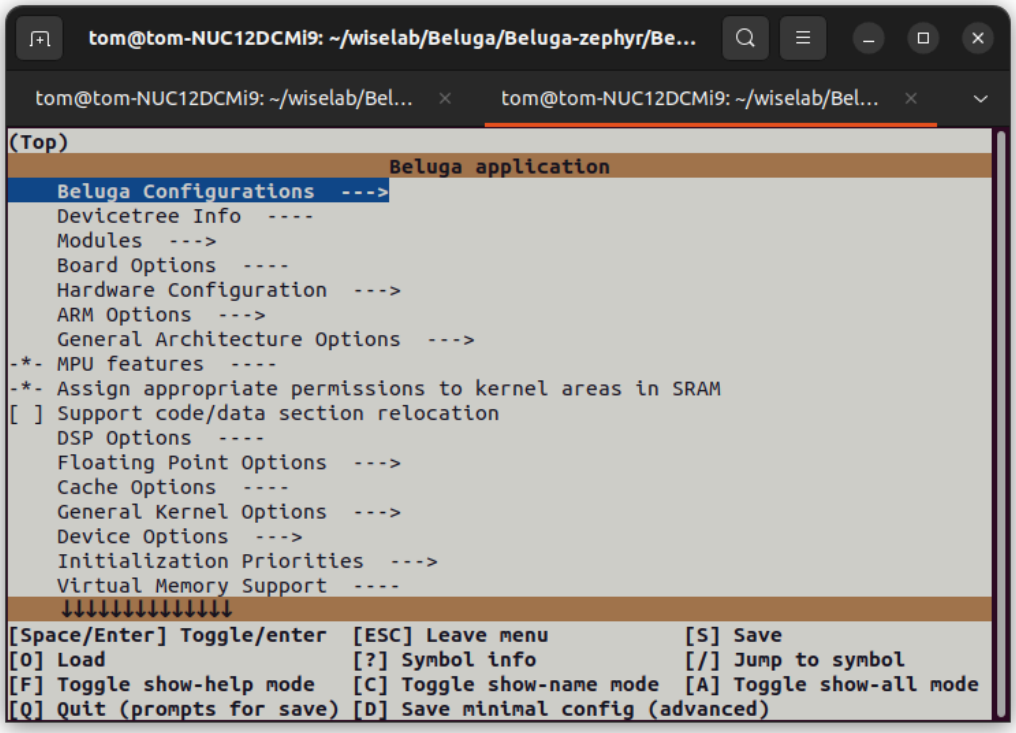
1. Open a terminal & source the ncs environment and zephyr environment.
2. Navigate to the Beluga directory within the repository
3. Run the build command below (make sure to replace the paths)

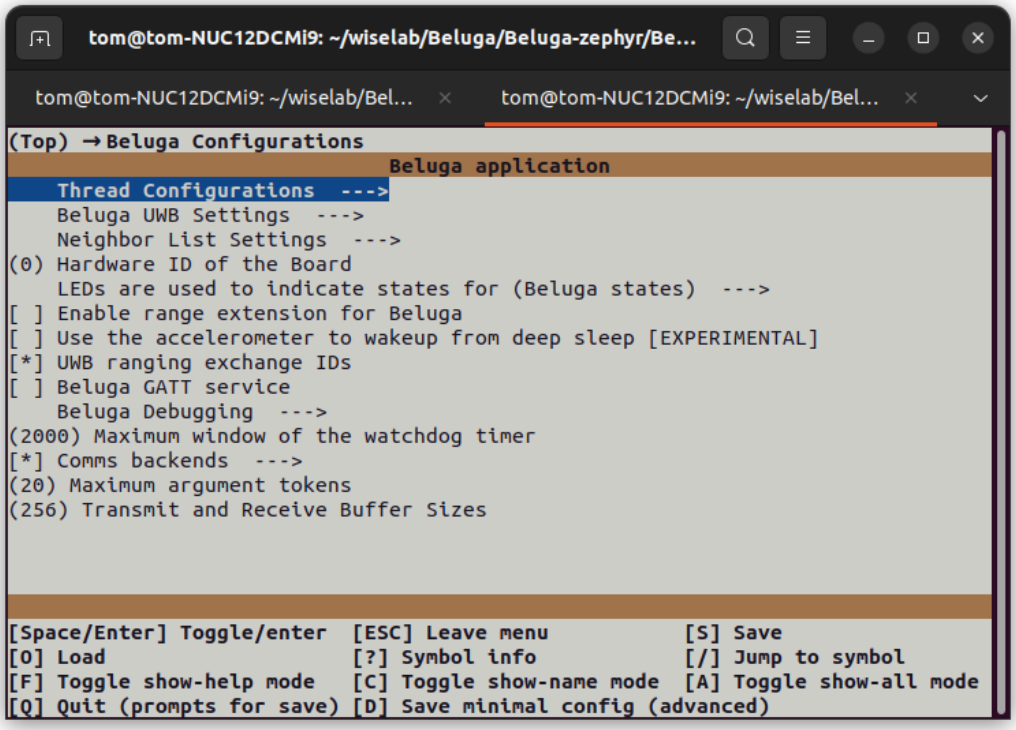
```
None
# DW1000
west build --build-dir <path to build directory> <path to Beluga directory>
--pristine --board decawave_dwm1001_dev/nrf52832 --no-sysbuild --
-DNCS_TOOLCHAIN_VERSION="NONE" -DCONF_FILE="prj.conf"
-DDTC_OVERLAY_FILE=overlay/decawave_dwm1001_dev.overlay
-DCONFIG_SIZE_OPTIMIZATIONS=y

# Beluga
west build --build-dir <path to build directory> <path to Beluga directory>
--pristine --board beluga/nrf52840 --sysbuild -- -DNCS_TOOLCHAIN_VERSION="NONE"
-DEXTRA_CONF_FILE="config/beluga.conf;config/mcumgr.conf;config/usb.conf"
-DCONF_FILE="prj.conf" -DEXTRA_DTC_OVERLAY_FILE="overlay/extra/usb.overlay"
-DDTC_OVERLAY_FILE="overlay/beluga.overlay" -DCONFIG_DEBUG_OPTIMIZATIONS=y
-DCONFIG_DEBUG_THREAD_INFO=y -DBOARD_ROOT="<path to repository>/Beluga"
```

4. (Optional) In builds that use sysbuild, navigate to <path to build directory>/Beluga
5. Run menuconfig with the following command

```
None
west build -t menuconfig
```


tom@tom-NUC12DCMi9: ~/wiselab/Beluga/Beluga-zephyr/Be...
tom@tom-NUC12DCMi9: ~/wiselab/Bel...
tom@tom-NUC12DCMi9: ~/wiselab/Bel...
(Top)
Beluga application
Beluga Configurations --->
Devicetree Info ----
Modules --->
Board Options ----
Hardware Configuration --->
ARM Options --->
General Architecture Options --->
-*- MPU features ----
-*- Assign appropriate permissions to kernel areas in SRAM
[ ] Support code/data section relocation
DSP Options ----
Floating Point Options --->
Cache Options ----
General Kernel Options --->
Device Options --->
Initialization Priorities --->
Virtual Memory Support ----
↓↓↓↓↓↓↓↓↓↓↓↓↓
[Space/Enter] Toggle/enter [ESC] Leave menu [S] Save
[O] Load [?] Symbol info [/] Jump to symbol
[F] Toggle show-help mode [C] Toggle show-name mode [A] Toggle show-all mode
[Q] Quit (prompts for save) [D] Save minimal config (advanced)


tom@tom-NUC12DCMi9: ~/wiselab/Beluga/Beluga-zephyr/Be...
tom@tom-NUC12DCMi9: ~/wiselab/Bel...
tom@tom-NUC12DCMi9: ~/wiselab/Bel...
(Top) → Beluga Configurations
Beluga application
Thread Configurations --->
Beluga UWB Settings --->
Neighbor List Settings --->
(0) Hardware ID of the Board
LEDs are used to indicate states for (Beluga states) --->
[ ] Enable range extension for Beluga
[ ] Use the accelerometer to wakeup from deep sleep [EXPERIMENTAL]
[*] UWB ranging exchange IDs
[ ] Beluga GATT service
Beluga Debugging --->
(2000) Maximum window of the watchdog timer
[*] Comms backends --->
(20) Maximum argument tokens
(256) Transmit and Receive Buffer Sizes
[Space/Enter] Toggle/enter [ESC] Leave menu [S] Save
[O] Load [?] Symbol info [/] Jump to symbol
[F] Toggle show-help mode [C] Toggle show-name mode [A] Toggle show-all mode
[Q] Quit (prompts for save) [D] Save minimal config (advanced)

Guiconfig

Zephyr also comes with another configuration editor called “guiconfig.” Like menuconfig, you have to build the project. NOTE: This does not work in the standard ncs environment. Follow the steps below:

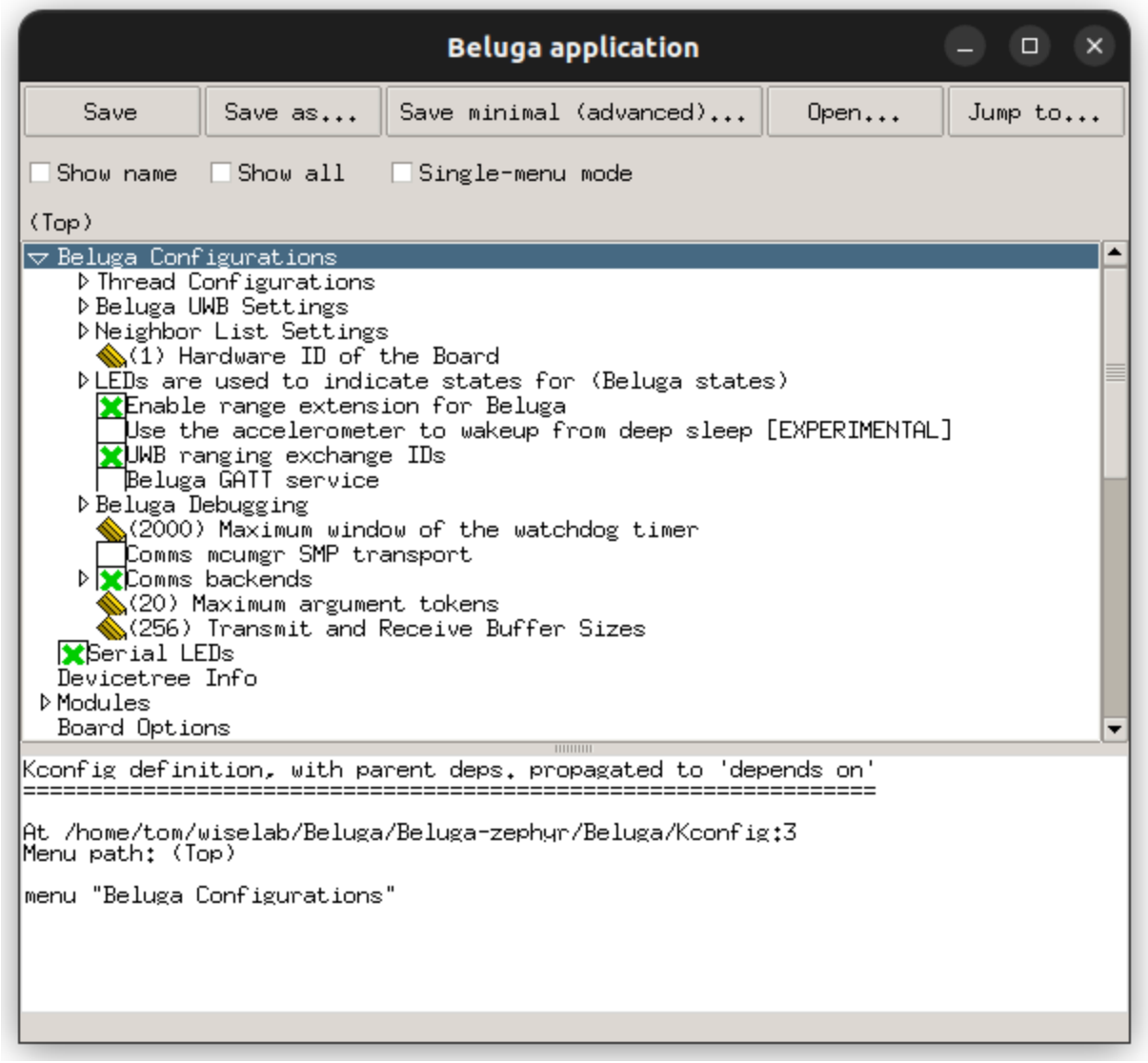
1. Open a terminal & source the zephyr environment.
2. Navigate to the Beluga directory within the repository
3. Run the build command below (make sure to replace the paths)

```
None
# DW1000
west build --build-dir <path to build directory> <path to Beluga directory>
--pristine --board decawave_dwm1001_dev/nrf52832 --no-sysbuild --
-DNCS_TOOLCHAIN_VERSION="NONE" -DCONF_FILE="prj.conf"
-DDTC_OVERLAY_FILE=overlay/decawave_dwm1001_dev.overlay
-DCONFIG_SIZE_OPTIMIZATIONS=y

# Beluga
west build --build-dir <path to build directory> <path to Beluga directory>
--pristine --board beluga/nrf52840 --sysbuild -- -DNCS_TOOLCHAIN_VERSION="NONE"
-DEXTRA_CONF_FILE="config/beluga.conf;config/mcumgr.conf;config/usb.conf"
-DCONF_FILE="prj.conf" -DEXTRA_DTC_OVERLAY_FILE="overlay/extra/usb.overlay"
-DDTC_OVERLAY_FILE="overlay/beluga.overlay" -DCONFIG_DEBUG_OPTIMIZATIONS=y
-DCONFIG_DEBUG_THREAD_INFO=y -DBOARD_ROOT="<path to repository>/Beluga"
```

4. (Optional) In builds that use sysbuild, navigate to <path to build directory>/Beluga
5. Run guiconfig with the following command

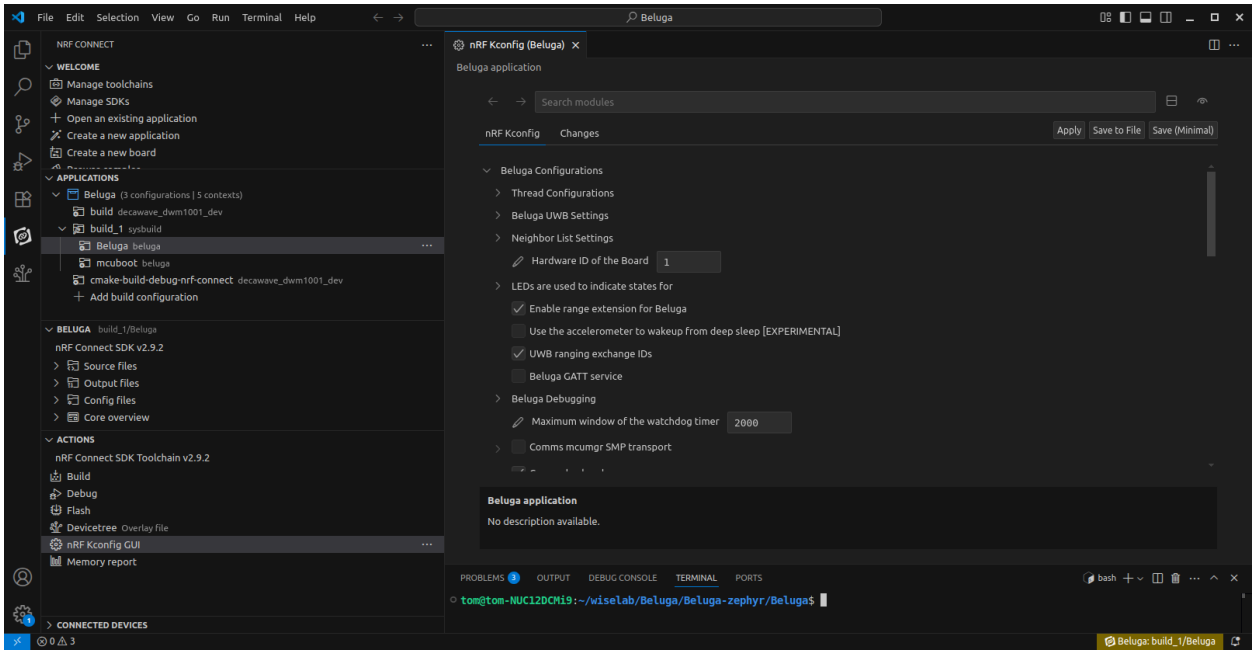
```
None
west build -t guiconfig
```



nRF Kconfig GUI

The final option is to use the nRF Kconfig GUI, which is packaged with their VS code environment (I honestly don't know why so many people like this crappy IDE). Follow the steps below:

1. Open the NCS extension
2. Select the application you want to edit the configs for
    a. For sysbuilds, click the drop down and select Beluga
3. Press the "nRF Kconfig GUI" button



Runtime Configurations and AT Commands

Beluga also comes with a few runtime configurations, some of which can be enabled during build time. These configurations are accessed and modified through the AT command interface. Below, each of the AT commands are outlined.

AT+ID

Sets or retrieves the node ID of the modem.

Usage

None

```
AT+ID <number>
AT+ID
```

Argument	Description
number	The node ID of the modem. This must be a positive, non-zero integer. Each ID should be unique in the network.

Properties

Property	Value
Readable	✅

Property	Value
Writable	✅
Saved	✅
Required Kconfig	-

AT+STARTUWB

Start the UWB initiator/responder

Usage

None

```
AT+STARTUWB
```

Properties

Property	Value
Readable	❌
Writable	✅
Saved	❌
Required Kconfig	-

AT+STOPUWB

Stop the UWB initiator/responder

Usage

None

```
AT+STOPUWB
```

Properties

Property	Value
Readable	❌
Writable	✅
Saved	❌

Property	Value
Required Kconfig	-

AT+STARTBLE

Start BLE advertising/scanning

Usage

None

```
AT+STARTBLE
```

Properties

Property	Value
Readable	❌
Writable	✅
Saved	❌
Required Kconfig	-

AT+STOPBLE

Stop BLE advertising/scanning

Usage

None

```
AT+STOPBLE
```

Properties

Property	Value
Readable	❌
Writable	✅
Saved	❌
Required Kconfig	-

AT+BOOTMODE

Determines how the node should behave when reset/powered on.

Usage

```
None
AT+BOOTMODE <mode>
AT+BOOTMODE
```

Argument	Description	Value	Mode Description
mode	The node ID of the modem. This must be a positive, non-zero integer. Each ID should be unique in the network.	0	(Default) Do nothing on startup (BLE and UWB off)
		1	Start BLE broadcasting/receiving on startup
		2	Start BLE and UWB on startup, full functionality.

Properties

Property	Value
Readable	✅
Writable	✅
Saved	✅
Required Kconfig	-

AT+RATE

Determines the rate that the node send poll messages

Usage

```
None
AT+RATE <period>
AT+RATE
```

Argument	Description	Related Kconfig
period	The rate (in ms) that the node initiates polling exchanges with neighbors.	MAX_POLLING_RATE

Argument	Description	Related Kconfig
	The input range is 0 - MAX_POLLING_RATE. A polling rate of 0 turns off the initiator and only allows the node to respond to ranging exchanges.	

Properties

Property	Value
Readable	✅
Writable	✅
Saved	✅
Required Kconfig	-

AT+CHANNEL

Determines which UWB channel to use for ranging.

Usage

None

```
AT+CHANNEL <channel>
AT+CHANNEL
```

Argument	Description	Value	Channel Description
channel	The channel to use for ranging exchanges	1	f_c = 2494.4 MHz, *BW* = 499.2 MHz
		2	f_c= 3993.6 MHz, *BW* = 499.2 MHz
		3	f_c= 4492.8 MHz, *BW* = 499.2 MHz
		4	f_c= 3993.6 MHz, *BW* = 1331.2 MHz[1]
		5	(Default) f_c= 6489.6 MHz, *BW* = 499.2 MHz
		7	f_c= 6489.6 MHz, *BW* = 1081.6[1]

[1] The DW1000 has a maximum receive bandwidth of 900 MHz

Properties

Property	Value
Readable	✅
Writable	✅
Saved	✅
Required Kconfig	-

AT+RESET

Reset all the saved settings back to their defaults.

Usage

```
None
AT+RESET
```

Properties

Property	Value
Readable	❌
Writable	✅
Saved	❌
Required Kconfig	-

AT+TIMEOUT

Determines how long a neighbor can stay in the neighbor list without any updates

Usage

```
None
AT+TIMEOUT <time>
AT+TIMEOUT
```

Argument	Description
time	The amount of time (in ms) that a node is allowed to stay in the neighbor list

Argument	Description
	without getting evicted. Default is 9000ms

Properties

Property	Value
Readable	✅
Writable	✅
Saved	✅
Required Kconfig	-

AT+TXPOWER

Adjusts the transmit power of the DW1000.

Usage

```
None
AT+TXPOWER <mode>
AT+TXPOWER <stage> <coarse gain> <fine gain>
AT+TXPOWER
```

The *stage*, *coarse gain*, and *fine gain* arguments allow for total control over the UWB transmit power.

Argument	Description	Value	Description
mode	Use a preset power setting.	0	(Default) Default TX power
		1	Maximum TX power
stage	The amplification stage to adjust the TX power	0	BOOSTNORM
		1	BOOSTP500
		2	BOOSTP250
		3	BOOSTP125
coarse gain	The coarse gain of the amplification stage	0	Off (No output)
		1	0 dB

Argument	Description	Value	Description
		2	2.5 dB
		3	5 dB
		4	7.5 dB
		5	10 dB
		6	12.5 dB
		7	15 dB
fine gain	The fine gain of the amplification stage. This argument has no effect when the coarse gain argument is *0*, however, it is still required. Whatever this setting is, the gain will be added to whatever the coarse gain is. For example, if coarse gain was entered to be 2 (2.5 dB) and this argument is 1 (0.5 dB), then the gain for the stage will be set to 2.5 dB + 0.5 dB = 3.0 dB.	0	0.0 dB
		1	0.5 dB
		2	1.0 dB
		3	1.5 dB
		4	2.0 dB
		5	2.5 dB
		6	3.0 dB
		7	3.5 dB
		8	4.0 dB
		9	4.5 dB
		10	5.0 dB
		11	5.5 dB
		12	6.0 dB
		13	6.5 dB
		14	7.0 dB
		15	7.5 dB
		16	8.0 dB
		17	8.5 dB
		18	9.0 dB

Argument	Description	Value	Description
		19	9.5 dB
		20	10.0 dB
		21	10.5 dB
		22	11.0 dB
		23	11.5 dB
		24	12.0 dB
		25	12.5 dB
		26	13.0 dB
		27	13.5 dB
		28	14.0 dB
		29	14.5 dB
		30	15.0 dB
		31	15.5 dB

Properties

Property	Value
Readable	✅
Writable	✅
Saved	✅
Required Kconfig	-

AT+STREAMMODE

Determines what neighbors to display

Usage

```
None
AT+STREAMMODE <mode>
```

```
AT+STREAMMODE
```

Argument	Description	Value	Mode description
mode	The neighbor display mode.	0	(Default) Display all neighbors, even those who have not been updated.
		1	Only display updated neighbors

Properties

Property	Value
Readable	✅
Writable	✅
Saved	✅
Required Kconfig	-

AT+TWRMODE

Determines the UWB ranging scheme

Usage

```
None
AT+TWRMODE <mode>
AT+TWRMODE
```

Argument	Description	Value	Mode description
mode	The UWB ranging mode	0	Single-sided two-way ranging (SS-TWR)
		1	(Default) Double-sided two-way ranging (DS-TWR)

Properties

Property	Value
Readable	✅
Writable	✅
Saved	✅
Required Kconfig	-

AT+LEDMODE

Determines the LED display mode

Usage

```
None
AT+LEDMODE <mode>
AT+LEDMODE
```

Argument	Description	Value	Mode description
mode	The LED mode	0	(Default) LEDs are allowed to be turned on
		1	All LEDs are turned off

Properties

Property	Value
Readable	✅
Writable	✅
Saved	✅
Required Kconfig	-

AT+REBOOT

Reboots Beluga. All internal states will be reset. This may result in a connection being lost between the host computer and the Beluga node.

Usage

```
None
AT+REBOOT
```

Properties

Property	Value
Readable	❌
Writable	✅
Saved	❌
Required Kconfig	-

AT+PWRAMP

Determines the state of the external power amplifiers

Usage

```
None
AT+PWRAMP <mode>
AT+PWRAMP
```

Argument	Description	Value	BLE Amplifier State	UWB Amplifier State
mode	The external power amplifier mode	0 (Default)	Off	Off
		1	Off	On
		2	On, 10dB gain	Off
		3	On, 10 dB gain	On
		4	On, 20 dB gain	Off
		5	On, 20 dB gain	On

Properties

Property	Value
Readable	✅
Writable	✅
Saved	✅
Required Kconfig	BELUGA_RANGE_EXTENSION

AT+ANTENNA

Determines which antenna to use for BLE

Usage

None

```
AT+ANTENNA <antenna>
AT+ANTENNA
```

Argument	Description	Value	Description
antenna	Select which antenna is used for BLE.	1	(Default) Use antenna 1
		2	Use antenna 2

Properties

Property	Value
Readable	✅
Writable	✅
Saved	❌
Required Kconfig	BELUGA_RANGE_EXTENSION

AT+TIME

Retrieve the current Beluga timestamp (ms since boot)

Usage

```
None
AT+TIME
```

Properties

Property	Value
Readable	✅
Writable	❌
Saved	❌
Required Kconfig	-

AT+FORMAT

Determine the output format mode

Usage

```
None
AT+FORMAT <mode>
AT+FORMAT
```

Argument	Description	Value	Mode description
mode	The format mode to use	0	(Default) CSV format
		1	JSON format with removed neighbors indicated with “rm <ID>”
		2	Framed Format

Properties

Property	Value
Readable	✅
Writable	✅
Saved	✅

Property	Value
Required Kconfig	-

AT+DEEPSLEEP

Places Beluga into deep sleep

Usage

None

```
AT+DEEPSLEEP
```

Properties

Property	Value
Readable	❌
Writable	✅
Saved	❌
Required Kconfig	-

AT+PHR

Determines whether extended frame lengths are allowed.

Usage

None

```
AT+PHR <mode>
AT+PHR
```

Argument	Description	Value	Mode description
mode	Allow transmit frame length extension	0	(Default) Use standard frame lengths
		1	Allow frame lengths up to 1023 bytes

Properties

Property	Value
Readable	✅
Writable	✅
Saved	✅
Required Kconfig	-

AT+DATARATE

Determines the data rate for the DW1000

Usage

```
None
AT+DATARATE <rate>
AT+DATARATE
```

Argument	Description	Mode	Mode description
rate	The data rate of the DW1000	0	(Default) 6.8 Mbits/s
		1	850 kbits/s
		2	110 kbits/s

Properties

Property	Value
Readable	✅
Writable	✅
Saved	✅
Required Kconfig	-

AT+PULSERATE

Determines the pulse repetition frequency of the DW1000

Usage

```
None
AT+PULSERATE <rate>
AT+PULSERATE
```

Argument	Description	Mode	Mode description
rate	The pulse repetition frequency to use	0	16 MHz
		1	(Default) 64 MHz

Properties

Property	Value
Readable	✅
Writable	✅
Saved	✅
Required Kconfig	-

AT+PREAMBLE

Determines the preamble length of the DW1000

Usage

```
None
AT+PREAMBLE <preamble>
AT+PREAMBLE
```

Argument	Description	Value	Description
preamble	The preamble length of the DW1000	64	64 symbols per transmission
		128	(Default) 128 symbols per transmission
		256	256 symbols per transmission
		512	512 symbols per transmission
		1024	1024 symbols per transmission

Argument	Description	Value	Description
		1536	1536 symbols per transmission
		2048	2048 symbols per transmission
		4096	4096 symbols per transmission

Properties

Property	Value
Readable	✅
Writable	✅
Saved	✅
Required Kconfig	-

AT+PAC

Determines the preamble acquisition chunk size for the DW1000

Usage

None

```
AT+PAC <size>
AT+PAC
```

Argument	Description	Value	Description
size	The size of the preamble acquisition chunk	0	(Default) 8 bytes (recommended for RX of preamble length 128 and below)
		1	16 bytes (recommended for RX of preamble length 256)
		2	32 bytes (recommended for RX of preamble length 512)
		3	64 bytes (recommended for RX of preamble length 1024 and up)

Properties

Property	Value
Readable	✅
Writable	✅
Saved	✅
Required Kconfig	-

AT+SFD

Determines the length of the start of frame delimiter on the DW1000

Usage

```
None
AT+SFD <mode>
AT+SFD
```

Argument	Description	Mode	Mode description
mode	The SFD mode to use	0	(Default) Standard SFD as defined in the IEEE802.15.4 standard
		1	DW Proprietary SFD

Properties

Property	Value
Readable	✅
Writable	✅
Saved	✅
Required Kconfig	-

AT+PANID

Determines the Personal Area Network (PAN) ID for the DW1000. This allows for different networks to be in the same area

Usage

```
None
AT+PANID <id>
AT+PANID
```

Argument	Description
id	The id of the personal area network. Allowable range: [0,65535]. Default: 57034.

Properties

Property	Value
Readable	✅
Writable	✅
Saved	✅
Required Kconfig	-

AT+EVICT

Determines the eviction policy of the neighbor list

Usage

```
None
AT+EVICT <scheme>
AT+EVICT
```

Argument	Description	Mode	Short	Description
mode	The eviction scheme to use when managing the neighbor list	0	Index RR	Use an index round-robin strategy. This will evict node 0 first, then node 1 on the next insertion, then node 2 on the following insertion. After evicting N - 1 nodes (N being the maximum number of nodes the neighbor list can hold), then node 0 will be evicted next. This

Argument	Description	Mode	Short	Description
				strategy is not recommended.
		1	(Default) RSSI	Evicts the node with the lowest RSSI. If there are no nodes with a lower RSSI than the scanned node, then no neighbors are evicted.
		2	Range	Evict the node with the largest ranging value.
		3	LRS	Evict the node that has not been scanned by the BLE in the longest amount of time.
		4	LRR	Evict the node that has been successfully ranged to in the longest amount of time.

Properties

Property	Value
Readable	✅
Writable	✅
Saved	✅
Required Kconfig	BELUGA_EVICT_RUNTIME_SELECT

AT+VERBOSE

Determine the response format of commands

Usage

None

```
AT+VERBOSE <mode>
AT+VERBOSE
```

Argument	Description	Mode	Mode description
mode	The verbose mode setting	0	(Default) Verbose mode turned off
		1	Verbose mode turned on

Properties

Property	Value
Readable	✅
Writable	✅
Saved	✅
Required Kconfig	-

AT+STATUS

Retrieve the firmware information and the current states

Usage

None

```
AT+STATUS
```

Status Return Fields		
Bitfield	Name	Description
31:12	-	Reserved
11	EVICT_ALGO	0: eviction algorithm is fixed to the build 1: eviction algorithm runtime selectable
10	ANT_SEL	0: Antenna 1 used 1: Antenna 2 used
9	UWB	0: UWB inactive 1: UWB active
8	BLE	0: BLE inactive 1: BLE active

Status Return Fields		
Bitfield	Name	Description
7:0	BOARD	The hardware platform ID. 0: DWM1001_dev 1: Beluga >=2: Unspecified

Properties

Property	Value
Readable	✅
Writable	❌
Saved	❌
Required Kconfig	-

AT+VERSION

Retrieve the firmware version

Usage

None

```
AT+VERSION
```

Properties

Property	Value
Readable	✅
Writable	❌
Saved	❌
Required Kconfig	-

AT+SYNC

Synchronize the wireless settings of a certain node on the network

Usage

```
None
AT+SYNC <id>
```

Argument	Description
id	The ID of the node on the network to synchronize

Properties

Property	Value
Readable	❌
Writable	✅
Saved	❌
Required Kconfig	-

AT+CALIBRATE

Calibrate the UWB antenna delays

Usage

```
None
AT+CALIBRATE <delay id> <calibration value
AT+CALIBRATE
```

<table>
<tr><td rowspan="2">Argument</td><td colspan="4">Description</td></tr>
<tr><td>Description</td><td>Value</td><td>PRF</td><td>Direction</td></tr>
<tr><td rowspan="4">delay id</td><td rowspan="4">The value to calibrate the antenna delay for</td><td>0</td><td>16 MHz</td><td rowspan="2">Receive</td></tr>
<tr><td>1</td><td>64 MHz</td></tr>
<tr><td>2</td><td>16 MHz</td><td rowspan="2">Transmit</td></tr>
<tr><td>3</td><td>64 MHz</td></tr>
<tr><td>calibration value</td><td colspan="4">The calibration value for the specified antenna delay</td></tr>
</table>

Properties

Property	Value
Readable	✅
Writable	✅
Saved	✅
Required Kconfig	-

AT+REASON

Retrieve the reboot reason

Usage

```
None
AT+REASON
```

Reset Reason Values	
Reason	Value[2]
External pin	1
Software reset	2
Brownout (drop in voltage)	4
Power-on reset (POR)	8
Watchdog timer expiration	16
Debug event	32
Security Violation	64
Waking up from low power mode	128
CPU lock-up detected	256
Parity error	512
PLL error	1024

[2] The values listed can be OR'ed together to give multiple reset reasons

Reset Reason Values	
Reason	Value[2]
Clock error	2048
Hardware reset	4096
User reset	8192
Temperature reset	16384
Bootloader reset (entry/exit)	32768
Flash ECC reset	65536

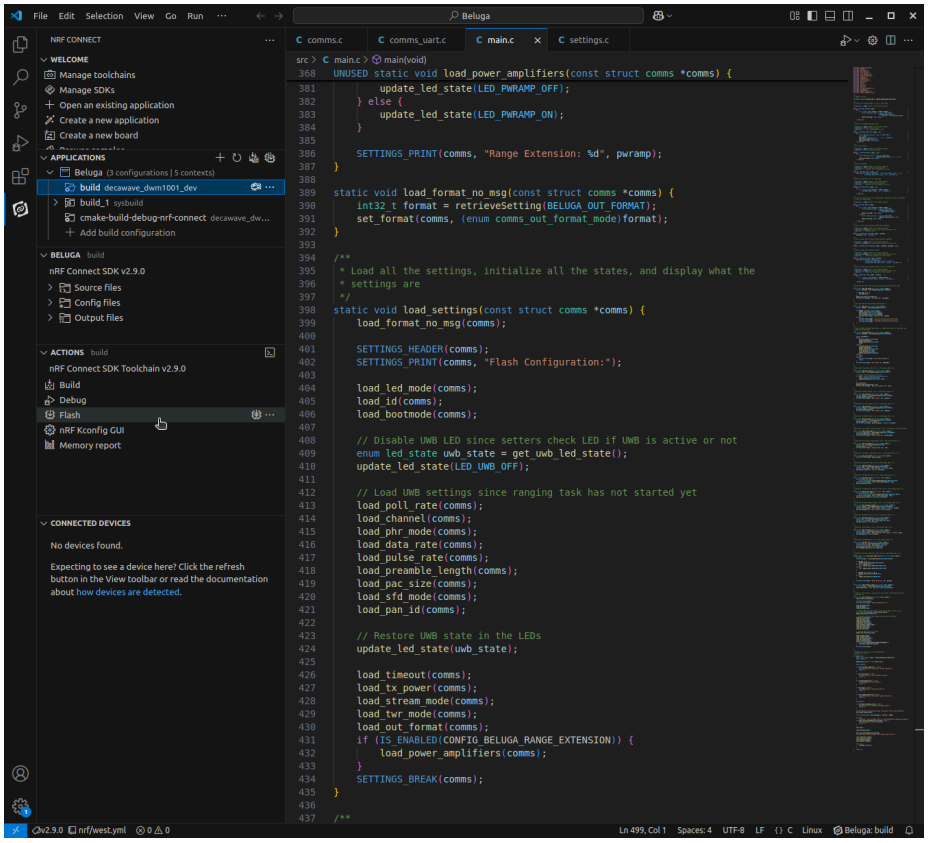
Properties

Property	Value
Readable	✅
Writable	❌
Saved	❌
Required Kconfig	BELUGA_RESET_REASON

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



Flashing onto the Custom Beluga Hardware

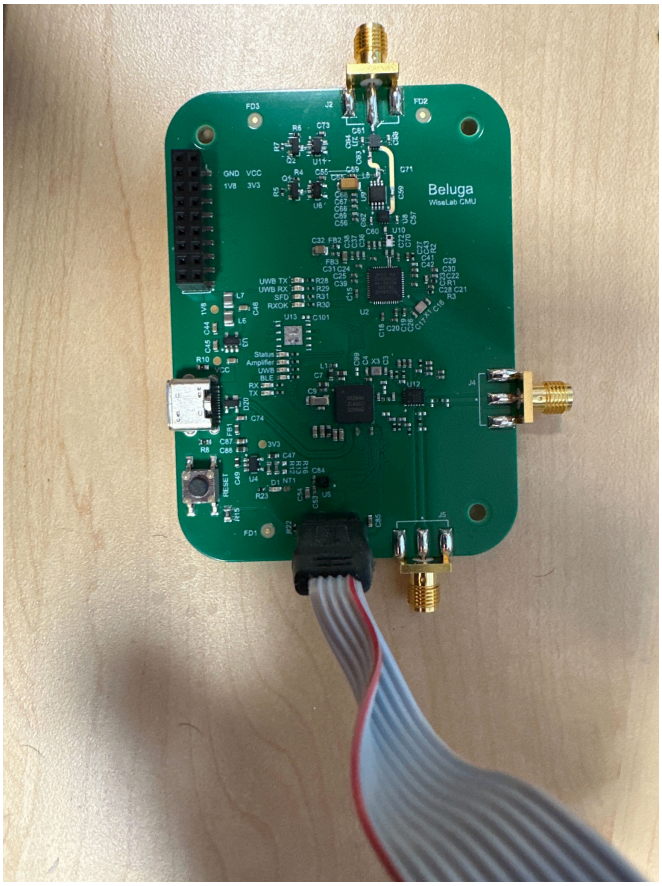
Flashing using JTAG is required for the first time. Any time after that can be done over DFU.

Using JTAG

Before flashing, make sure to select the Beluga image.

Using SEGGER

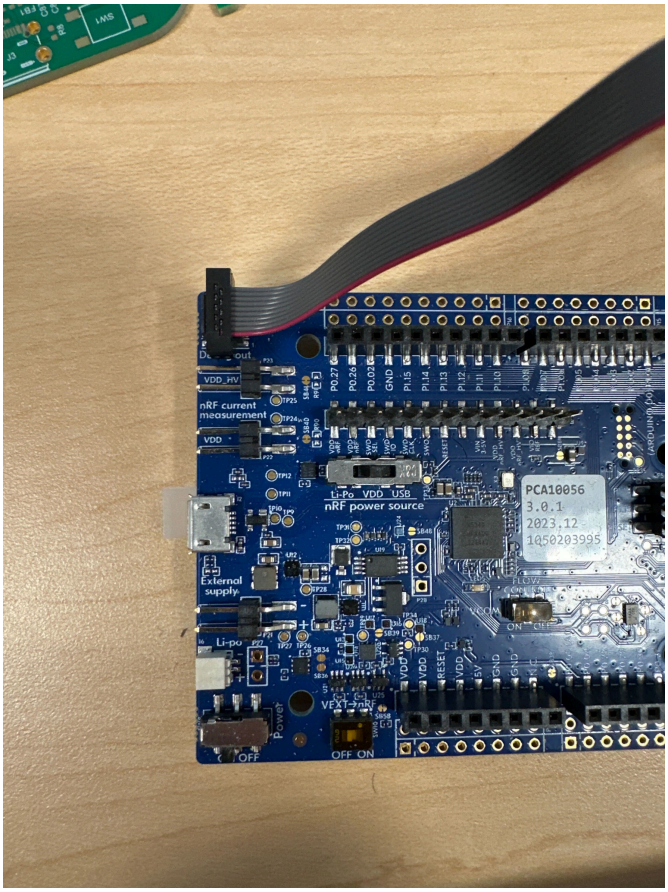
1. Plug the SEGGER J-Link into your computer
2. Connect the J-Link to the board using the Tag-Connect TC2050-IDC



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 (if applicable)
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

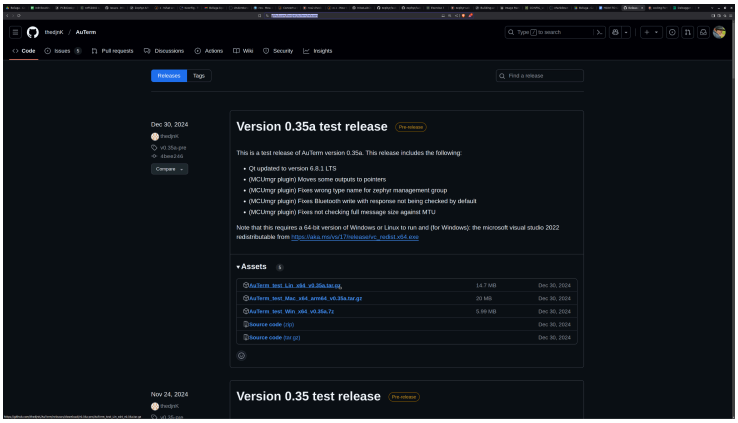
After flashing using JTAG, the custom Beluga hardware will be able to be flashed over USB. This will require an SMP client to do. There are a few tools and libraries that Zephyr recommends, all of which can be found here. If the web page is no longer working, reference the table below:

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

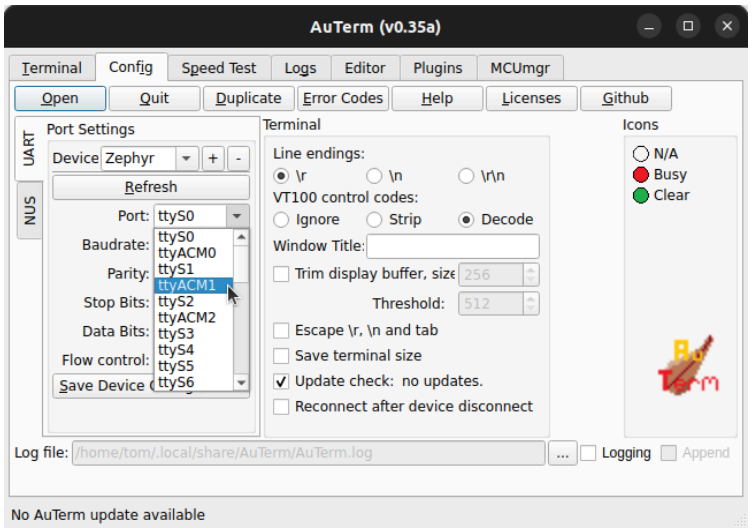
smpclient	Yes	Yes	Yes	Library	Python	Apache-2.0
Zephyr MCUmgr client (in-tree)	No	Yes	No	Library	C	Apache-2.0

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.

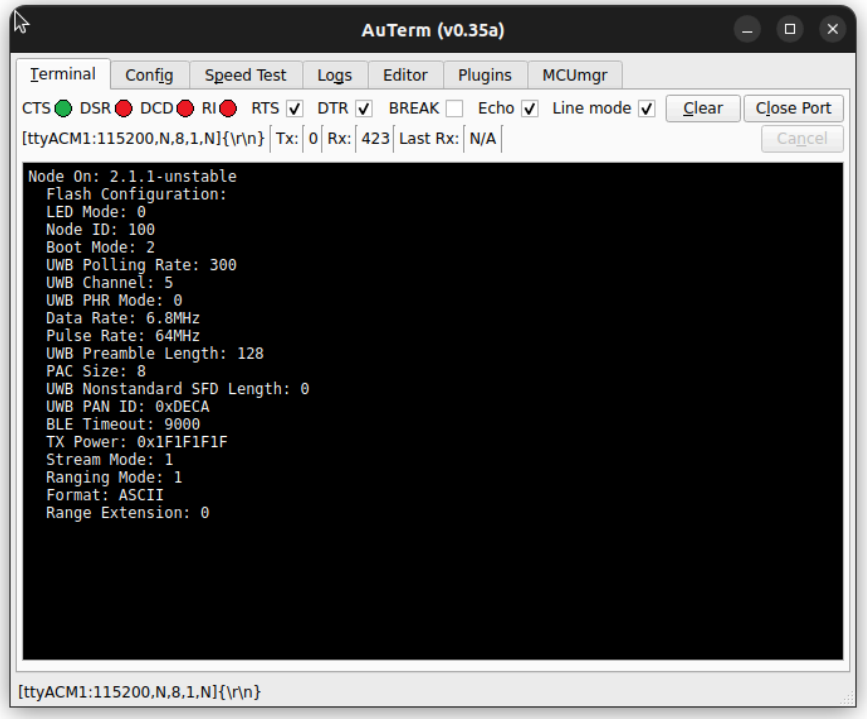
1. Download the latest release of AuTerm from the releases page (At the time of writing, there were no releases, only pre-releases).
    a. Download the build for your operating system



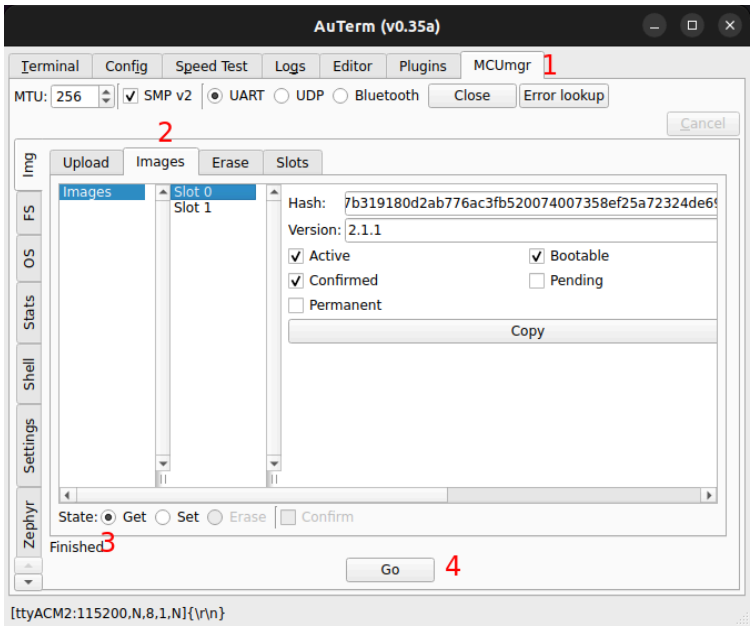
2. Extract the binary
3. Launch AuTerm
    a. For Linux and maybe macOS, you need to mark the binary as executable
4. Press refresh and click on the dropdown list to find the serial port to connect to
    a. On Linux, it should be one of the ttyACM ports
    b. On MacOS, it should be one of the <insert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

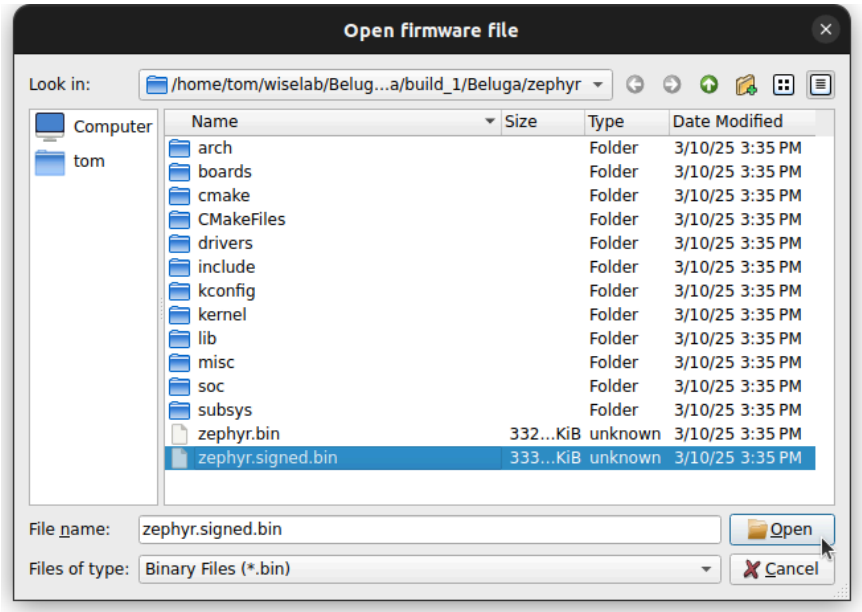


    b. If you see nothing in the terminal, then you can try running “AT+ID” to see if the settings have already been printed somewhere else. If no response is seen, then you may have the correct port.
6. Once you have a potentially correct port follow the instructions below:
    a. Open the MCUmgr tab
 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

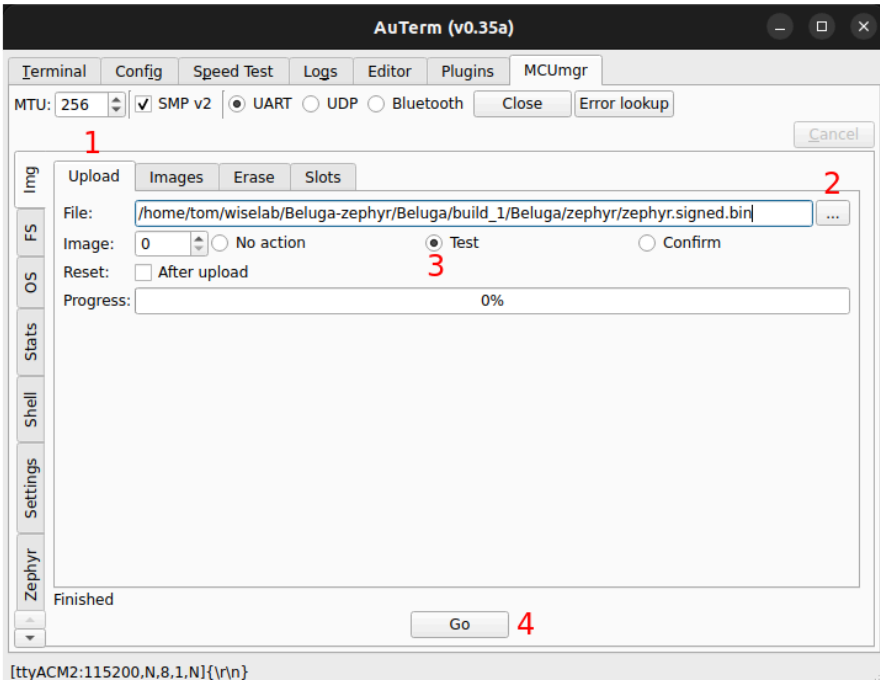


e. If you see images (like the figure above), then you have the correct port

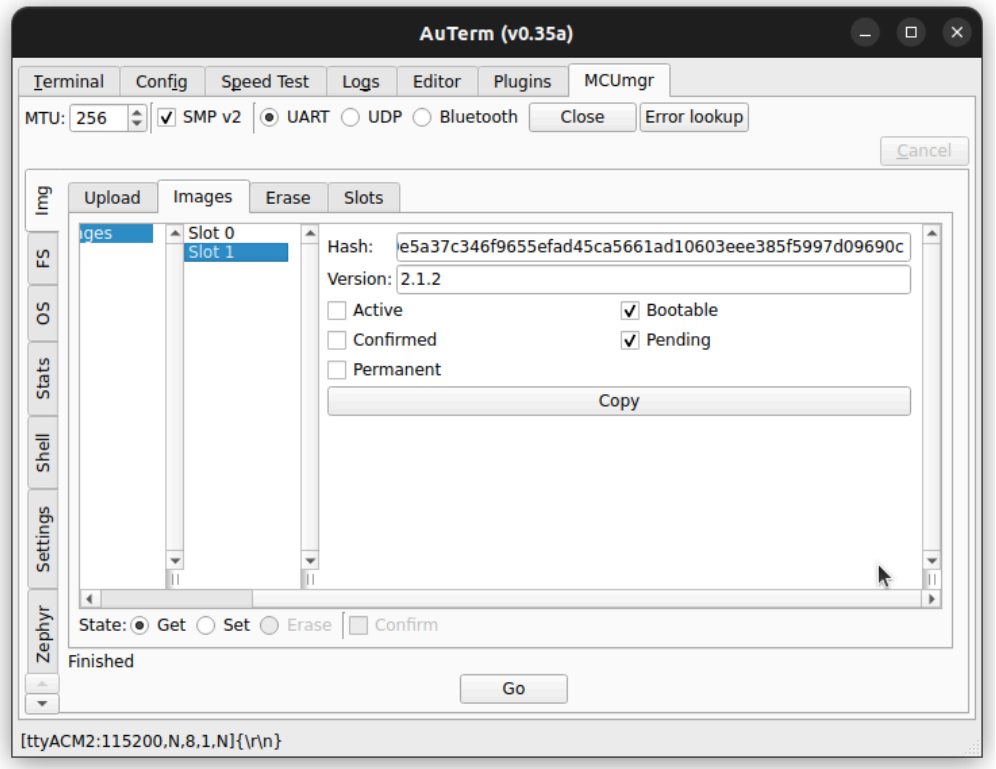
7. Switch to the upload tab and do the following:
   a. Select a binary file to upload
      i. Make sure it is the signed file
      ii. This should be in beluga-2.0/Beluga/build*/Beluga/zephyr



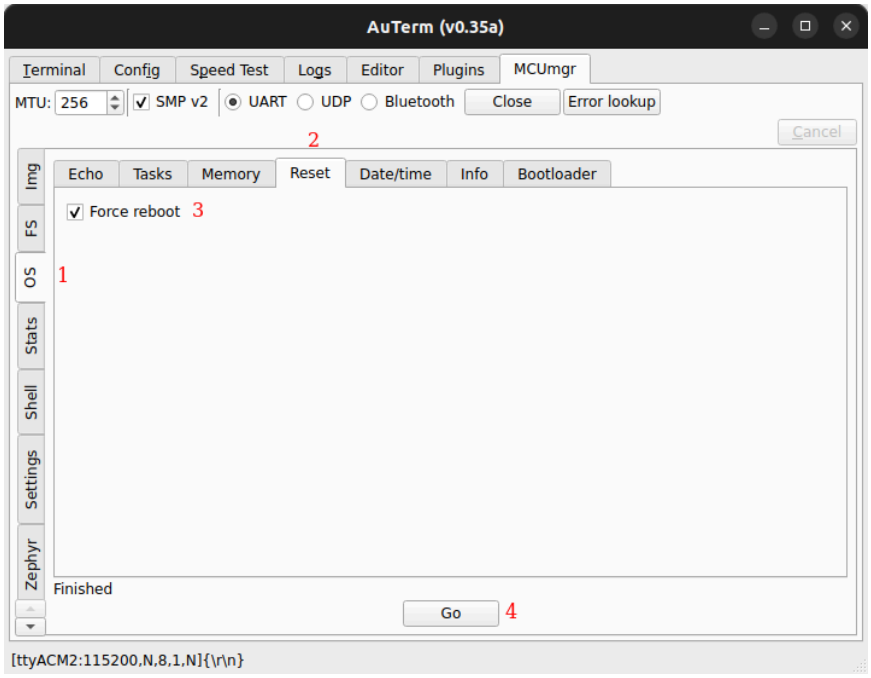
   b. Select “Test”
   c. Press “Go”
      i. Sometimes it will time out. If it does, just retry.



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

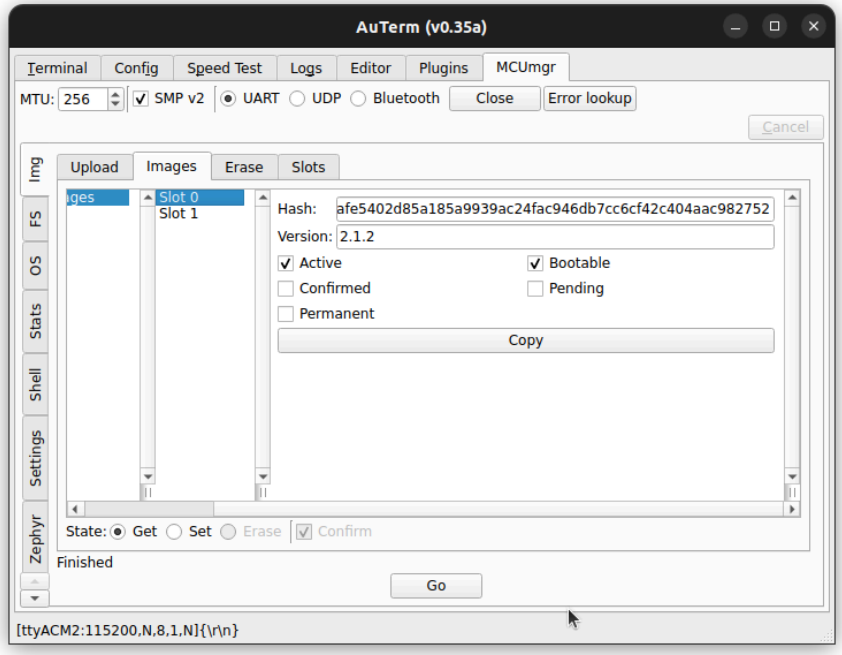


9. Switch to the OS tab
    a. Select the Reset tab
    b. Tick the “Force Reboot” box
    c. Press “Go”

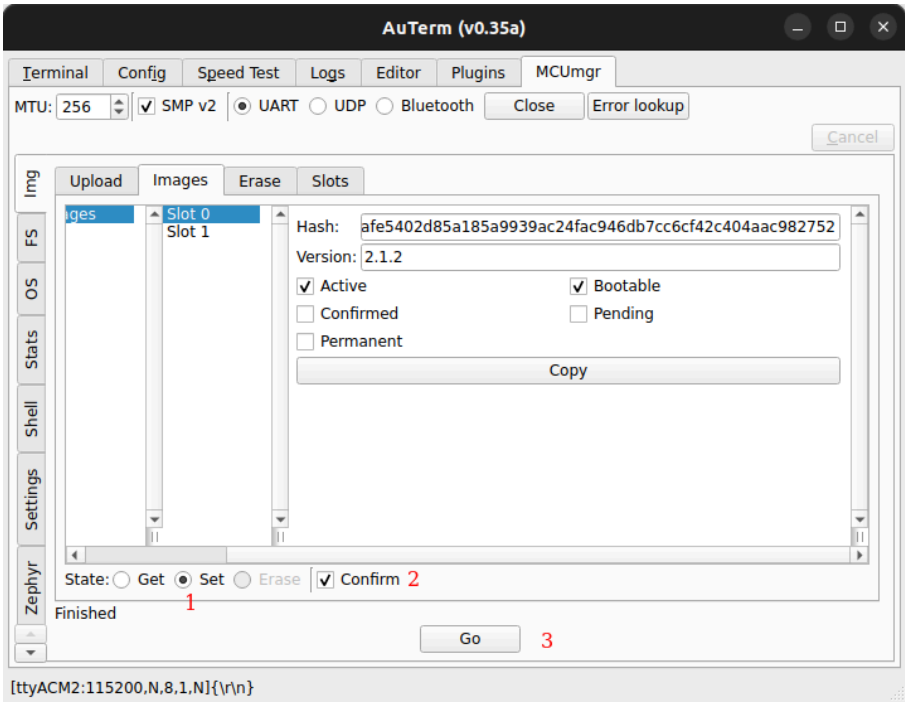


d. The other option is to press the “RESET” button on the hardware

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:
    a. Selecting the “Set” state
    b. Tick the “Confirm” box
    c. 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:

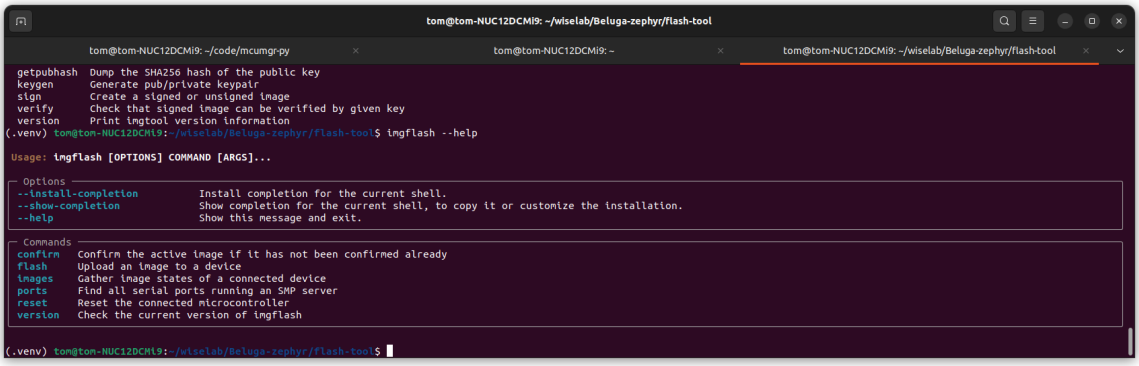
1. Python version >=3.9
2. Beluga repository is cloned

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

```
None
cd flash-tool
python3 -m venv .venv
source .venv/bin/activate
pip install .
```

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



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.

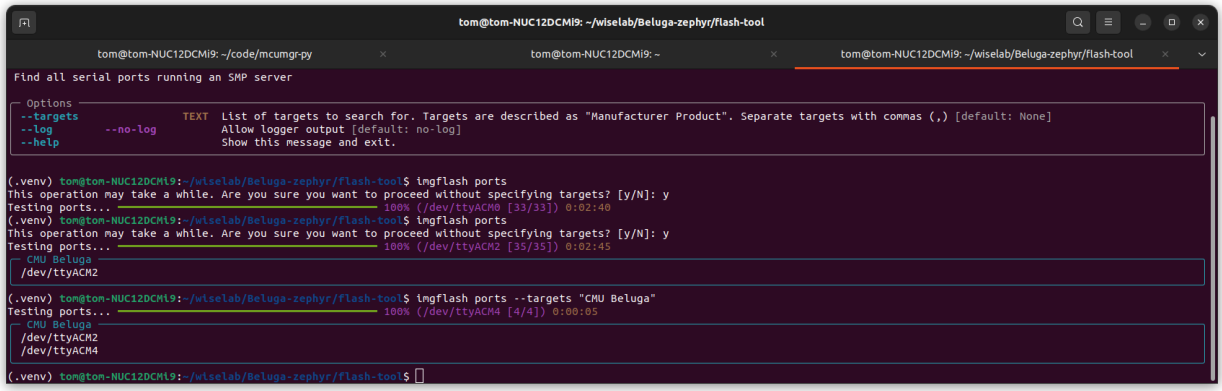
Ports

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



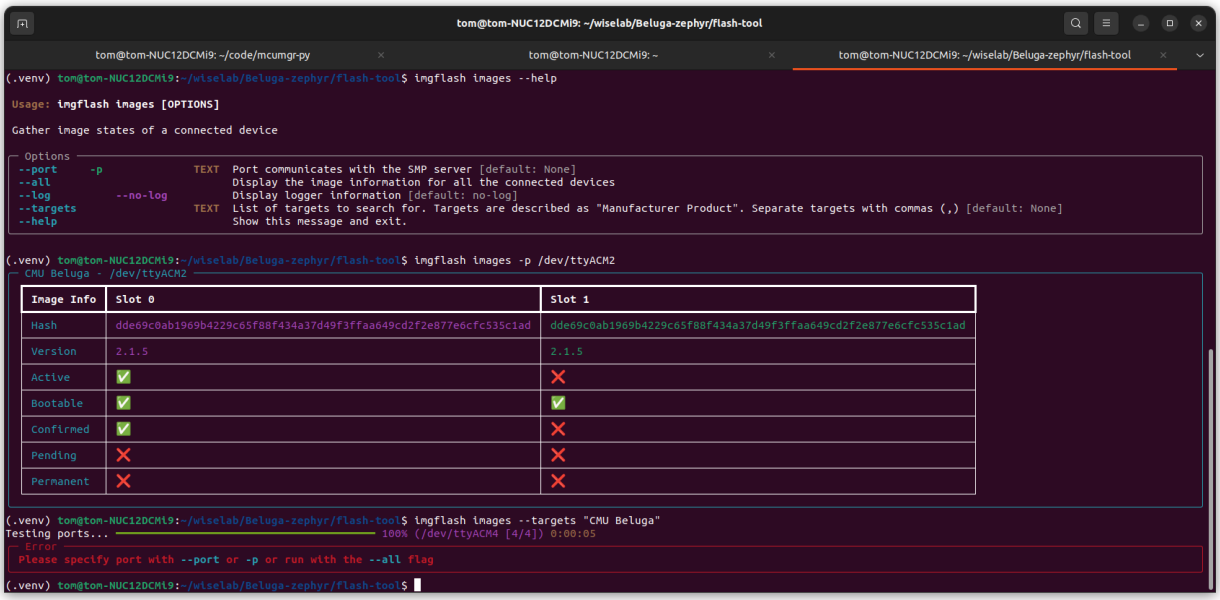
Images

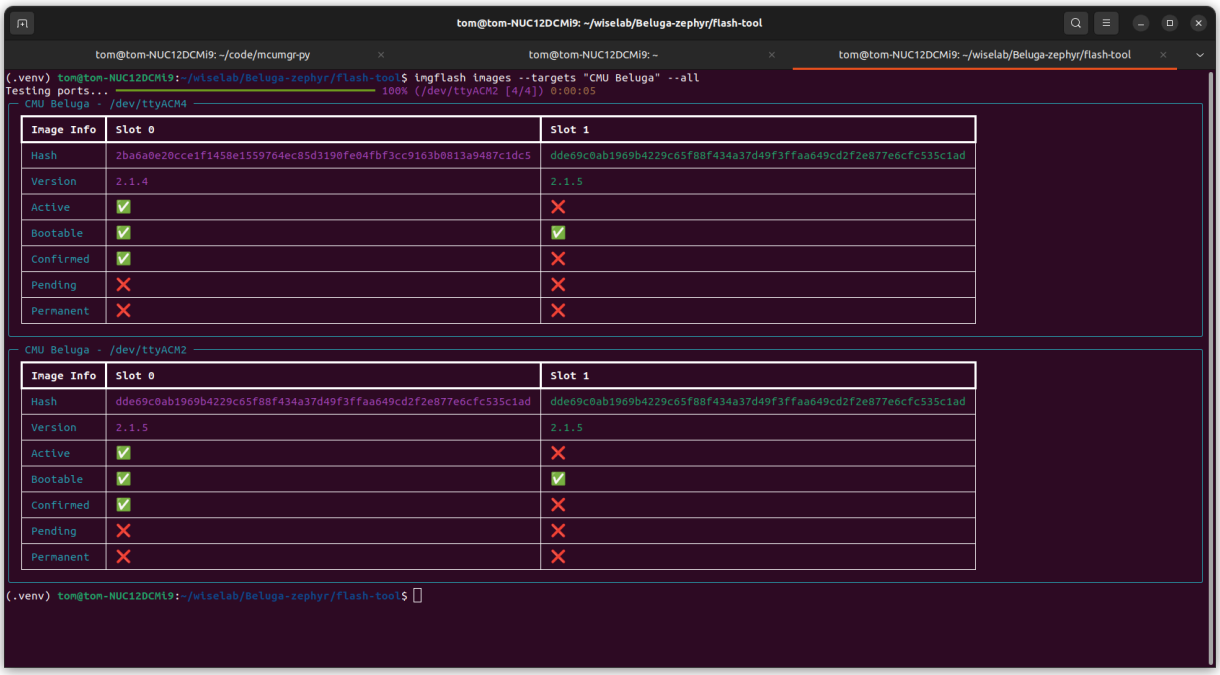
The images command displays the image information on the connected devices. This can be for a specific device, a specific device type, or for all connected devices.

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





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.

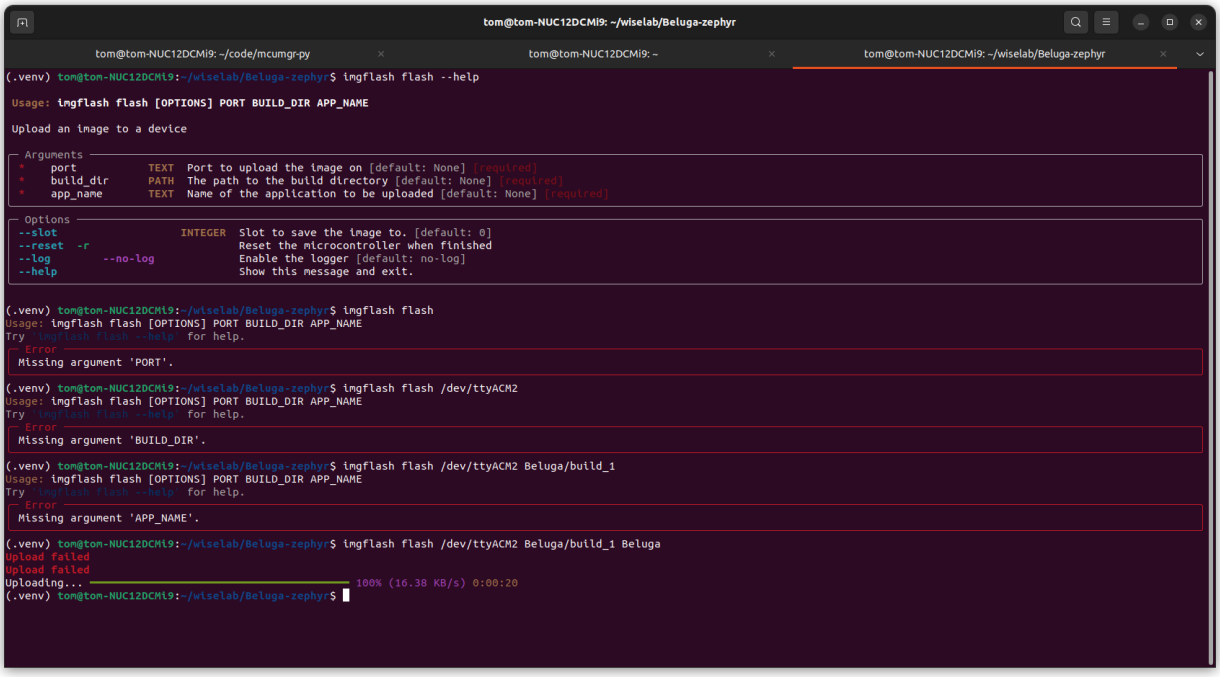
Arguments

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

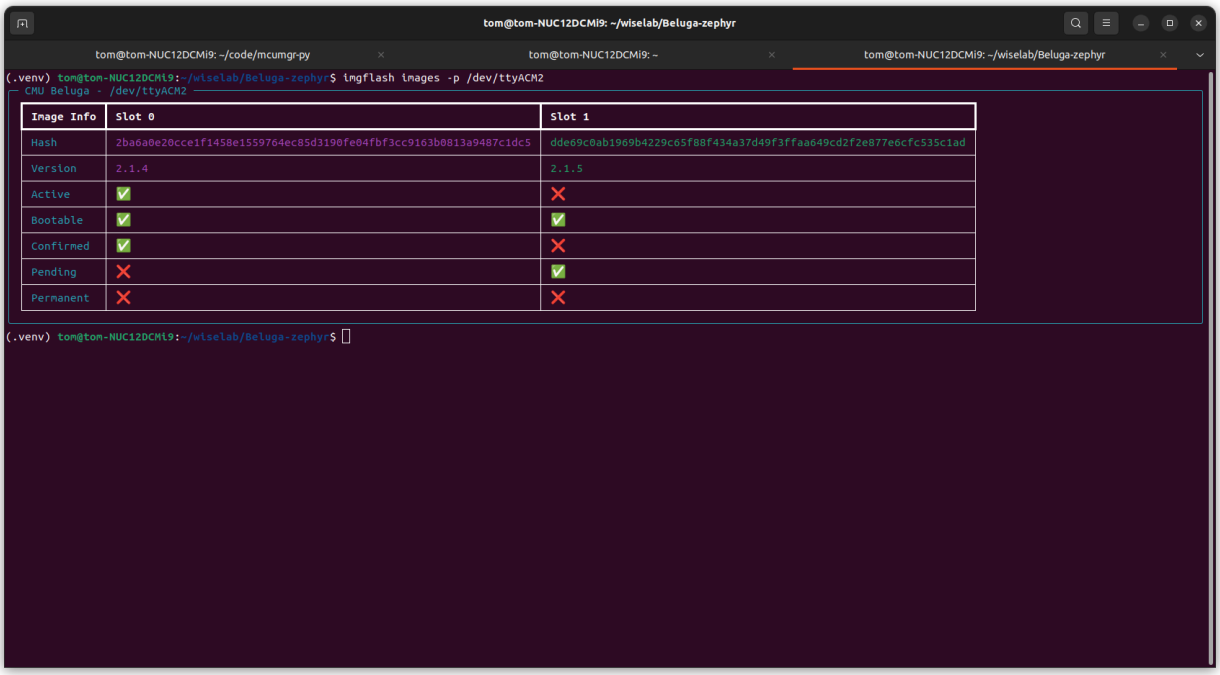
Options

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	

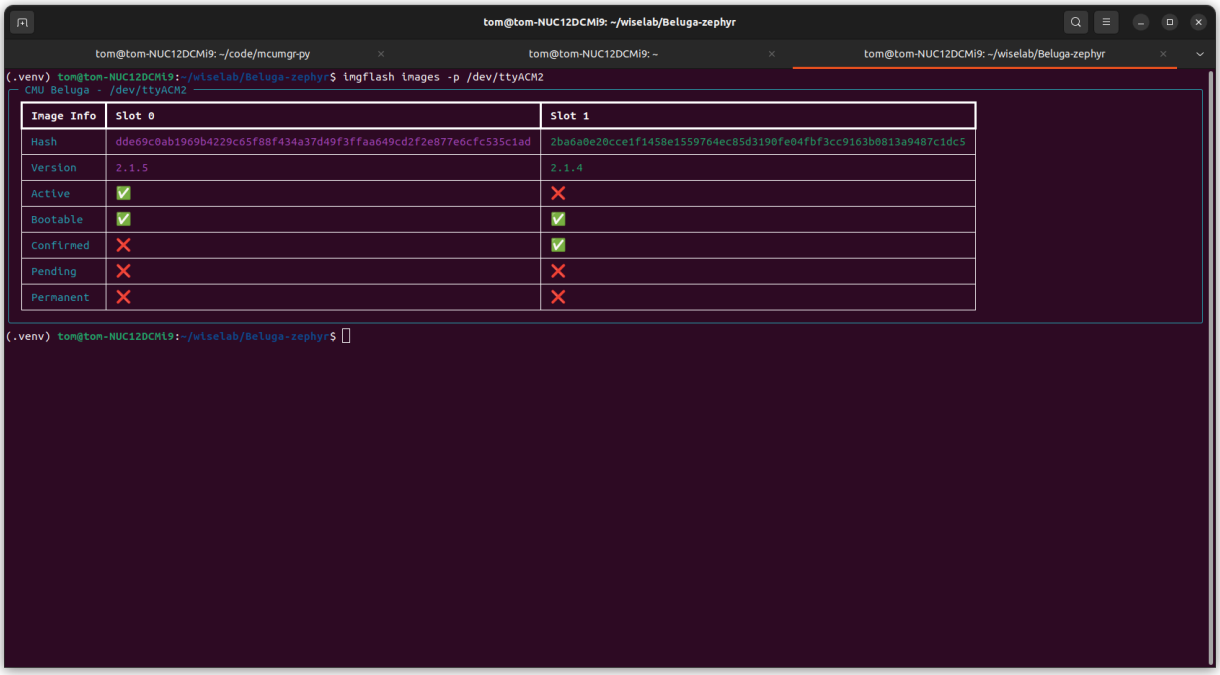
Example output



Images after uploading:



Images after uploading if reset option is specified



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.

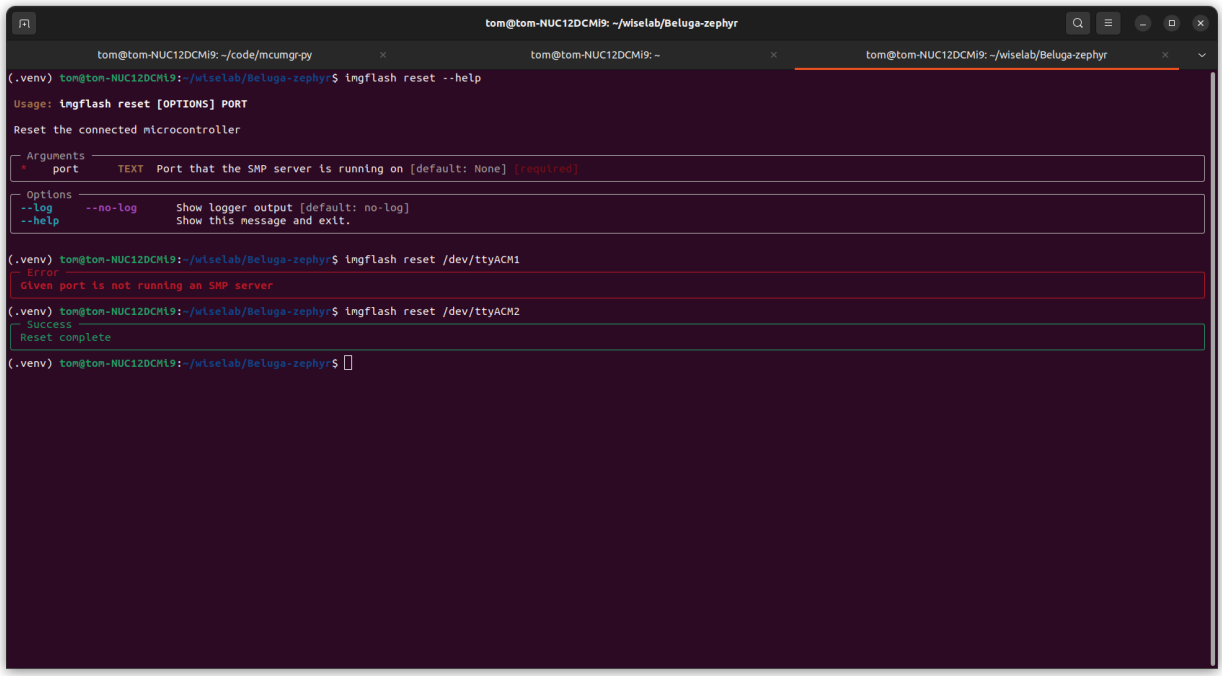
Arguments

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



Notes

Sometimes, reset will not work. I 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.

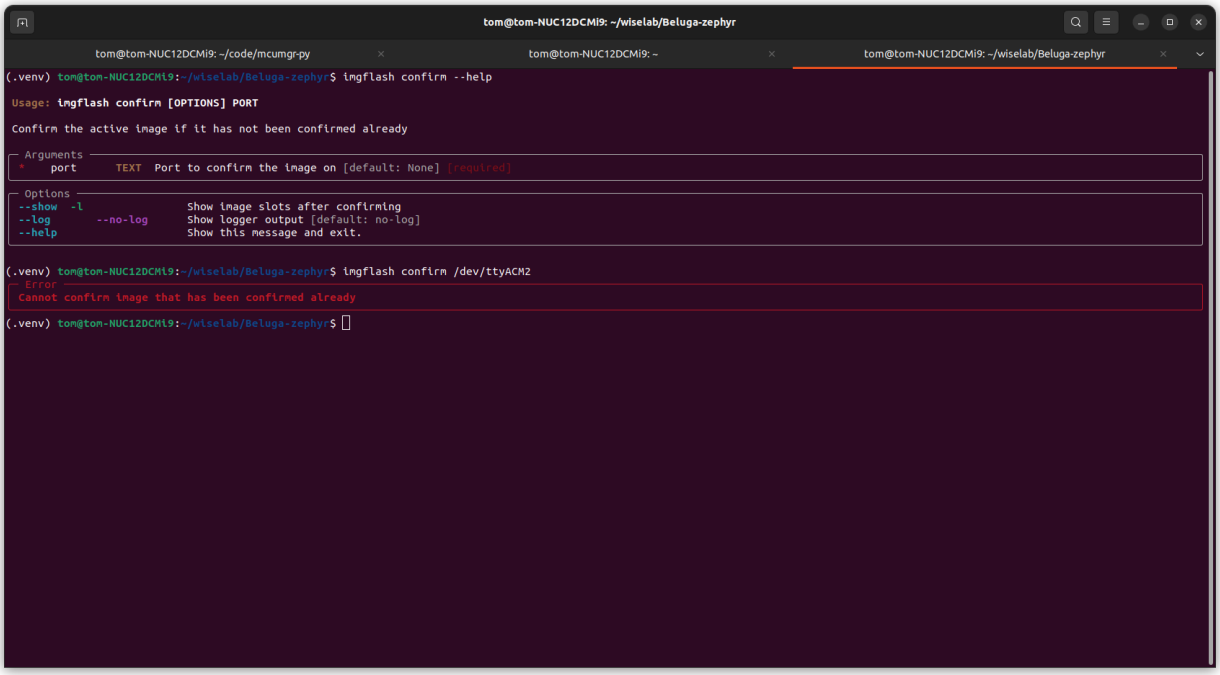
Arguments

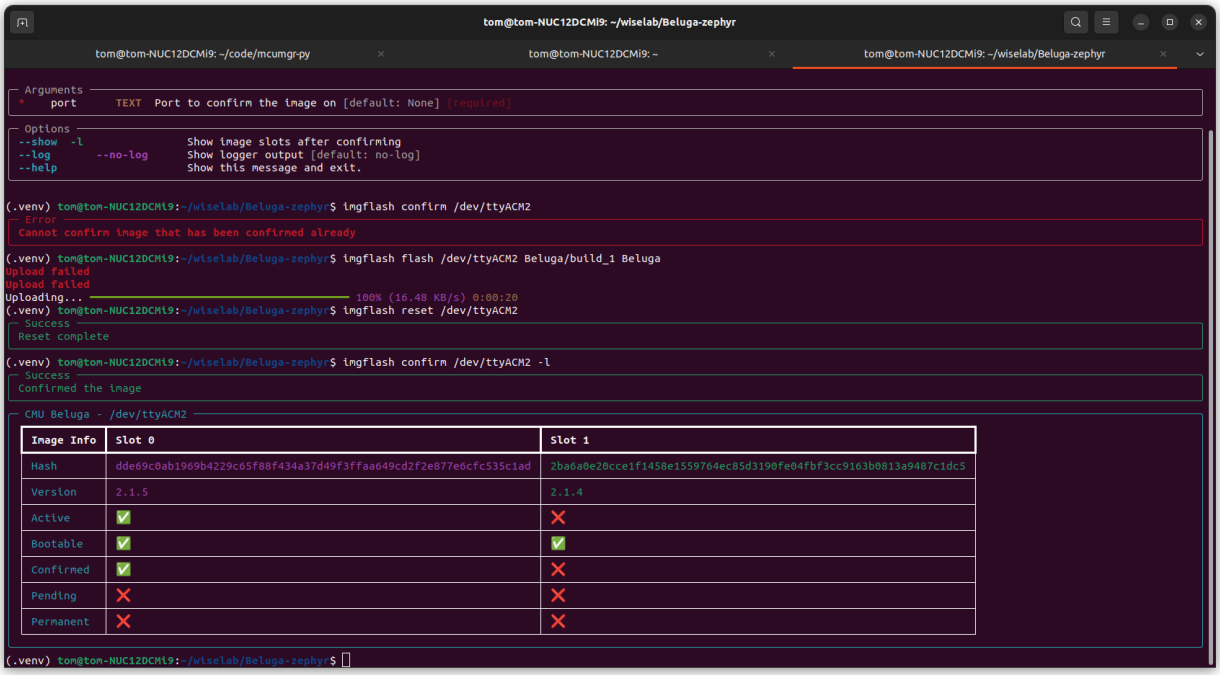
Argument	Type	Description	Position
port	TEXT	Port to confirm the image on	1

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





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.

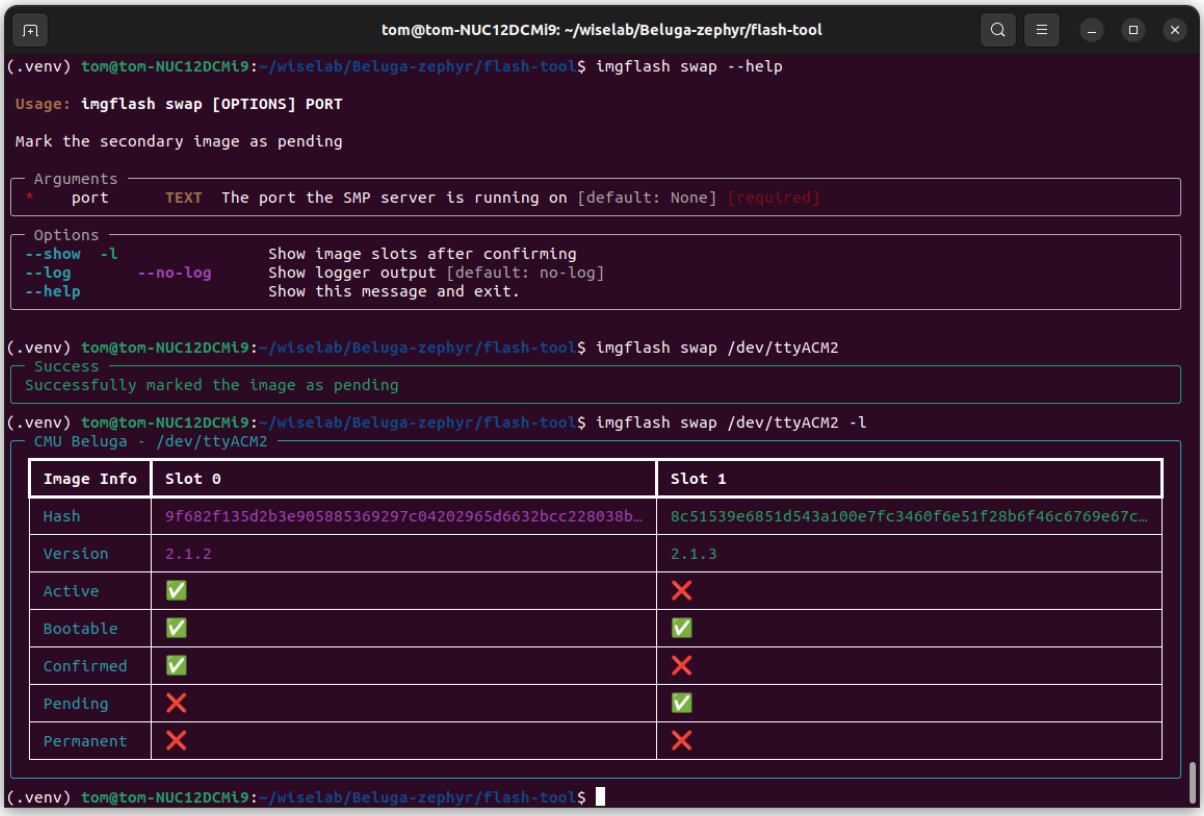
Arguments

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

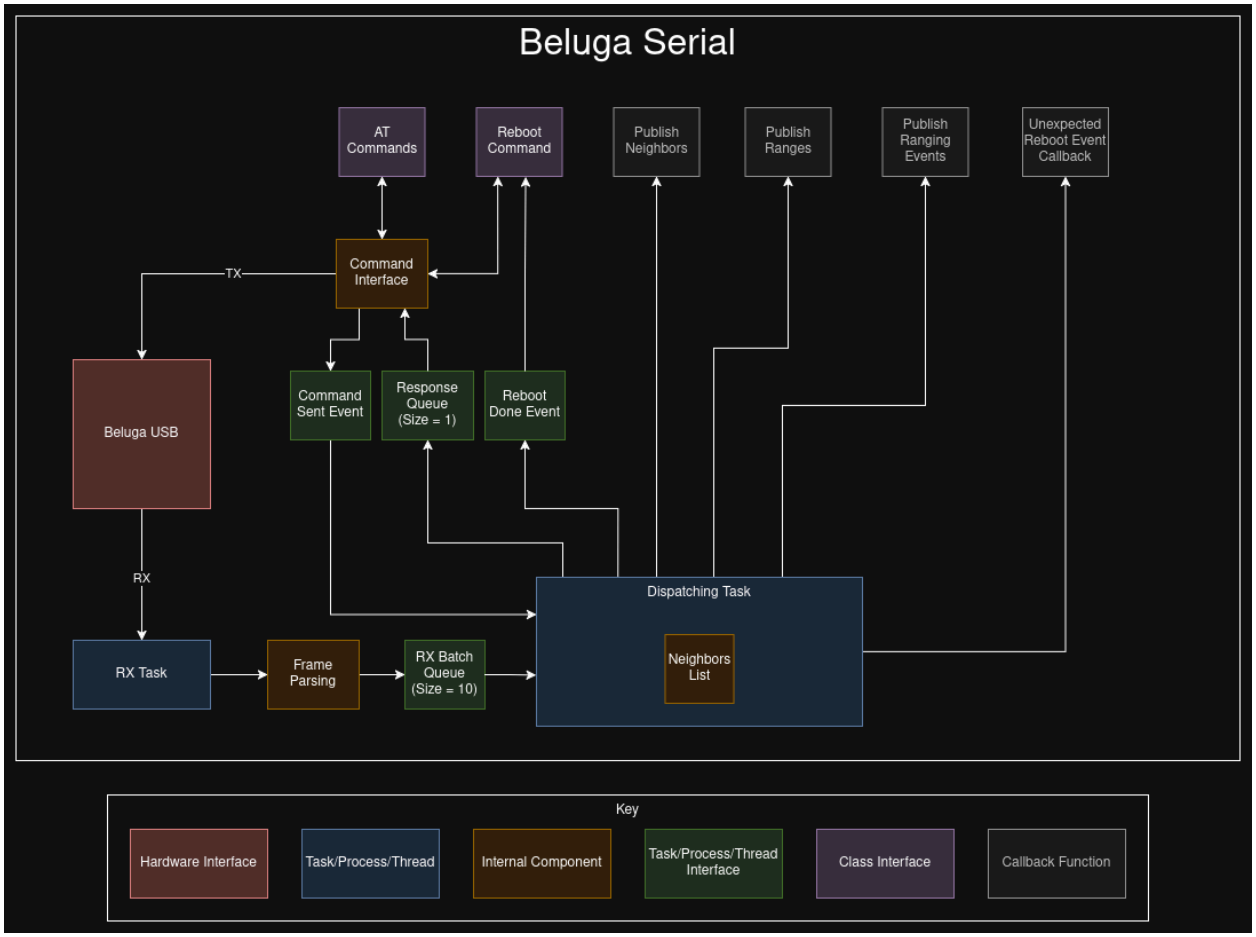


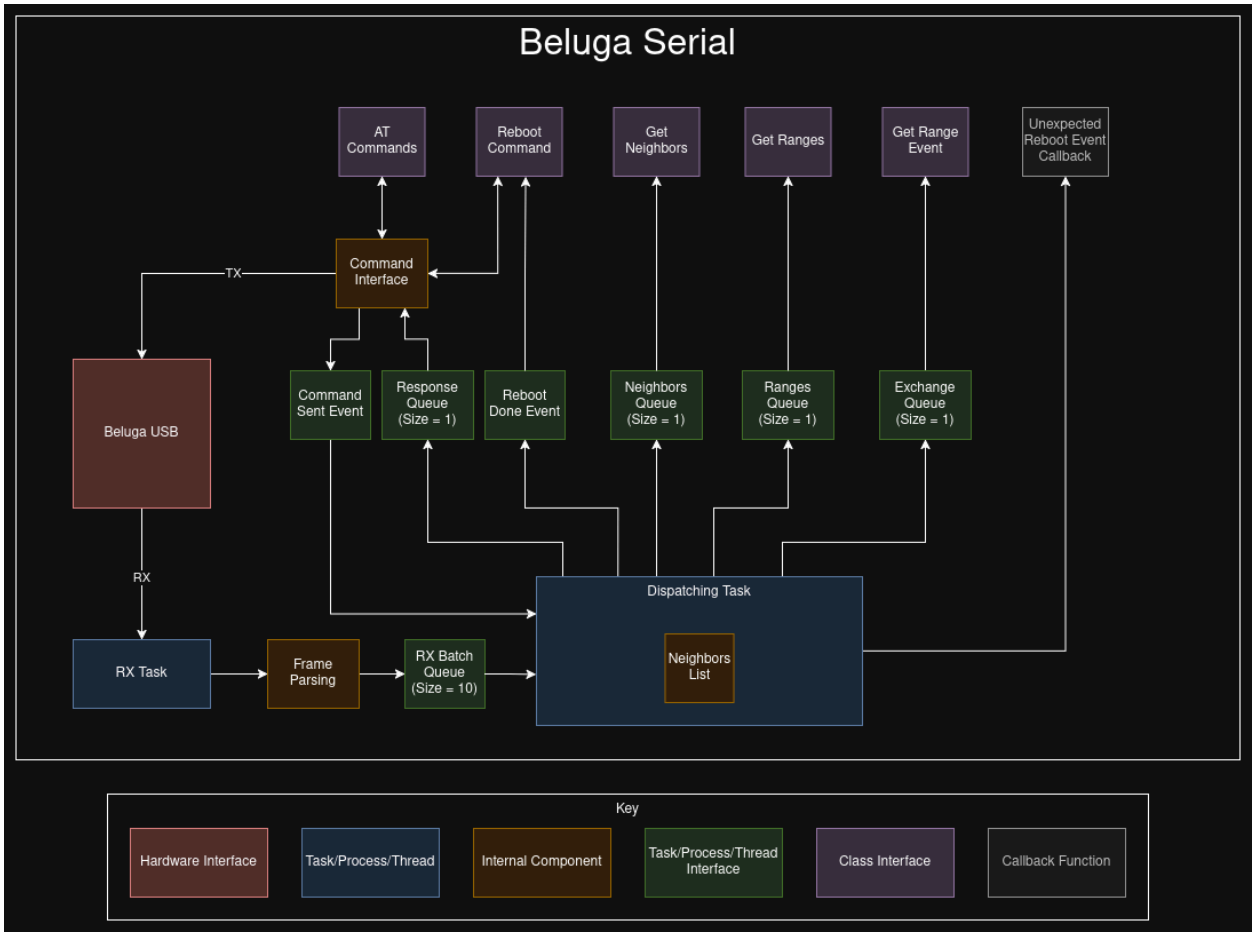
Serial Driver

In the Beluga repository, there are two serial driver APIs, one written for Python applications and the other written for C++ applications. Both APIs follow the same basic structure and are configurable.

Software Architecture

The serial drivers are written in a way to support both an event-driven architecture and a queue-based architecture (or a combination of both). The figures below show the software architectures:





Message Queues

BelugaSerial contains 3 message queues that can be read from: The neighbor updates queue, range updates queue, and the range exchange queue. Each of these queues have a depth of 1 meaning they only contain the latest reported data. When reading the data from the queue, the data is removed from the queue, and the queue remains empty until the processing task inserts new data into the queue.

Event Callbacks

BelugaSerial allows the use of callback functions as an alternative to queues. However, keep in mind that callback functions should have limited execution time, as they are called directly from the processing task.

Python API

The Python library implements a producer-consumer system where there are at least 3 processes running. The user-defined process for sending commands and consuming the data from Beluga, a serial read process for reading data from Beluga and decoding the data, and a

dispatching process for processing the data and dispatching it to the proper locations. This API was tested to work with Linux and MacOS. It did not work in Windows because Microsoft writes really, really bad software.

Installation

To install, BelugaSerial can be installed through pip with the following command:

```shell
pip install
git+https://github.com/WiseLabCMU/Beluga-2.0.git#subdirectory=serial-comms/pyth
on
```

Command Shell

BelugaSerial also comes with a command shell that can be ran through your Python environment. To invoke it, run the following command in your Python environment:

```shell
beluga-miniterm
```

C++ API

The C++ library implements 2 things:

1. A serial communication library designed to communicate with most serial devices on the Linux operating system.
2. The implementation of the communications with a Beluga node as described in the architecture diagram

The library implements a producer-consumer system where there are at least 3 threads running: The user-defined thread for sending commands to Beluga and consuming the data from Beluga, a serial read thread for reading data from Beluga and decoding the data, and a dispatching thread for processing the data from Beluga and dispatching the data to the proper locations.

Installing/Using

The C++ library comes packaged with a CMake file meaning that it can easily be integrated into a project that uses a CMake build system.

Downloading and using in CMake

The first option (and least convenient) is to download the beluga directory and drop it right into your project. After that, add the following to your CMakeLists.txt file:

```
None
add_subdirectory(beluga-serial)
target_link_libraries( executable-name PRIVATE beluga-serial )
```

Including in CMake project via FetchContent

The second option (preferable one) is to include the package via FetchContent:

```
None
include( FetchContent )
FetchContent_Declare(
        beluga-serial
        GIT_REPOSITORY https://github.com/WiseLabCMU/Beluga-2.0.git
        GIT_TAG master
        SOURCE_SUBDIR serial-comms/cpp/beluga-serial
)
FetchContent_MakeAvailable(beluga-serial)
target_link_libraries( target-name PRIVATE beluga-serial )
```

ROS Node

The Beluga repository also has a ROS node implementation in-tree. The ROS node contains 4 publishers and 2 services, each being outlined later in this section. In addition, the node can be configured via the node parameters and through build options. Additionally, a JSON file can be used to specify default node settings when starting.

Publishers

Below is a description of each of the publishers, including what messages they publish.

neighbor_list

This publisher publishes the neighbor list when a node is added or removed from the neighbor list. It publishes messages of type “BelugaNeighbors.”

```
None
# BelugaNeighbors.msg
BelugaNeighbor[] neighbors
```

```
None
# BelugaNeighbor.msg
uint16 id
float32 distance
int8 rssi
uint32 exchange
builtin_interfaces/Time timestamp
```

range_updates

This publisher publishes when there has been an update to a neighboring node’s range/distance. It publishes messages of type “BelugaRanges.”

```
None
# BelugaRanges.msg
BelugaRange[] ranges
```

```
None
# BelugaRange.msg
uint16 id
float32 range
uint32 exchange
builtin_interfaces/Time timestamp
```

range_exchanges

This publisher publishes when a ranging exchange successfully completes on the responder side. It publishes messages of type “BelugaExchange.”

```
None
# BelugaExchange.msg
uint16 id
uint32 exchange
builtin_interfaces/Time timestamp
```

unexpected_beluga_reboot

This publisher publishes whenever Beluga reboots unexpectedly. It publishes messages of type “BelugaUnexpectedReboot.”

```
None
# BelugaUnexpectedReboot.msg
builtin_interfaces/Time timestamp
```

Services

Below is a description of each of the services, including what messages the service messages look like.

at_command

This service directs the Beluga ROS node to run the specified AT command and return the response. It uses the service message of type “BelugaATCommand.”

```
None
# BelugaATCommand.srv
uint8 AT_COMMAND_STARTUWB=0
uint8 AT_COMMAND_STOPUWB=1
uint8 AT_COMMAND_STARTBLE=2
uint8 AT_COMMAND_STOPBLE=3
uint8 AT_COMMAND_ID=4
uint8 AT_COMMAND_BOOTMODE=5
uint8 AT_COMMAND_RATE=6
uint8 AT_COMMAND_CHANNEL=7
uint8 AT_COMMAND_RESET=8
uint8 AT_COMMAND_TIMEOUT=9
uint8 AT_COMMAND_TXPOWER=10
uint8 AT_COMMAND_STREAMMODE=11
uint8 AT_COMMAND_TWRMODE=12
uint8 AT_COMMAND_LEDMODE=13
uint8 AT_COMMAND_REBOOT=14
uint8 AT_COMMAND_PWRAMP=15
uint8 AT_COMMAND_ANTENNA=16
uint8 AT_COMMAND_TIME=17
uint8 AT_COMMAND_DEEPSLEEP=18
uint8 AT_COMMAND_DATARATE=19
uint8 AT_COMMAND_PREAMBLE=20
uint8 AT_COMMAND_PULSERATE=21
uint8 AT_COMMAND_PHR=22
uint8 AT_COMMAND_PAC=23
uint8 AT_COMMAND_SFD=24
uint8 AT_COMMAND_PANID=25
uint8 AT_COMMAND_EVICT=26
uint8 AT_COMMAND_VERBOSE=27
uint8 AT_COMMAND_STATUS=28
uint8 AT_COMMAND_VERSION=29

string arg
uint8 at_command
---
string response
```

power_control

This service directs the Beluga ROS node to set the power level of both the BLE and the UWB. It uses the service message of type “BelugaPowerControl.”

```
None
# BelugaPowerControl.srv
# BLE amplifier states
uint8 POWER_CONTROL_BLE_AMP_OFF=0
uint8 POWER_CONTROL_BLE_AMP_LOW=2
uint8 POWER_CONTROL_BLE_AMP_HIGH=4

# UWB external amplifier states
bool POWER_CONTROL_UWB_AMP_OFF=false
bool POWER_CONTROL_UWB_AMP_ON=true

# Indices for the DW1000 power control stages
uint8 POWER_CONTROL_BOOSTNORM=0
uint8 POWER_CONTROL_BOOSTP500=1
uint8 POWER_CONTROL_BOOSTP250=2
uint8 POWER_CONTROL_BOOSTP125=3

uint8 ble_amp_state
bool uwb_amp_state
uint8[4] coarse_gain
uint8[4] fine_gain
---
# Indices for the AT command responses
uint8 POWER_CONTROL_RESP_EXTERNAL_AMP=0
uint8 POWER_CONTROL_RESP_BOOSTNORM=1
uint8 POWER_CONTROL_RESP_BOOSTP500=2
uint8 POWER_CONTROL_RESP_BOOSTP250=3
uint8 POWER_CONTROL_RESP_BOOSTP125=4

string[5] responses
string set_power
```

Build Options

One of the ways to configure the ROS node is through build options. This is useful for things like turning on/off loggers and determining which type of publisher to use. In total, there are 8 build options specific to the Beluga ROS node. All these options are ON/OFF.

Option	Description	Default
TIMED_PUBS	Enable all timed publishers	OFF
TIMED_NEIGHBOR_PUB	Enable timed publisher for neighbor list updates	OFF

Option	Description	Default
TIMED_RANGES_PUB	Enable timed publisher for range updates	OFF
TIMED_EXCHANGE_PUB	Enable timed publisher for ranging exchanges	OFF
LOG_PUBS	Log all messages being published	OFF
LOG_NEIGHBORS	Log neighbor list updates	OFF
LOG_RANGES	Log range updates	OFF
LOG_EXCHANGES	Log ranging exchanges	OFF

Runtime Parameters

Another way to configure the ROS node for Beluga is through the runtime parameters. This is useful for adjusting the behavior of the node to the user's application.

Parameter Name	Description	Default	Type
neighbors_name	Neighbor list updates publisher topic name	neighbor_list	string
ranges_name	Range updates publisher topic name	range_updates	string
exchange_name	Range exchanges publisher topic name	range_exchanges	string
history_depth	Determines the quality of service for the publishers	10	int
service_topic	AT command service topic name	at_command	string
port	The serial port to connect to		string
config	Path to the JSON file for the Beluga Node settings (Not ROS node settings)		string
reboot_topic	Unexpected reboot	unexpected_beluga_reboot	string

Parameter Name	Description	Default	Type
	event publisher topic name		
power_control_topic	Power control service topic name	power_control	string
neighbor_period	Timer period (seconds) of the neighbor list publisher (if the timed publisher is enabled)	30	int
ranging_period	Timer period (seconds) of the range updates publisher (if the timed publisher is enabled)	30	int
events_period	Timer period (milliseconds) of the range exchanges publisher (if the timed publisher is enabled)	100	int

Config File for Customized Beluga Settings

The ROS node has the ability to take in a JSON file for Beluga firmware configurations. The path of this file is given through the “config” parameter when launching the ROS node. Below is a starter configuration file.

```JSON
{
        "id": 1,
        "bootmode": 2,
        "rate": 100,
        "channel": 5,
        "timeout": 9000,
        "txpower": 235405858,
        "streammode": 1,
        "twrmode": 1,
        "ledmode": 0,
        "pwramp": 1,
        "antenna": 1,
        "phr": 0,
        "datarate": 0,
        "pulserate": 0,
```

```
        "preamble": 128,
        "pac": 0,
        "sfd": 0,
        "panid": 41760
}
```