DVXplorer-S

Release master

iniVation AG

Jun 13, 2023

CONTENTS

Date: 2023-06-11

Note: This documentation is a work-in-progress. Please check back frequently for updates and provide us your feedback!

Please read carefully through the whole documentation to use the DVXplorer S to its full potential and be aware of its limitations.

The DVXplorer S (S standing for System) is an embedded camera, featuring a dual-sensor solution comprised of our DVS event-based sensor and a state-of-the-art OnSemi AR0234 global-shutter industrial color image sensor. Processing power is provided by an Nvidia Jetson Nano SOM[1], with a quad-core ARM Cortex-A57 processor, running at up to 1.43 GHz, 4 GB of DDR4 RAM and an Nvidia Maxwell GPU with 128 CUDA cores.

[1] https://developer.nvidia.com/embedded/jetson-nano

CHAPTER

ONE

FEATURES

1.1 Hardware

- Nvidia Jetson Nano SOM (4x ARM Cortex-A57 @ 1.43 GHz, 4GB DDR4-RAM, Maxwell GPU @ 128 cores)
- iniVation DVS event sensor, 640x480 resolution (VGA)
- OnSemi AR0234 global-shutter color frame sensor, 1920x1200 resolution (WUXGA)
- Bosch BMI160 IMU sensor, 6 axis gyroscope + accelerometer
- 16 GB of eMMC storage total, 6 GB available for user data and recordings
- Gigabit Ethernet connector (RJ45), supports Power-Over-Ethernet (POE 802.3af)
- USB 3.0 Type C connector, with locking screws
- Mini-HDMI 1.4 connector
- 10x 2.54mm pins female connector for SPI, GPIO and PWM
- Size: H 32mm x W 80mm x D 92mm
- Weight: 220g, without lenses
- Lenses: 2x S-Mount (M12), 2x 6mm focal length provided with camera

1.2 Software

- Yocto Linux-based distribution
- Linux4Tegra kernel 4.19
- DV-SDK remote runtime
- Yocto-based SDK to cross-compile custom software
- Robust A/B update procedure, with separate partition for user data

CHAPTER

TWO

INTENDED USAGE

This device is meant to be used as an embedded platform for processing sensor data, running preferably attached to an Ethernet network.

It runs a custom Linux distribution, based upon the Yocto project[2], that directly integrates our DV-SDK software and automatically runs its runtime on boot. This allows users to connect using the GUI, setup modules and their connectivity on the embedded camera itself and run them there. Custom modules can also be uploaded and used.

Processing *performance is limited*, and will be at its worst if you try to use the camera like any of our other USB camera products, by just visualizing data on the DV GUI. Processing and data reduction should happen as much as possible on the device itself.

Data can be sent off the camera via either USB or Ethernet, but their *limited bandwidth* has to be taken into consideration. We recommend Ethernet for more robust deployments, due to its easier cabling situation and guaranteed higher power delivery capabilities.

[2] https://www.yoctoproject.org/

CHAPTER

THREE

PERFORMANCE CONSIDERATIONS

3.1 Processing

The Nvidia Jetson Nano offers relatively limited CPU and GPU performance. Custom code will need to be carefully optimized to run in real-time.

The amount of data that can be processed is limited and below what the sensors can provide at their peak performance.

The main bottlenecks for sensor data processing are:

- decoding of events from the camera into AEDAT4 format, caps at around 30 Mevents/s
- debayering (color interpolation) of image frames, caps at around 50 FPS

Todo: Later software releases will try to continuously improve the performance of algorithms and modules included by default in the DV-SDK software.

3.2 Networking

Getting data off the camera is limited by the respective networking connection's bandwidth:

- Ethernet: max. 1 Gbit/s
- USB 3.0 (network mode): max. 1.3 Gbit/s

A full resolution color frame at 1920x1200 takes 6.6 MB of data uncompressed, so you could transmit at most 18 frames per second, using all the bandwidth only for frames, over the Ethernet connection. Or, if sending only events, the USB connection would max out at around 9 Mevents/s. Sending all the data off the camera is *not* its intended mode of usage and will result in mediocre performance. It is imperative that as much processing and data reduction happens on the device. Best-case, only final features such as "person detected at position (X,Y)" or "42% activity in scene" would get sent off the device during normal operation.

The visualization function in the GUI also works by sending data out over the network and is constrained by these same limitations.

Compression can be enabled in the network output modules to improve the situation, though the trade-off is increased CPU usage. By default compression is disabled (**NONE**) for this reason, if you want to enable it, we recommend **LZ4**, as it is a good trade-off between compression and CPU usage.

3.3 Power

The Jetson Nano can run in two power states:

- 5W mode, plus system, total ~7.5W
- 10W MAXN mode, plus system, total ~12.5W

Higher power modes offer better CPU and GPU performance.

Ethernet POE 802.3af can provide up to 15W and thus run both modes without issues.

USB 3 can by default only provide up to 4.5W, so a normal USB 3.0 port will never be able to safely run this device. In truth, many USB ports and especially powered USB hubs can provide more power, often up to 15W (3A). In the case of USB Type A ports used with an A-to-C cable, this cannot be detected by the device. For USB Type C ports and cables, it is possible to detect higher power modes of 7.5W (1.5A) and 15W (3A) using the CC lines[3], if the host reports these properly.

For this reason, to try and guarantee power stability, our device always boots in the lowest power mode, and currently stays there.

Todo: Later software releases will detect and report as much information as possible concerning power available, mode selected and current power consumption. We'll also implement automatically switching to higher performance modes based upon that information, so as to always offer optimal performance. You can use the *nvpmodel* tool over SSH for now if you want to force higher performance modes manually.

3.4 Thermal

Warning: Please always be careful of the temperature when handling the device!

The device is passively cooled through the visible aluminium fins on its top. For best cooling performance, make sure these point up to allow for convective cooling to happen optimally. The device can become quite hot under continuous full load in the highest power settings, we have measured temperatures of up to 60°C (140° F) on the outer metal casing. Under normal load the temperature stays around 40° C (104° F).

Todo: Later software releases will report in detail on measured temperature. You can use the *tegra-stats* tool over SSH for now if you want to get this information displayed.

---

[3] https://community.silabs.com/s/article/what-s-the-role-of-cc-pin-in-type-c-solution?language=en_US

CHAPTER

FOUR

TIME SYNCHRONIZATION

The timestamps of event, frame and IMU data in DV-SDK are defined as microseconds since the Unix Time Epoch[4]. Since the device runs Linux, it must be able to accurately set its real-time clock and keep it synchronized. This is a well-known and well-solved problem using network time synchronization techniques, such as the Network Time Protocol (NTP) or the Precision Time Protocol (PTP). It is even possible to use external hardware such as GPS and Atomic Clocks for certain use cases. For network time synchronization to work, the device must be able to reach an external network, usually the internet, and talk to pool.ntp.org[5]. If you connect over Ethernet with DHCP, we assume the resulting connection can access the internet and the global NTP servers. If you connect over USB, you'll have to enable network forwarding for the camera first, *see below* for instructions.

Todo: Support for setting NTP server addresses manually and getting them from DHCP will come in a later software release. Later software releases will also show the time synchronization status in an easy to digest manner in DV.

Note: Support for advanced time synchronization via PTP and GPS is not supported on the current hardware.

The network chip of the Nvidia Jetson Nano does not support PTP hardware timestamping, nor is it possible to attach most common GPS receivers since the UART pins are not exposed.

4.1 USB Network Forwarding

Network forwarding enables the device to access the wider network and internet when connected over USB. This is required for time synchronization features, and in the future, for automatic system image updates.

4.1.1 Linux

Execute the following shell-script as root to enable network forwarding to the camera:

```
sudo ./usb-internet.sh
```

Script content (*usb-internet.sh*):

```
#!/bin/sh

echo 1 > /proc/sys/net/ipv4/ip_forward
```

(continues on next page)

[4] https://en.wikipedia.org/wiki/Unix_time
[5] https://www.ntppool.org/en/

(continued from previous page)

```
iptables -A FORWARD -m conntrack --ctstate ESTABLISHED,RELATED -j ACCEPT
iptables -A FORWARD -j ACCEPT -s 192.168.55.0/24
iptables -A POSTROUTING -t nat -j MASQUERADE -s 192.168.55.0/24
```

4.1.2 MacOS

Todo: MacOS USB Network Forwarding instructions not yet available.

4.1.3 Windows

Todo: Windows USB Network Forwarding instructions not yet available.

CHAPTER

FIVE

DV-RUNTIME ACCESS

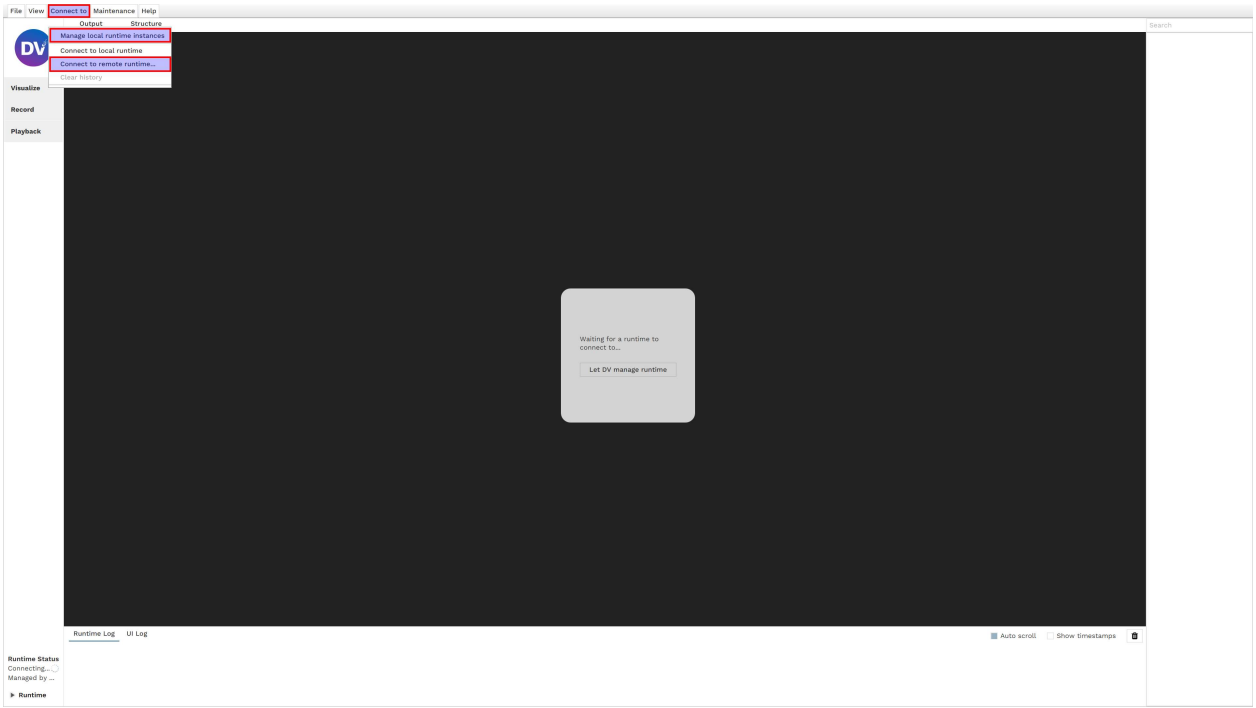
The embedded camera is running an instance of our DV-SDK software's runtime. It allows to connect to the camera from the GUI remotely, visualize data, add/remove and connect modules to process data.

5.1 Via USB

Open the GUI, in the top menu select "Connect to", in the resulting drop-down menu disable "Manage local runtime instances", the greyed out "Connect to remote runtime…" will then become available. Select it and in the ensuing pop-up enter the following parameters:

- IP Address: 192.168.55.1
- Port: 4040

Finally, click on Connect.



Note: When connecting via USB, the IP address is always this same one. As such, only *one* camera can be connected

via USB at the same time to the same host system.

5.2 Via Ethernet

The same procedure as for USB applies to connect to the remote runtime, with the difference the IP address will be different and depend on your network configuration. Usually DHCP will assign an unique IP to the camera automatically. The presence of the camera's dv-runtime service on the network is advertised through the mDNS auto-discovery service, you can use tools such as *avahi-discover* to see all the cameras on your network. Alternatively ask your network administrator to check and communicate the address to you.

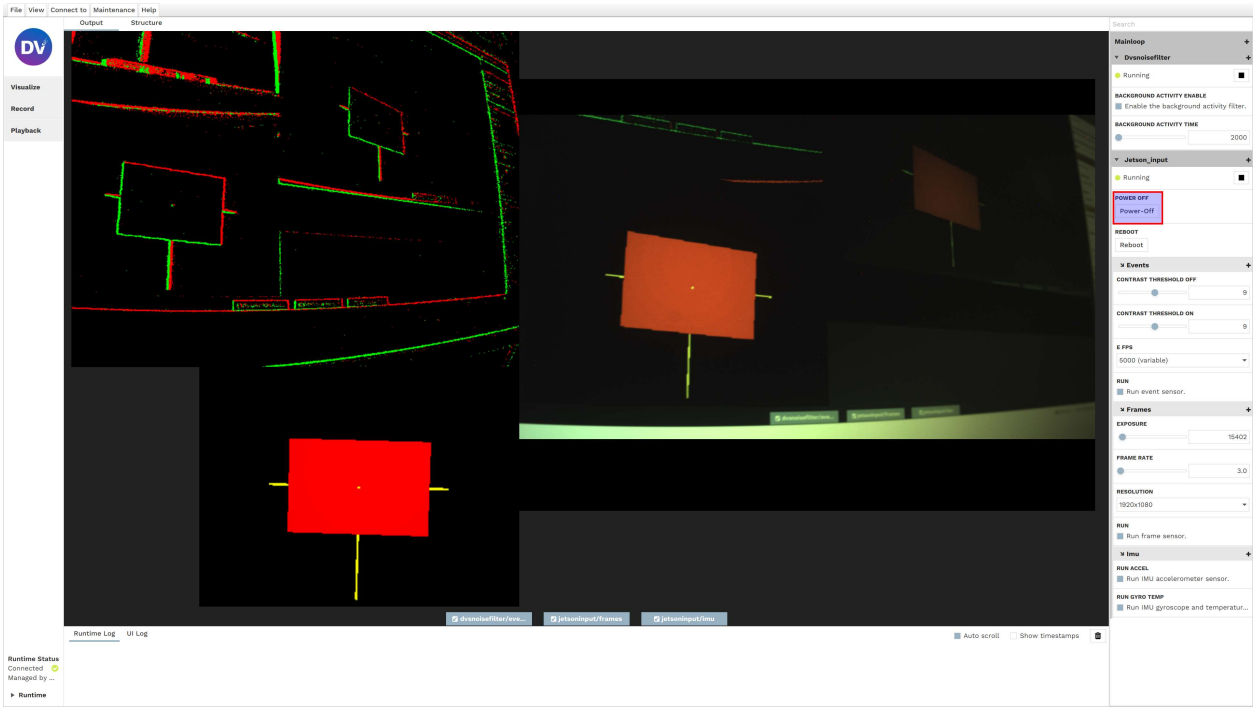
Todo: Later software releases will allow more flexible network configurations. For now, if you have need for a very specific network configuration, SSH into the device and edit the systemd-networkd configuration file at */data/config/20-wired.conf* manually.

CHAPTER

SIX

DV-RUNTIME USAGE

The default configuration for the DV-SDK runtime always contains a *dv_jetson_input* module, which is the interface to the system's three main sensors.



All sensor settings can be reached from the right side-menu, as usual click on the + sign to access and search the full list of options. The most important options are already displayed in the right side-menu by default.

6.1 Power-Off / Reboot

> **Warning:** Please power off the device using the appropriate *Power-Off* button in DV!

Of particular note are the *Power-Off* and *Reboot* buttons. As the names suggest, they turn off or reboot the device. It is very important to properly power off the device through clicking this button or, alternatively, sending the `poweroff` command via SSH. Improperly shutting the device down by disconnecting its power supply (USB or Ethernet cable) will result in the loss of the current DV runtime configuration, as it is only saved at shut-down. Further, if any recording was

happening at the time (*dv_output_file* module), it will be truncated and improperly terminated, which can lead to loss of data and performance.

Todo: Later software releases will offer a button to manually save the current configuration to the device at any point in time. It will still be recommended to properly shut it down via the appropriate buttons.

You can also always use the DV GUI Projects feature to save and reload any configuration states as XML files.

6.2 Frame Sensor Control

The most important controls for the frame sensor concern the frame-rate, the exposure and the resolution. The resolution defines the number of pixels to read out, which takes a certain amount of time *T_READOUT* for a specific resolution. This defines the maximum theoretical frame-rate. The frame-rate control allows the artifical lengthening of *T_READOUT* to slow down readout to a preferred rate by the user, let's call that time *T_READOUT_USER*. The *MAXIMUM(T_READOUT_USER, EXPOSURE)* defines the effective time needed to read a frame, and thus the real frame-rate you'll get from the device.

Todo: Later software releases will introduce automatic exposure control support, as well as more fine-grained controls for gain and other advanced features.

6.3 Calibration

Currently no special calibration support is provided. The two image sensors work independently without any calibration. You'll have to perform standard lens calibration, followed by stereo calibration[6], if you want to put the two sensors in precise relation with each other.

[6] https://inivation.gitlab.io/dv/dv-docs/docs/tutorial-calibration/

CHAPTER

SEVEN

SSH ACCESS

Warning: Please set a secure root password for SSH access as soon as possible!

It is possible to access the device via SSH as root, currently no password is set by default. We urge users to set a secure password as soon as possible. SFTP is also enabled to upload custom DV modules and update the system image.

- IP Address: see *DV-runtime access*
- Port: 22
- User: root
- Password: *none*

Todo: Future software releases will change how authentication works, disabling password-less root access and requiring users to log-in using SSH keys. A method to upload/import SSH keys over DV will be provided.

CHAPTER

EIGHT

WRITE AND COMPILE YOUR OWN MODULES

You can write your own DV-SDK C/C++ modules to process data following your personal requirements. You will need to compile them specifically to run on the DVXplorer S. To do this, you'll need to install the SDK and configure your development environment to use it for building your module. We currently provide detailed instructions for the JetBrains CLion IDE[7].

Todo: Instructions for setting up the Yocto SDK with the Qt-Creator and Visual Studio Code IDEs will follow.

Todo: Support for installing the SDK on Windows and MacOS will follow. MacOS should work the same as Linux but has not yet been tested.

8.1 Install Yocto SDK on Linux

Download the latest Yocto SDK installer for Linux (.sh file) from release.inivation.com[8].

Latest official release[9] (2023-06-07_13-18-51 git revision ef6d1e5c)

Install the SDK by executing the installer:

```
chmod 0755 ./inivation-image-full-jetson-nano-inivation-v2-x86_64-glibc-toolchain-
↪2023-06-07_13-18-51-master-ef6d1e5c.sh
./inivation-image-full-jetson-nano-inivation-v2-x86_64-glibc-toolchain-2023-06-07_13-
↪18-51-master-ef6d1e5c.sh
```

Install it to a user-accessible, easy to remember folder like */home/youruser/inivation-sdk/*.

Note: If you're not running the latest system image, you'll have to download and install the correct version of the SDK for that specific version of the system image. To find out the system image version you're running, SSH into the device and take a look at the */etc/build* file.

Todo: Later software versions will show build version information in DV.

[7] https://www.jetbrains.com/clion/

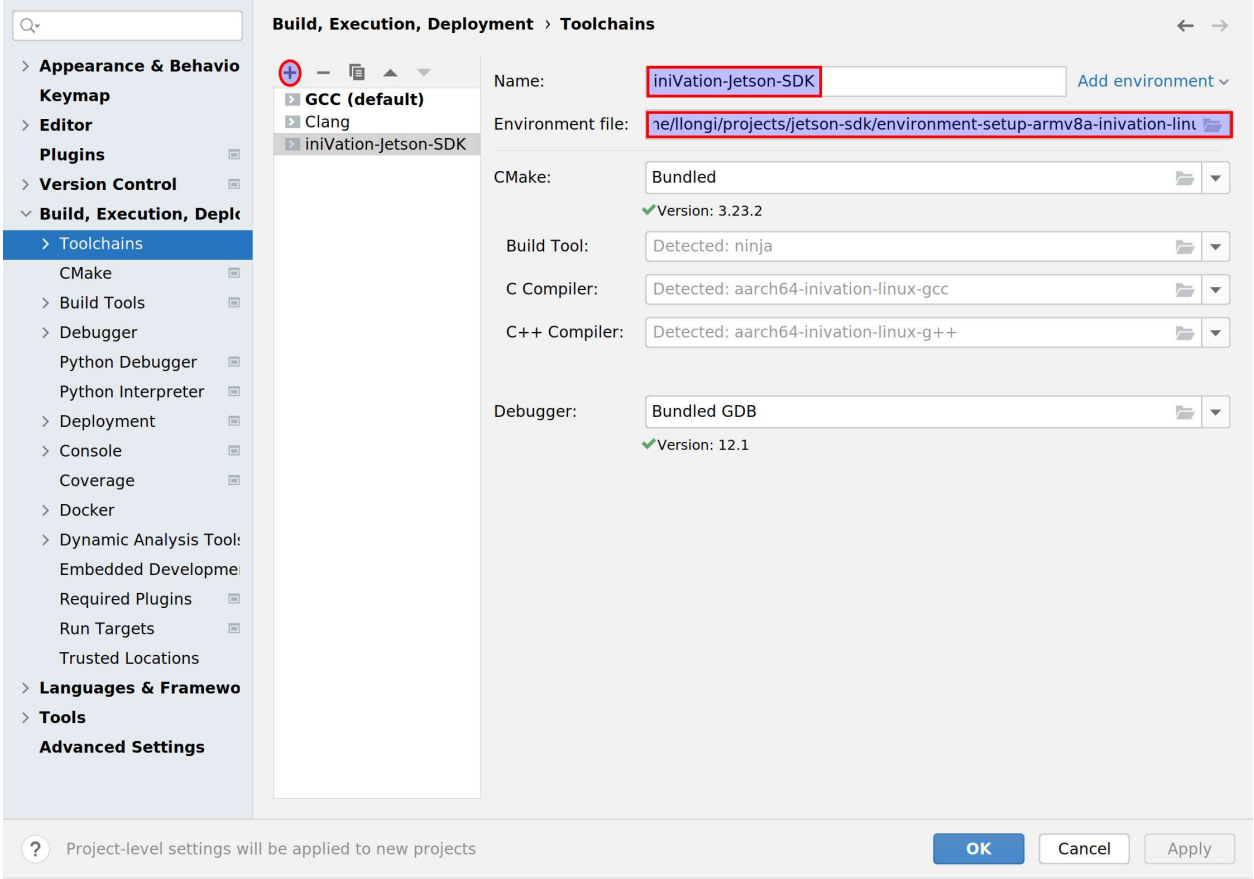
[8] https://release.inivation.com/?prefix=jetson/jetson-nano-inivation-v2/master/

[9] https://s3.eu-central-1.amazonaws.com/release.inivation.com/jetson/jetson-nano-inivation-v2/master/inivation-image-full-jetson-nano-inivation-v2-x86_64-glibc-toolchain-2023-06-07_13-18-51-master-ef6d1e5c.sh

8.2 Add Yocto SDK to CLion

Make sure to have at least version 2023.1 of CLion installed. In CLion you first have to add the SDK as a toolchain. Go to "Settings -> Build, Execution, Deployment -> Toolchains", press the + sign above the left-side list of toolchains and then select "System" to add a new system toolchain. Now change its name to "iniVation-Jetson-SDK" and click on "Add environment -> From File", then in the "Environment file" dialog select the SDK environment file at:

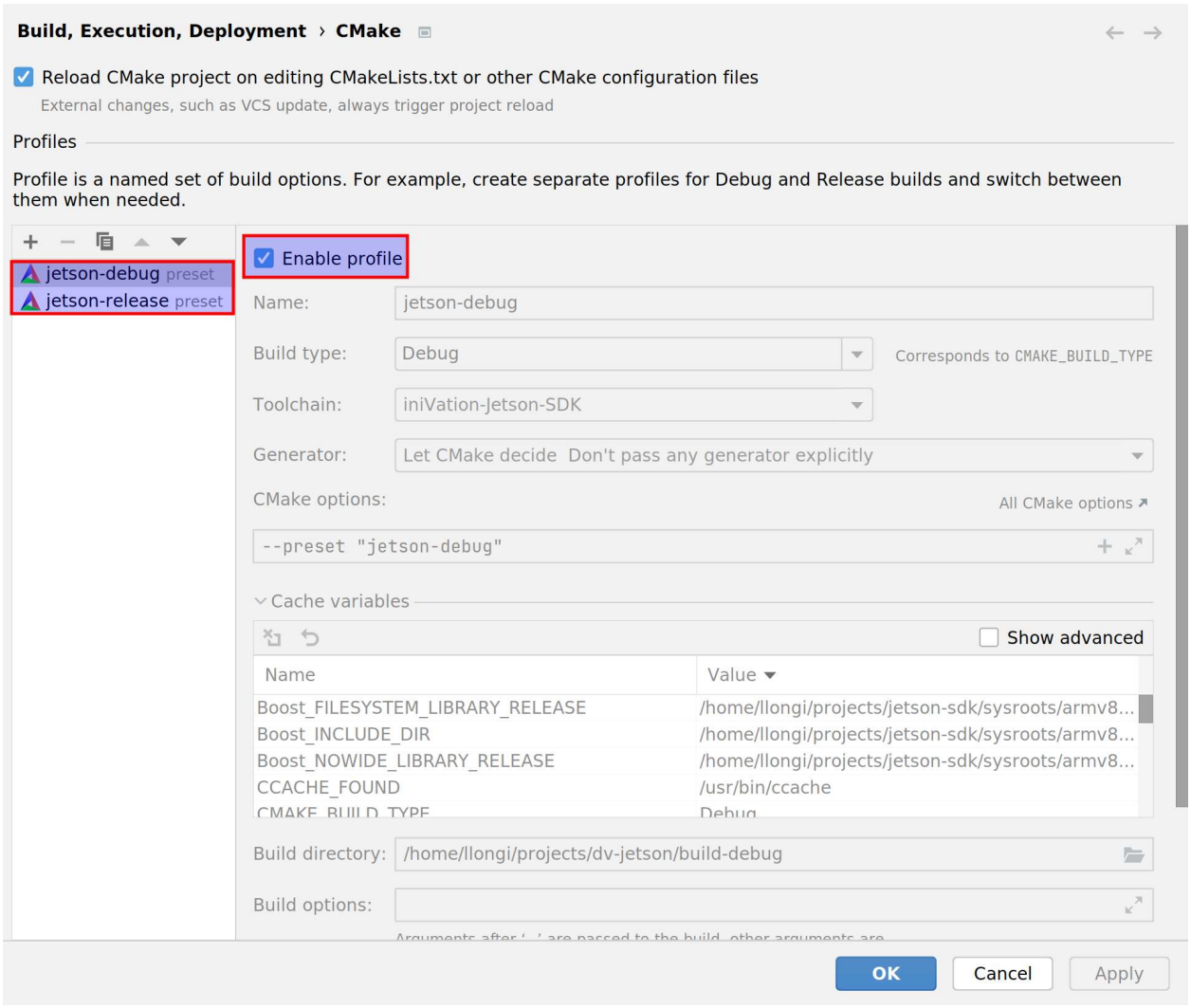
```
/home/youruser/inivation-sdk/environment-setup-armv8a-inivation-linux
```



Leave everything else unchanged, CMake and Debugger should remain the "Bundled" version. Please ensure the name is exactly "**iniVation-Jetson-SDK**".

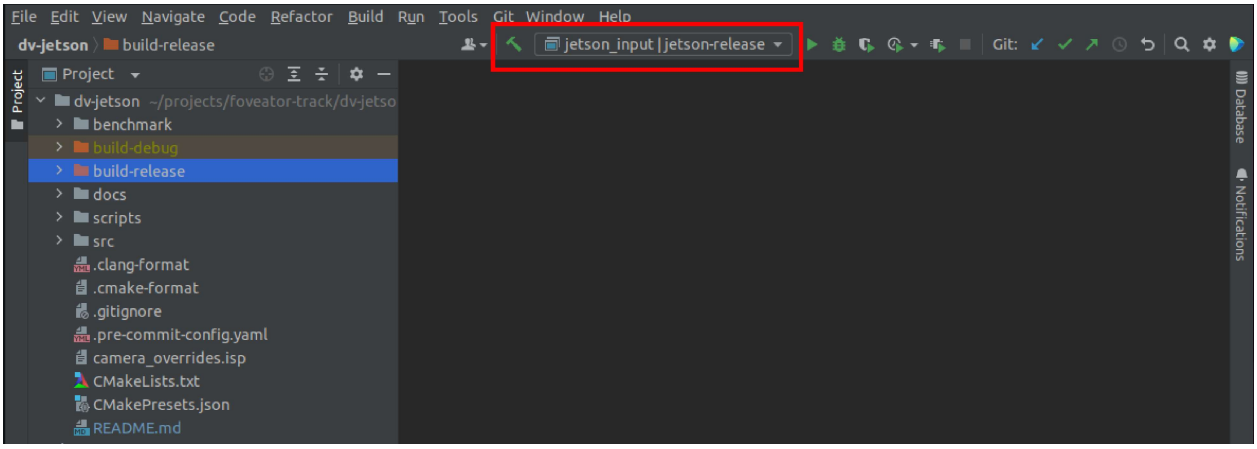
8.3 Open Project in CLion

Now open the *dv-example-cpp* project as you would normally by navigating to it and selectings its directory. The CMake Configuration window will pop up, disable and remove the default 'Debug' or 'Release' profiles, and instead enable the 'jetson-debug' and 'jetson-release' profiles. These are automatic CMake Presets that will configure all the needed paths and variables for you.



8.4 Build a Module in CLion

Once the CMake presets are set, you can run CMake and build the modules as you would normally do using CLion.



8.5 Transfer Modules to Device

To ease the transfer of built modules to the camera device during testing, we recommend using a script to automate this operation. The following will move the module to its proper place and restart dv-runtime on the device so it gets reloaded.

```
Usage: ./module-to-system.sh <device-IP-address> <path-to-built-module>

./module-to-system.sh 192.168.55.1 build-release/user_example.so
```

Script content (*module-to-system.sh*):

```
#!/bin/sh

sftp root@$1:/var/lib/dv-runtime/modules <<< $"put $2"
ssh root@$1 "systemctl restart dv-runtime.service"
```

CHAPTER

NINE

UPDATE SYSTEM IMAGE

Download the latest system image from release.inivation.com[10], you will need the *.mender* file.

Latest official release[11] (2023-06-07_13-18-51 git revision ef6d1e5c)

Upload it to the device using SFTP, we recommend putting the file in the temporary directory */var/tmp/*:

```
sftp root@192.168.55.1:/var/tmp/ <<< "put inivation-image-full-jetson-nano-inivation-
↪v2-2023-06-07_13-18-51-master-ef6d1e5c.mender"
```

Then log-in to the device via SSH, execute the installation command and reboot:

```
ssh root@192.168.55.1
mender install /var/tmp/inivation-image-full-jetson-nano-inivation-v2-2023-06-07_13-
↪18-51-master-ef6d1e5c.mender
reboot
```

Once the device has rebooted, if everything appears to work correctly, log-in via SSH again and confirm the update:

```
ssh root@192.168.55.1
mender commit
```

In case you forget this last step, the device will automatically roll-back to the previously installed version on the next reboot. This is not an issue, you can simply execute the upgrade steps again.

Todo: Later software releases will implement automatic commit upon successfull update completion, as well as automatic discovery, download and installation of updates on users' request.

[10] https://release.inivation.com/?prefix=jetson/jetson-nano-inivation-v2/master/

[11] https://s3.eu-central-1.amazonaws.com/release.inivation.com/jetson/jetson-nano-inivation-v2/master/inivation-image-full-jetson-nano-inivation-v2-2023-06-07_13-18-51-master-ef6d1e5c.mender

CHAPTER

TEN

HISTORY

- 2023-06-11: first version of documentation published

CHAPTER

ELEVEN

OPEN ISSUES

Todo: Later software releases will try to continuously improve the performance of algorithms and modules included by default in the DV-SDK software.

original entry

Todo: Later software releases will detect and report as much information as possible concerning power available, mode selected and current power consumption. We'll also implement automatically switching to higher performance modes based upon that information, so as to always offer optimal performance. You can use the *nvpmodel* tool over SSH for now if you want to force higher performance modes manually.

original entry

Todo: Later software releases will report in detail on measured temperature. You can use the *tegra-stats* tool over SSH for now if you want to get this information displayed.

original entry

Todo: Support for setting NTP server addresses manually and getting them from DHCP will come in a later software release. Later software releases will also show the time synchronization status in an easy to digest manner in DV.

original entry

Todo: MacOS USB Network Forwarding instructions not yet available.

original entry

Todo: Windows USB Network Forwarding instructions not yet available.

original entry

Todo: Later software releases will allow more flexible network configurations. For now, if you have need for a very specific network configuration, SSH into the device and edit the systemd-networkd configuration file at */data/config/20-wired.conf* manually.

original entry

Todo: Later software releases will offer a button to manually save the current configuration to the device at any point in time. It will still be recommended to properly shut it down via the appropriate buttons.

You can also always use the DV GUI Projects feature to save and reload any configuration states as XML files.

original entry

Todo: Later software releases will introduce automatic exposure control support, as well as more fine-grained controls for gain and other advanced features.

original entry

Todo: Future software releases will change how authentication works, disabling password-less root access and requiring users to log-in using SSH keys. A method to upload/import SSH keys over DV will be provided.

original entry

Todo: Instructions for setting up the Yocto SDK with the Qt-Creator and Visual Studio Code IDEs will follow.

original entry

Todo: Support for installing the SDK on Windows and MacOS will follow. MacOS should work the same as Linux but has not yet been tested.

original entry

Todo: Later software versions will show build version information in DV.

original entry

Todo: Later software releases will implement automatic commit upon successfull update completion, as well as automatic discovery, download and installation of updates on users' request.

original entry