HAILO

Empowering Intelligence

TAPPAS User Guide



Release 5.1.0

15 September 2025

Confidential and Proprietary. Unauthorized Reproduction Prohibited

Table of Contents

1 Hailo TAPPAS - Optimized Execution of Video-Processing Pipelines **2**
1.1 Overview . . . 2
1.2 Getting Started with Hailo-8 And Hailo-10H . . . 2
1.3 Getting Started with Hailo-15 . . . 3
1.4 Example Applications Built with TAPPAS . . . 3
1.5 Support . . . 4
1.6 Changelog . . . 4

2 Installation **5**
2.1 Verify HailoRT Installation . . . 5
2.2 Installation via the Hailo SW Suite . . . 5
2.3 Yocto . . . 6
2.4 Manual Installation . . . 9

3 Application Examples **14**
3.1 GStreamer Based General Applications . . . 14
3.2 Detection Pipeline . . . 14

4 TAPPAS as a framework **17**
4.1 TAPPAS Framework . . . 17
4.2 GStreamer Framework . . . 18
4.3 Write Your Own Application . . . 19

5 Elements **55**
5.1 Hailo Cropper . . . 55
5.2 Hailo Overlay . . . 57
5.3 Hailo DeviceStats . . . 58
5.4 Hailo Filter . . . 59
5.5 Hailo Tracker . . . 61
5.6 Hailo Tile Aggregator . . . 62
5.7 Hailo Tile Cropper . . . 64
5.8 Hailo Python . . . 66
5.9 Hailo Net . . . 67
5.10 Hailo Muxer . . . 67
5.11 Hailo Aggregator . . . 68
5.12 Hailo Gallery . . . 70
5.13 Hailo RoundRobin . . . 71
5.14 Hailo Stream Router . . . 73
5.15 Hailo Export File . . . 75
5.16 Hailo Export ZMQ . . . 76
5.17 Hailo Import ZMQ . . . 77
5.18 Hailo On-Screen Display . . . 78
5.19 Hailo Upload . . . 81
5.20 Hailo Gray to NV12 . . . 83
5.21 Hailo NV12 to Gray . . . 84

6 Pipelines **86**
6.1 Cascaded Networks Structure . . . 86
6.2 Single Network Multi-Device Pipeline Structure . . . 88
6.3 Multi Stream Pipeline Structure . . . 89
6.4 Parallel Networks Structure . . . 90
6.5 Single Network Pipeline Structure . . . 91

 Confidential and Proprietary | Copyright © 2025 – Hailo Technologies Ltd.

7 Tools **94**
7.1 Dot Visulaizer 94
7.2 Cross-compile Hailo's GStreamer plugins 94

8 Scripts **98**
8.1 Using record_perf script to get records from tappas 98

 Confidential and Proprietary | Copyright © 2025 – Hailo Technologies Ltd.

Disclaimer and Proprietary Information Notice

Copyright

© 2025 Hailo Technologies Ltd ("Hailo"). All Rights Reserved.

No part of this document may be reproduced or transmitted in any form without the expressed, written permission of Hailo. Nothing contained in this document should be construed as granting any license or right to use proprietary information for that matter, without the written permission of Hailo.

This version of the document supersedes all previous versions.

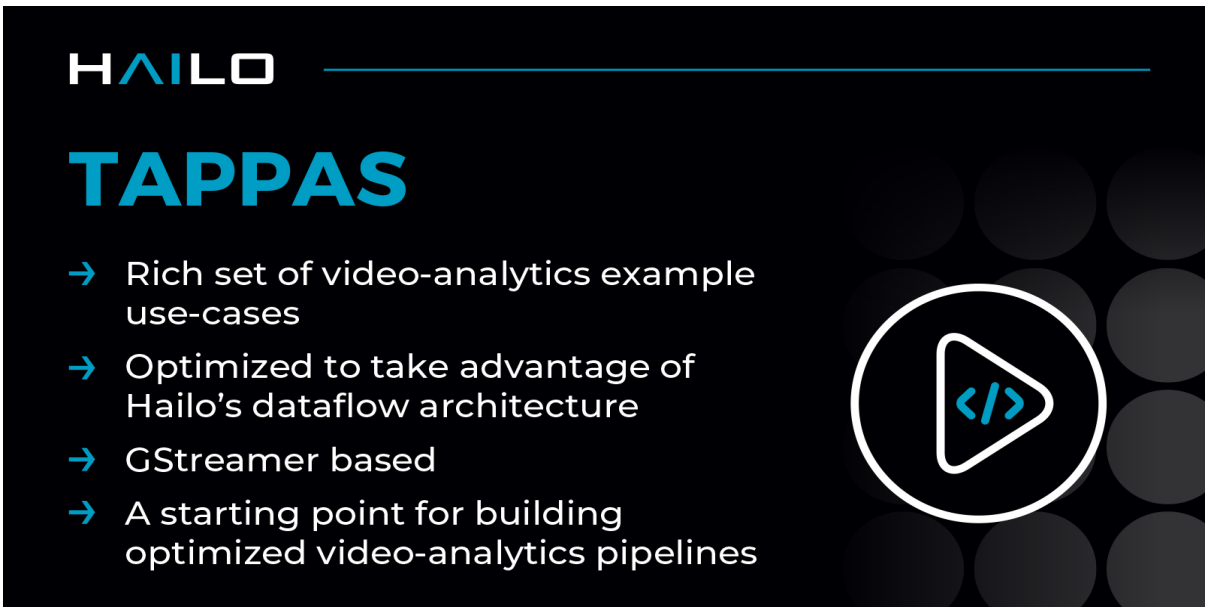
General Notice

Hailo, to the fullest extent permitted by law, provides this document "as-is" and disclaims all warranties, either express or implied, statutory or otherwise, including but not limited to the implied warranties of merchantability, non-infringement of third parties' rights, and fitness for particular purpose.

Although Hailo used reasonable efforts to ensure the accuracy of the content of this document, it is possible that this document may contain technical inaccuracies or other errors. Hailo assumes no liability for any error in this document, and for damages, whether direct, indirect, incidental, consequential or otherwise, that may result from such error, including, but not limited to loss of data or profits.

The content in this document is subject to change without prior notice and Hailo reserves the right to make changes to content of this document without providing a notification to its users.

1. Hailo TAPPAS - Optimized Execution of Video-Processing Pipelines



gstreamer 1.16 | 1.18 | 1.20 HailoRT 4.23.0 | 5.1.0 License LGPLv2.1

1.1. Overview

TAPPAS is Hailo's infrasturcture for building applications, implementing pipeline elements and pre-trained AI tasks.

Hailo apllications are now maintained at this repository.

Demonstrating Hailo's system integration scenario of specific use cases on predefined systems (software and Hardware platforms). It can be used for evaluations, reference code and demos:

- Accelerating time to market by reducing development time and deployment effort
- Simplifying integration with Hailo's runtime SW stack
- Providing a starting point for customers to fine-tune their applications

1.2. Getting Started with Hailo-8 And Hailo-10H

1.2.1. Prerequisites

- Hailo-8 or Hailo-10H device
- HailoRT PCIe driver installed
- At least 6GB's of free disk space

Note: This version is compatible with HailoRT v4.23.0 for Hailo-8 devices, and with HailoRT v5.1.0 for Hailo-10H devices.

 Confidential and Proprietary | Copyright © 2025 – Hailo Technologies Ltd.

1.2.2. Installation

Option	Instructions	Supported OS
Hailo SW Suite*	*SW Suite Install guide*	Ubuntu x86 24.04, Ubuntu x86 22.04
Manual install	*Manual install guide*	Ubuntu x86 24.04, Ubuntu x86 22.04, Ubuntu aarch64 20.04
Yocto installation	*Read more about Yocto installation*	Yocto supported BSP's
Raspberry Pi 5 installation	Read more about Raspberry Pi 5 installation	Raspberry Pi OS

`* It is recommended to start your development journey by first installing the Hailo SW Suite`

1.2.3. Documentation

- *Framework architecture and elements documentation*
- *Guide to writing your own C++ postprocess element*
- *Guide to writing your own Python postprocess element*
- *Debugging and profiling performance*
- *Cross compile* - A guide for cross-compiling

1.3. Getting Started with Hailo-15

TAPPAS is now released separately for Hailo-8 and Hailo-10H, for Hailo-15 please refer to https://github.com/hailo-ai/hailo-camera-apps.

For a quick start with Hailo-15, please refer to the Vision Processor Software Package documentation section in Hailo's Developer Zone.

1.4. Example Applications Built with TAPPAS

TAPPAS includes a *single-stream object detection pipeline* built on top of GStreamer. These example application is part of the Hailo AI Software Suite.

Hailo offers an additional set of Application Code Examples. For the Raspberry Pi 5 applications, go to Hailo Raspberry Pi 5 Examples.

Important:

- Example application utilize both the host (for non-neural tasks) and the Neural-Network Core (for neural-networks inference), therefore performance results are affected by the host.
- This application example does not include any architecture-specific accelerator usage, and therefore will provide the easiest way to run an application, but with sub-optimal performance.

 Confidential and Proprietary | Copyright © 2025 – Hailo Technologies Ltd.

Note: Running application examples requires a direct connection to a monitor.

1.5. Support

If you need support, please post your question on our Hailo community Forum for assistance.

Contact information is available at hailo.ai.

1.6. Changelog

v5.1.0 (October 2025)

- Downloader: removed redundant CLI arguments (`--platform`, `--app-list`);
- Downloader: HEF files now downloaded from `model_zoo` and media files from the TAPPAS bucket; removed the uploader;
- Detection app: `detection.sh` now supports `--arch` (Hailo-8/Hailo-10H);
- Models and resources: migrated model files to `model_zoo`; TAPPAS bucket is now used only for general MP4 files; updated resources directory structure; changed `yolov5m_wo_spp_60p.hef` to `yolov5m_wo_spp.hef`.
- Hailo□10H support: added Hailo-10H HEF downloads.
- Build and packaging: separated GCC apt installation and removed fixed GCC version; updated related documentation.
- Dependencies: updated package versions for Python 3.13 compatibility; migrated pandas to support the newer environment.
- Cleanup: removed Hailo□8 references where appropriate; removed nested directories under apps; various comment updates.
- This release supports both HailoRT v4.23.0 (Hailo-8) and HailoRT v5.1.0 (Hailo-10H)

v5.0.0 (July 2025)

- All example applications, except the object detection application, are now maintained at Hailo Applications.
- Updated manual installation process
- Added support for Ubuntu 24.04
- Added support for Python 3.12
- This release supports both HailoRT v4.22.0 (Hailo-8) and HailoRT v5.0.0 (Hailo-10H)
- Known issue: When installing via GitHub, only Hailo-8 models are downloaded.

 Confidential and Proprietary | Copyright © 2025 – Hailo Technologies Ltd.

2. Installation

2.1. Verify HailoRT Installation

Note: This section does not apply to Hailo SW Suite users since HailoRT is already installed.

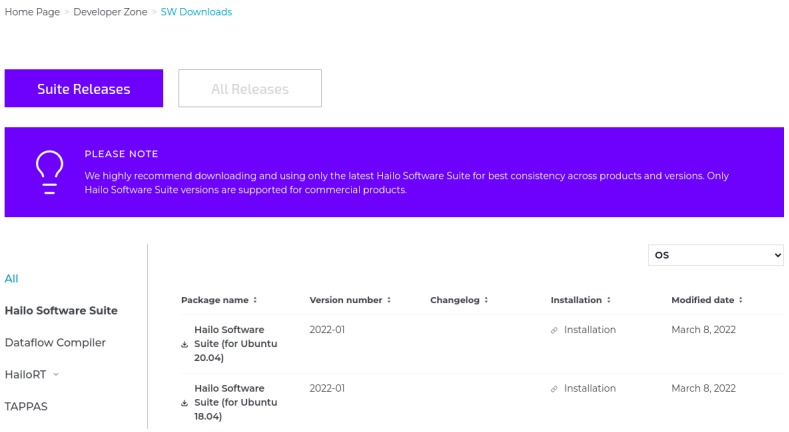
Confirm that hailo has been identified correctly by running this command: `hailortcli fw-control identify`, The expected output should look similar to the one below:

```
$ hailortcli fw-control identify
Identifying board
Control Protocol Version: 2
Firmware Version: X.X.X (develop,app)
Logger Version: 0
Board Name: Hailo-8
Device Architecture: HAILO8_B0
Serial Number: 0000000000000009
Part Number: HM218B1C2FA
Product Name: HAILO-8 AI ACCELERATOR M.2 M KEY MODULE
```

2.2. Installation via the Hailo SW Suite

2.2.1. Download the SW Suite

To get started, go to the Hailo Developer Zone and enter the SW Suite download page. From there you can download your prefered version of the `Hailo SW Suite`. If this is your first time in the `Developer Zone`, you will be asked to register to the site before downloads are enabled, login and then you will be able to download the package.



2.2.2. Install the SW Suite

Once the zip is downloaded, extract the folder with:

```
tar -xf hailo_sw_suite_2023_01.tar.gz
```

There are three options for installing the `Suite`: `Docker` installation, `Manual` installation, and a `Self Extractable` installation.

 Confidential and Proprietary | Copyright © 2025 – Hailo Technologies Ltd.

The instructions can be found for all installation flows, along with their required pre-requisites and sanity checks, in the Hailo Developer Zone.

This is all that is required to start the installation. TAPPAS is included in the SW Suite, so by installing the Suite you should now have the corresponding version of TAPPAS installed inside.

Note that the TAPPAS system requirements are documented in the Required Packages section *Here*.

2.3. Yocto

This section will guide through the integration of Hailo's Yocto layer's into your own Yocto environment.

Two layers are provided by Hailo, the first one is meta-hailo which compiles the HailoRT sources, and the second one is meta-hailo-tappas which compiles the TAPPAS sources.

meta-hailo-tappas is a layer that based un-top of meta-hailo that adds TAPPAS recipes.

The layers are stored in Meta-Hailo Github, with branch for each supported yocto release:

- Dunfell 3.1 (kernel 5.4.85)
- Kirkstone 4.0 (kernel 5.15)

Warning: On i.MX8-based devices Kirkstone branch does not support OpenGL, therefore, the Kirkstone applications portfolio is reduced.

2.3.1. Setup

HailoRT

Add the following to your image in your conf/bblayers.conf:

```
BBLAYERS += " ${BSPDIR}/sources/meta-hailo/meta-hailo-accelerator \
       ${BSPDIR}/sources/meta-hailo/meta-hailo-libhailort"
```

Add the recipes to your image in your conf/local.conf:

```
IMAGE_INSTALL_append = "hailo-firmware libhailort hailortcli hailo-pci libgsthailo"
```

TAPPAS

Add the following to your image in your conf/bblayers.conf:

```
BBLAYERS += "${BSPDIR}/sources/meta-hailo/meta-hailo-tappas"
```

Add the following to your image in your conf/local.conf:

```
IMAGE_INSTALL:append = "libgsthailotools tappas-apps hailo-post-processes tappas-
↪tracers"
```

 Confidential and Proprietary | Copyright © 2025 – Hailo Technologies Ltd.

2.3.2. Build your image

Run bitbake and build your image. After the build successfully finished, burn the Image to your embedded device.

Note: building on non-IMX devices: To increase the performance of our applications, we patched imx gstreamer-plugins-base. In non-IMX devices you may encounter an error indicating that recipes under `meta-hailo-tappas/recipes-multimedia/gstreamer/` cannot be parsed. In this case remove this directory under the meta-hailo-tappas layer, and re-build the image.

```
rm -rf meta-hailo/meta-hailo-tappas/recipes-multimedia/gstreamer/
```

Validating the integration's success

Make sure that the following conditions have been met on the target device:

- Running `hailortcli fw-control identify` prints the right configurations
- Running `gst-inspect-1.0 | grep hailo` returns hailo elements:

```
hailo: hailonet: hailonet element
hailodevicestats: hailodevicestats element
```

- Running `gst-inspect-1.0 | grep hailotools` returns hailotools elements:

```
hailotools: hailomuxer: Muxer pipe fitting
hailotools: hailofilter: Hailo postprocessing and drawing element
...
```

- post-processes shared object files exists at `/usr/lib/hailo-post-processes`

2.3.3. Recipes

libgsthailo

Hailo's GStreamer plugin for running inference on the hailo8 chip. Depends on `libhailort` and GStreamer.

The recipe compiles and copies the `libgsthailo.so` file to `/usr/lib/gstreamer-1.0` on the target device's root file system, make it loadable by GStreamer as a plugin.

libgsthailotools

Hailo's TAPPAS gstreamer elements. Depends on `libgsthailo`, GStreamer, opencv, xtensor and xtl. The source files located in the TAPPAS release under `core/hailo`. The recipe compiles with meson and copies the `libgsthailotools.so` file to `/usr/lib/gstreamer-1.0` on the target device's root file system.

 Confidential and Proprietary | Copyright © 2025 – Hailo Technologies Ltd.

tappas-apps

Hailo's TAPPAS embedded application recipe, including GStreamer apps for embedded. The recipe copies the app script, the hef and media files to /home/root/apps/. Depends on GStreamer, opencv, cxxopts, xtensor and xtl.

hailo-post-processes

The recipe compiles and copies the post processes to `/usr/lib/hailo-post-processes`. Deppends on opencv, xtensor, xtl, rapidjson and cxxopts.

tappas-tracers

Hailo's TAPPAS gstreamer tracers. Depends on `libgsthailo` and GStreamer. The source files located in the TAPPAS release under `core/hailo/tracers`. The recipe compiles with meson and copies the `libgsthailotracers.so` file to `/usr/lib/gstreamer-1.0` on the target device's root file system.

For instructions on how to use the tracers on a yocto built machine, see *debugging*

2.3.4. Troubleshooting

1. The device does not appear on lspci

If the device does not appear after running lspci, there may be two possible reasons:

- Symptom:

  The device is not connected correctly

- Symptom:

  The u-boot device tree does not support pcie.

  Solution:

  To fix this, replace the ftd_file you are using on u-boot.

```
setenv fdt_file imx6q-sabresd-pcie.dtb
```

2. HDMI port is connected but there is no display

Symptom:

On some imx devices you need to manually configure the u-boot to show video using HDMI port.

Solution:

To fix this issue you should set the u-boot to use HDMI port, defining the resolution, FPS and output format. The configuration is "added" (do not override this) to the mmcargs:

For example on IMX6Q-Sabresd, this the default value of mmargs:

```
mmcargs="setenv bootargs console=${console},${baudrate} ${smp} root=$
↪{mmcroot}"
```

Using this command we add the needed info to this variable:

```
setenv mmcargs "setenv bootargs console=${console},${baudrate} ${smp} root=
↪${mmcroot} video=mxcfb0:dev=hdmi,1280x720M@30,if=RGB24"
```

 Confidential and Proprietary | Copyright © 2025 – Hailo Technologies Ltd.

2.4. Manual Installation

The manual installation of TAPPAS requires preparation, Hailo's recommended method is to begin with `Hailo SW Suite`. This guide will instruct how to install the required components manually.

Note: Only Ubuntu 20.04 and 22.04 are supported

2.4.1. Hailort Installation

First HailoRT + HailoRT PCIe driver, needs to be installed. Follow the HailoRT installation guide for further instructions. After the installation, *confirm that HailoRT is working correctly*.

2.4.2. Preparations

Download from Hailo developer zone `tappas_VERSION_linux_installer.zip`.

x86

Unzip `tappas_VERSION_linux_installer.zip`

```
unzip tappas_VERSION_linux_installer.zip
```

Non-x86

1. Unzip TAPPAS and clone HailoRT sources

   ```
   unzip tappas_VERSION_linux_installer.zip
   ```

2. Change the directory to TAPPAS, and create a directory named `hailort` and clone `HailoRT` sources

   ```
   cd `tappas_VERSION`
   mkdir hailort
   git clone https://github.com/hailo-ai/hailort.git hailort/sources
   ```

Note: In some cases, HailoRT (which is installed via TAPPAS) is not the latest version, while the driver, which was installed before HailoRT and TAPPAS - has a different version. In these cases, this error will occur:

> "Could not find a configuration file for package "HailoRT" that exactly matches requested version."

Meaning, HailoRT library and driver installed versions are mismatched - while they are required to be fully identical.

In such a case - re-install the driver version which is compatible to your installed HailoRT version.

 Confidential and Proprietary | Copyright © 2025 – Hailo Technologies Ltd.

2.4.3. Required Packages

The following APT packages need to be installed, using the command below:

- ffmpeg
- x11-utils
- python3 (pip and setuptools).
- python3-virtualenv
- python-gi-dev
- libgirepository1.0-dev
- cmake
- libzmq3-dev
- git
- rsync
- gcc (>= gcc-9)
- g++ (>= g++-9)

To install the above packages, run the following command:

```
sudo apt-get install -y rsync ffmpeg x11-utils python3-dev python3-pip python3-
↪setuptools python3-virtualenv python-gi-dev libgirepository1.0-dev gcc-12 g++-12
↪cmake git libzmq3-dev
```

Note: The gcc and g++ version depends on the OS support. The versions shown in the example (gcc-12, g++-12) are recommended, but ensure the version is at least gcc-9 and g++-9 or higher.

The following packages are required as well, and their installation instructions can be viewed from the links below:

- *OpenCV installation*.
- *GStreamer installation*.
- *PyGobject installation*.

In case any requirements are missing, a requirements table will be printed when calling manual installation.

2.4.4. OpenCV Installation

To install OpenCV, run the following commands:

```
sudo apt-get install -y libopencv-dev python3-opencv
```

To check your OpenCV version, run the following command:

```
# To check the OpenCV version installed
pkg-config --modversion opencv4
```

Tip: If you are running on an old OS the apt-get version might be too old (You will be notified on the next steps), you can install OpenCV manually as shown below.

 Confidential and Proprietary | Copyright © 2025 – Hailo Technologies Ltd.

Opencv compilation from source

```
# Download Opencv and unzip
wget https://github.com/opencv/opencv/archive/4.5.2.zip
unzip 4.5.2.zip

# cd and make build dir
cd opencv-4.5.2
mkdir build
cd build

# Make and install
cmake -DOPENCV_GENERATE_PKGCONFIG=ON \
  -DBUILD_LIST=core,imgproc,imgcodecs,calib3d,features2d,flann \
  -DCMAKE_BUILD_TYPE=RELEASE \
  -DWITH_PROTOBUF=OFF -DWITH_QUIRC=OFF \
  -DWITH_WEBP=OFF -DWITH_OPENJPEG=OFF \
  -DWITH_GSTREAMER=OFF -DWITH_GTK=OFF \
  -DOPENCV_DNN_OPENCL=OFF -DBUILD_opencv_python2=OFF \
  -DINSTALL_C_EXAMPLES=ON \
  -DINSTALL_PYTHON_EXAMPLES=ON \
  -DCMAKE_INSTALL_PREFIX=/usr/local ..

num_cores_to_use=$(($(nproc)/2))
make -j$num_cores_to_use
sudo make install

# Update the linker
sudo ldconfig
```

2.4.5. GStreamer Installation

Run the following command to install GStreamer:

```
sudo apt-get install -y libcairo2-dev libgirepository1.0-dev libgstreamer1.0-dev
↪libgstreamer-plugins-base1.0-dev libgstreamer-plugins-bad1.0-dev gstreamer1.0-
↪plugins-base gstreamer1.0-plugins-good gstreamer1.0-plugins-bad gstreamer1.0-
↪plugins-ugly gstreamer1.0-libav gstreamer1.0-tools gstreamer1.0-x gstreamer1.0-
↪alsa gstreamer1.0-gl gstreamer1.0-gtk3 gstreamer1.0-qt5 gstreamer1.0-pulseaudio
↪gcc-12 g++-12 python-gi-dev
```

Please refer to: GStreamer official installation guide for more details

2.4.6. PyGobject Installation

Run the following command to install PyGobject:

```
sudo apt install python3-gi python3-gi-cairo gir1.2-gtk-3.0
```

Please refer to: PyGobject official installation guide for more details

 Confidential and Proprietary | Copyright © 2025 – Hailo Technologies Ltd.

2.4.7. TAPPAS Installation

On most platforms (such as x86-based platforms), run:

```
./install.sh --skip-hailort
```

and then, *Make sure that HailoRT works*

On Rockchip, run:

```
./install.sh --skip-hailort --target-platform rockchip
```

and then, return to the Rockchip section.

2.4.8. Upgrade TAPPAS

To Upgrade TAPPAS, first clean the GStreamer cache

```
rm -rf ~/.cache/gstreamer-1.0/
```

Remove old `libgsthailotools.so`

```
rm /usr/lib/$(uname -m)-linux-gnu/gstreamer-1.0/libgsthailotools.so
```

and then, *TAPPAS installation section*

2.4.9. Troubleshooting

Cannot allocate memory in static TLS block

In some scenarios (especially `aarch64`), this warning might occur:

```
(gst-plugin-scanner:15): GStreamer-WARNING **: 13:58:20.557: Failed to load plugin '/
↪usr/lib/aarch64-linux-gnu/gstreamer-1.0/libgstlibav.so': /lib/aarch64-linux-
↪gnu/libgomp.so.1: cannot allocate memory in static TLS block
```

The solution is to export an environment variable:

```
export LD_PRELOAD=/usr/lib/aarch64-linux-gnu/libgomp.so.1
```

PCIe descriptor page size error

If you encounter the following error: (actual page size might vary)

```
[HailoRT] [error] CHECK_AS_EXPECTED failed - max_desc_page_size given 16384 is
↪bigger than hw max desc page size 4096"
```

Some hosts doesn't support certain PCIe descriptor page size. in order to overcome this issue add the text below to /etc/modprobe.d/hailo_pci.conf (create the file if it doesn't exist)

```
options hailo_pci force_desc_page_size=4096
# you can do this by running the following command:
echo 'options hailo_pci force_desc_page_size=4096' >> /etc/modprobe.d/hailo_pci.
↪conf
```

Reboot the machine for this change to take effect. You can also reload the driver without rebooting by running the following commands:

 Confidential and Proprietary | Copyright © 2025 – Hailo Technologies Ltd.

```
modprobe -r hailo_pci
modprobe hailo_pci
```

 Confidential and Proprietary | Copyright © 2025 – Hailo Technologies Ltd.

3. Application Examples

3.1. GStreamer Based General Applications

3.1.1. Where to Begin?

Important: Example applications performance varies on different hosts (affected by the host's processing power and throughput).

---

1. *Detection* - single-stream object detection pipeline on top of GStreamer using the Hailo-8 device.

3.2. Detection Pipeline

3.2.1. Overview

`detection.sh` **demonstrates detection on one video file source and verifies Hailo's configuration.** This is done by running a `single-stream object detection pipeline` on top of GStreamer using the Hailo-8 device.

3.2.2. Options

```
./detection.sh [--input FILL-ME]
```

- `--network` is an optional flag that sets which network to use. Choose from [yolov5, mobilenet_ssd, yolov8], default is yolov8. This will set which `hef file` to use, the corresponding `hailofilter` function, and the scaling of the frame to match the width/height input dimensions of the network.
- `--arch` is an optional flag that sets which Hailo architecture to use, default is h8. Choose from [h8, h10].
- `--input` is an optional flag, a path to the video displayed (default is detection.mp4).
- `--show-fps` is an optional flag that enables printing FPS on screen.
- `--print-gst-launch` is a flag that prints the ready gst-launch command without running it.
- `--print-device-stats` prints the power and temperature measured on the Hailo device.

3.2.3. Configuration

In case the selected network is yolo, the app post process parameters can be configured by a json file located in $TAPPAS_WORKSPACE/apps/detection/resources/configs

 Confidential and Proprietary | Copyright © 2025 – Hailo Technologies Ltd.

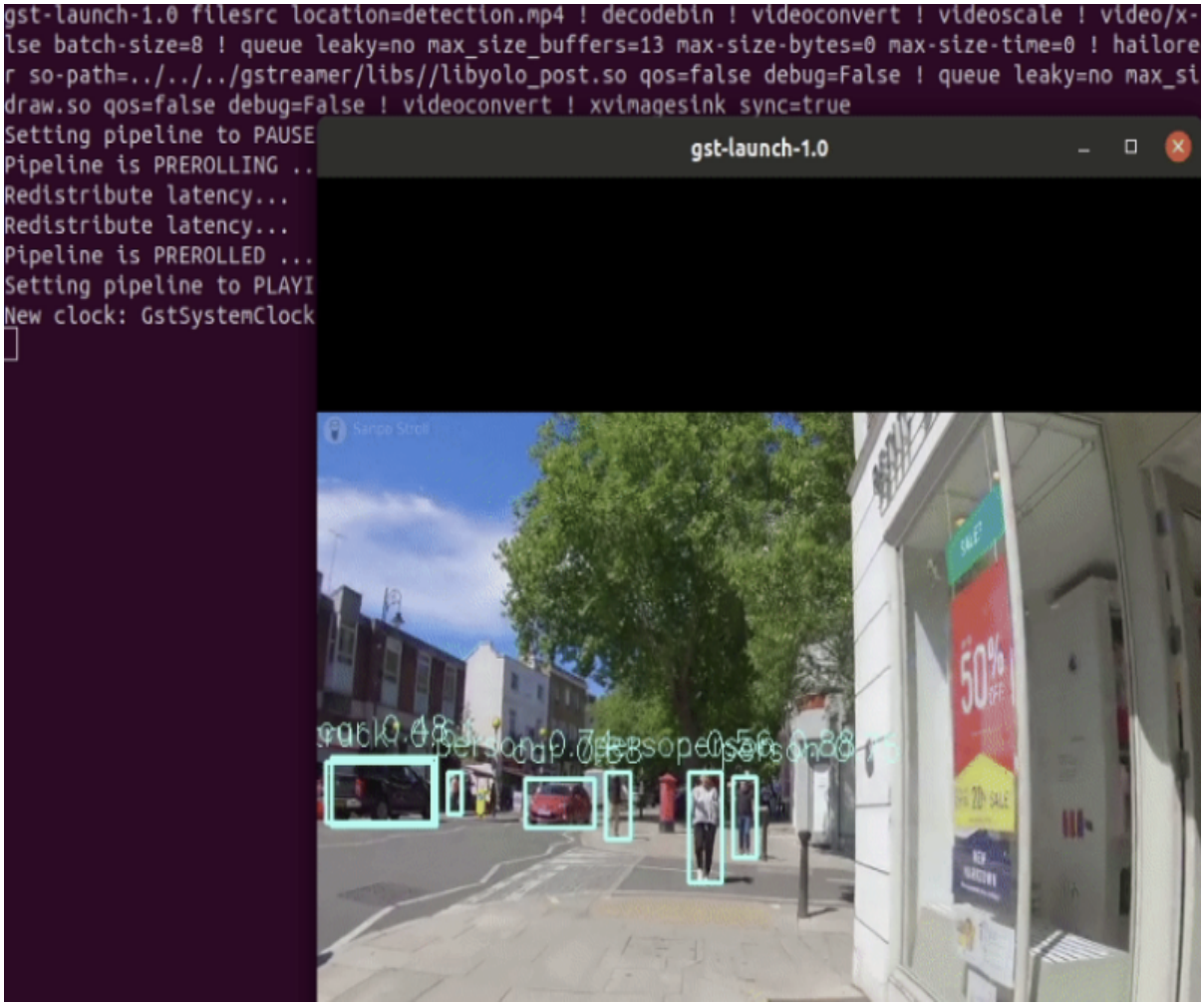
3.2.4. Supported Networks

- 'yolov8m' - https://github.com/hailo-ai/hailo_model_zoo/blob/master/hailo_model_zoo/cfg/networks/yolov8m.yaml
- 'yolov5m_wo_spp' - https://github.com/hailo-ai/hailo_model_zoo/blob/master/hailo_model_zoo/cfg/networks/yolov5m_wo_spp.yaml
- 'mobilenet_ssd' - https://github.com/hailo-ai/hailo_model_zoo/blob/master/hailo_model_zoo/cfg/networks/ssd_mobilenet_v1.yaml

3.2.5. Run

```
cd $TAPPAS_WORKSPACE/apps/detection
./detection.sh
```

The output should display as:



 Confidential and Proprietary | Copyright © 2025 – Hailo Technologies Ltd.

3.2.6. Method of Operation

This app is based on our *single network pipeline template*

3.2.7. How to Use Retraining to Replace Models

Note: It is recommended to first read the *Retraining TAPPAS Models* page.

Retraining Dockers (available on Hailo Model Zoo), can be used to replace the following models with ones that are trained on the users own dataset:

- `yolov8m`
  - Retraining docker
    - For optimum compatibility and performance with TAPPAS, use for compilation the corresponding YAML file from above.
  - TAPPAS changes to replace model:
    - Update HEF_PATH on the .sh file
- `yolov5m`
  - Retraining docker
    - For optimum compatibility and performance with TAPPAS, use for compilation the corresponding YAML file from above.
  - TAPPAS changes to replace model:
    - Update HEF_PATH on the .sh file
- `mobilenet_ssd`
  - Retraining docker
    - For optimum compatibility and performance with TAPPAS, use for compilation the corresponding YAML file from above.
  - TAPPAS changes to replace model:
    - Update HEF_PATH on the .sh file
    - Update mobilenet_ssd.cpp with your new parameters, then recompile to create `libmobilenet_ssd_post.so`

 Confidential and Proprietary | Copyright © 2025 – Hailo Technologies Ltd.

4. TAPPAS as a framework

4.1. TAPPAS Framework

TAPPAS is a GStreamer based library of plug-ins. It enables using a Hailo devices within gstreamer pipelines to create intelligent video processing applications.

4.1.1. What is GStreamer?

GStreamer is a framework for creating streaming media applications.

GStreamer's development framework makes it possible to write any type of streaming multimedia application. The GStreamer framework is designed to simplify for the user how to write applications that handle audio or video or both. It isn't restricted to audio and video and can process any kind of data flow. The framework is based on plugins that will provide various codecs and other functionalities. The plugins can be linked and arranged in a pipeline. This pipeline defines the flow of the data. The GStreamer core function is to provide a framework for plugins, data flow, and media type handling/negotiation. It also provides an API to write applications using the various plugins.

For additional details check *GStreamer overview*

4.1.2. Hailo GStreamer Concepts

The functionality that Hailo brings into the GStreamer-framework allows us to infer video frames easily and intuitively without compromising on performance and flexibility.

Hailo Concepts

- **Network encapsulation** - Since in a configured network group, there are only input and output layers a GstHailoNet will be associated to a "Network" by its configured input and output pads
- **Network independent elements** - The GStreamer elements will be network independent, so the same infrastructure elements can be used for different applicative pipelines that use different NN functionality, configuration, activation, and pipelines. Using the new API we can better decouple network configuration and activation stages and thus better support network switch
- **GStreamer Hailo decoupling** - Applicative code will use Hailo API and as such will be GStreamer independent. This will help us build and develop the NN and postprocessing functionality in a controlled environment (with all modern IDE and debugging capabilities).
- **Context control** - Hailo elements will be contextless and thus leave the context (thread) control to the pipeline builder
- **GStreamer reuse** - The pipeline will use many off the shelf GStreamer elements

Hailo GStreamer Elements

- *HailoNet* - Element for sending and reciveing data from Hailo-8 chip
- *HailoFilter* - Element that enables the user to apply a postprocess or drawing operation to a frame and its tensors
- *HailoPython* - Element that enables the user to apply a postprocess or drawing operation to a frame and its tensors via python.
- *HailoMuxer* - Muxer element used for Multi-Hailo-8 setups
- *HailoDeviceStats* - Hailodevicestats is an element that samples power and temperature
- *HailoAggregator* - HailoAggregator is an element designed for applications with cascading networks. It has 2 sink pads and 1 source

 Confidential and Proprietary | Copyright © 2025 – Hailo Technologies Ltd.

- *HailoCropper* - HailoCropper is an element designed for applications with cascading networks. It has 1 sink and 2 sources
- *HailoTileAggregator* - HailoTileAggregator is an element designed for applications with tiles. It has 2 sink pads and 1 source
- *HailoTileCropper* - HailoCropper is an element designed for applications with tiles. It has 1 sink and 2 sources
- *HailoTracker* - HailoTracker is an element that applies Joint Detection and Embedding (JDE) model with Kalman filtering to track object instances.
- *HailoRoundRobin* - HailoRoundRobin is an element that provides muxing functionality in roundrobin method.
- *HailoStreamRouter* - HailoStreamRouter is an element that provides de-muxing functionality.

4.2. GStreamer Framework

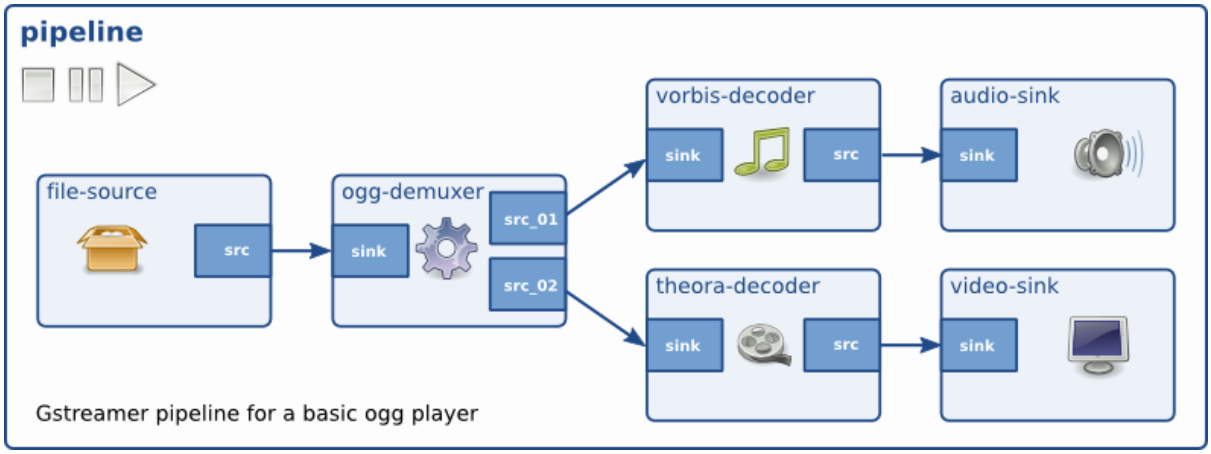
4.2.1. GStreamer Principles

- Object-oriented - All GStreamer Objects can be extended using the GObject inheritance methods. All plugins are loaded dynamically and can be extended and upgraded independently.
- GStreamer adheres to GObject, the GLib 2.0 object model. A programmer familiar with GLib 2.0 or GTK+ will be comfortable with GStreamer.
- Extensible core plugins to encapsulate all common media streaming functionalities.
- Allow binary-only plugins - Plugins are shared libraries that are loaded at runtime.
- High performance
  - using GLib's GSlice allocator
  - ref-counting and copy on write minimize the usage of memcpy.
  - allowing hardware acceleration by using specialized plugins.

4.2.2. GStreamer Elements

- **Elements** - have one specific function for processing/ generating / consuming data. By chaining together several such elements, a pipeline can be created to perform a specific task.
- **Pads** - are an element's input and output, which can connect to other elements. A pad can be viewed as a "plug" or "port" on an element where links may be made with other elements, and through which data can flow to or from those elements. Data types are negotiated between pads using a process called Caps Negotiation. Data types are described by GstCaps.
- **Bin** - A bin is a container for a collection of elements. Since bins are subclasses of elements themselves, a bin can be mostly controlled as if it was an element, thereby subtracting away a lot of complexity from the user's application. A pipeline is a top-level bin. It provides a bus for the application and manages the synchronization for its children.

 Confidential and Proprietary | Copyright © 2025 – Hailo Technologies Ltd.



4.2.3. Terminology

NVR (Network Video Recorder)

NVR is a specialized hardware and software solution used in IP (Internet Protocol) video surveillance systems. In most cases, the NVR is intended for obtaining video streams from the IP cameras (via the IP network) for the purpose of storage and subsequent playback.

Real Time Streaming Protocol (RTSP)

Is a network control protocol designed for use in entertainment and communications systems to control streaming media servers. This protocol is used for establishing and controlling media sessions between endpoints.

Rockchip Media Process Platform (MPP)

The MPP is a library for Rockchip SoC's which provides high performance on multimedia (video and image) processing.

4.3. Write Your Own Application

4.3.1. Compiling Your Code

The Meson Build System

Meson is an open source build system that places the emphasis on speed and ease of use. GStreamer uses meson for all sub-projects to generate build instructions to be executed by ninja, another build system focuses entirely on a speed that requires a higher level build system (ie: meson) to generate its input files. Similar to GStreamer, `TAPPAS` also uses Meson, and compiling new projects requires the adjustment of the `meson.build` files.

 Confidential and Proprietary | Copyright © 2025 – Hailo Technologies Ltd.

How to Compile

To help streamline this process the TAPPAS Framework have provided a script that handles most of the work. The script can be found at scripts/gstreamer/install_hailo_gstreamer.sh.
The following arguments are available:

- `--build-dir` Path to the build directory. Defaults to `core/hailo`.
- `--build-mode` Build mode, debug/release, default is release.
- `--skip-hailort` Skip compiling HailoRT.
- `--python-version` Specify which Python version to use.
- `--compile-libgsthailo` Compile libgsthailo instead of copying it from the release.
  From the TAPPAS home directory folder you can run:

```
./scripts/gstreamer/install_hailo_gstreamer.sh
```

```
hailo@hailo_tappas:/local/workspace/tappas$ ./scripts/gstreamer/install_hailo_gstreamer.sh
/local/workspace/tappas/core/hailo/gstreamer /local/workspace/tappas
Directory already configured.

Just run your build command (e.g. ninja) and Meson will regenerate as necessary.
If ninja fails, run "ninja reconfigure" or "meson --reconfigure"
to force Meson to regenerate.

If build failures persist, run "meson setup --wipe" to rebuild from scratch
using the same options as passed when configuring the build.
To change option values, run "meson configure" instead.
ninja: Entering directory `build.release'
[0/1] Regenerating build files.
The Meson build system
Version: 0.58.0
Source dir: /local/workspace/tappas/core/hailo/gstreamer
Build dir: /local/workspace/tappas/core/hailo/gstreamer/build.release
Build type: native build
Project name: gst-hailo-tools
Project version: 3.16.0
C compiler for the host machine: gcc-9 (gcc 9.4.0 "gcc-9 (Ubuntu 9.4.0-1ubuntu1~18.04) 9.4.0")
C linker for the host machine: gcc-9 ld.bfd 2.30
C++ compiler for the host machine: g++-9 (gcc 9.4.0 "g++-9 (Ubuntu 9.4.0-1ubuntu1~18.04) 9.4.0")
C++ linker for the host machine: g++-9 ld.bfd 2.30
Host machine cpu family: x86_64
Host machine cpu: x86_64
Configuring config.h using configuration
Message: GCC >= 9.0.0 detected, applying extra arguments.
Dependency gstreamer-1.0 found: YES 1.14.5 (cached)
Dependency gstreamer-base-1.0 found: YES 1.14.5 (cached)
Dependency gstreamer-app-1.0 found: YES 1.14.5 (cached)
Dependency gstreamer-video-1.0 found: YES 1.14.5 (cached)
Dependency opencv4 found: YES 4.5.2 (cached)
Dependency glib-2.0 found: YES 2.56.4 (cached)
Dependency gmodule-2.0 found: YES 2.56.4 (cached)
Dependency gobject-2.0 found: YES 2.56.4 (cached)
Dependency pygobject-3.0 found: YES 3.26.1 (cached)
Program python3.6 found: YES (/usr/bin/python3.6)
Found pkg-config: /usr/bin/pkg-config (0.29.1)
Dependency python found: YES (pkgconfig)
Dependency python found: YES (pkgconfig)
Dependency blas found: YES 3.7.1 (cached)
Dependency lapack found: YES 3.7.1 (cached)
Library dl found: YES
Library stdc++fs found: YES
Dependency threads found: YES unknown (cached)
Build targets in project: 29
```

 Confidential and Proprietary | Copyright © 2025 – Hailo Technologies Ltd.

4.3.2. Hailo Objects API

The TAPPAS provides a set of abstractions for the different objects that might be handled in a user's pipeline, such as `tensors`, `detections`, `classifications`, or `landmarks`. If the user wants to write their own custom pipeline behavior (`postprocessing`, `drawing`, etc..), then it is worthwhile to become familiar with these classes.

This guide documents the different classes available in the TAPPAS and their interactions.

Structs

HailoBBox

Represents a bounding box.

SOURCE: core/hailo/general/hailo_objects.hpp

Constructor

```
HailoBBox(float xmin, float ymin, float width, float height)
```

Functions

Function	Return Type	Description
`xmin()`	float	Normalized xmin position.
`ymin()`	float	Normalized ymin position.
`width()`	float	Normalized width of bounding box.
`height()`	float	Normalized height of bounding box.
`xmax()`	float	Normalized xmax position.
`ymax()`	float	Normalized ymax position.

HailoPoint

Represents a detected point (landmark).

SOURCE: core/hailo/general/hailo_objects.hpp

Constructor

```
HailoPoint(float x, float y, float confidence = 1.0f)
```

 Confidential and Proprietary | Copyright © 2025 – Hailo Technologies Ltd.

Functions

Function	Return Type	Description
`x()`	float	Normalized x position.
`y()`	float	Normalized y position.
`confidence()`	float	The confidence in the point's accuracy, float between 0.0 to 1.0 - default is 1.0.

Enumerations

hailo_object_t

Typing enumeration for *HailoObject* instances.

```
typedef enum
{
  HAILO_ROI,
  HAILO_CLASSIFICATION,
  HAILO_DETECTION,
  HAILO_LANDMARKS,
  HAILO_TILE,
  HAILO_UNIQUE_ID,
  HAILO_MATRIX,
  HAILO_DEPTH_MASK,
  HAILO_CLASS_MASK,
  HAILO_CONF_CLASS_MASK,
  HAILO_USER_META
} hailo_object_t;
```

HailoTensor

`Tensors` are the ouput vector of the network inference. Usually these are N-dimensional matrices that hold little "human readable" value at first, but after a little postprocessing become meaningful objects such as detections or landmarks. All postprocesses start by looking at the output tensors. In fact, usually there will be no need to construct one it will be *provided to your postprocess* via the *hailofilter* element. To make handling these vectors easier, they are provided in a `HailoTensor` class. `Shared pointer handle`: **HailoTensorPtr** `SOURCE`: core/hailo/general/hailo_tensors.hpp

 Confidential and Proprietary | Copyright © 2025 – Hailo Technologies Ltd.

Functions

Function	Return Type	Description
`name()`	std::string	Get the tensor name.
`data()`	uint8_t *	Get the tensor data pointer.
`width()`	uint32_t	Get the tensor width.
`height()`	uint32_t	Get the tensor height.
`features()`	uint32_t	Get the tensor features.
`size()`	uint32_t	Get the tensor total length.
`shape()`	std::vector<std::size_t>	Get the tensor dimensions.
`nms_shape()`	hailo_tensor_nms_shape_t	Get the nms shape.
`quant_info()`	hailo_tensor_quant_info_t	Get the quantization information.
`format()`	hailo_tensor_format_t	Get the tensor format type.
`fix_scale(T num)`	float	Takes a quantized number of template type 'T' and returns its dequantized value ('float').
`quantize(T num)`	T	Takes a dequantized number of template type 'T' and returns its quantized value ('T').
`get(uint row, uint col, uint channel)`	uint8_t	Get the tensor value at this location.
`get_uint16(uint row, uint col, uint channel)`	uint16_t	Get the tensor value (uint16_t) at this location.
`get_full_percision( uint row, uint col, uint channel, bool is_uint16)`	float	Get the tensor dequantized value at this location.

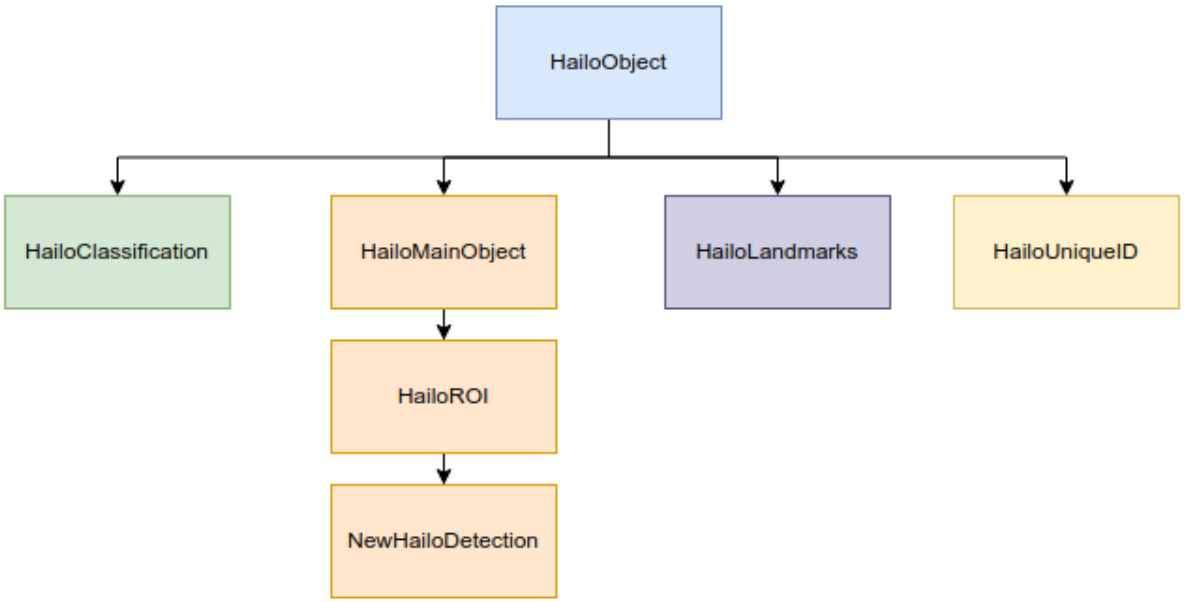
 Confidential and Proprietary | Copyright © 2025 – Hailo Technologies Ltd.

HailoObject

`HailoObject` represents objects that are usable output after postprocessing. They can be detections, classifications, landmarks, or any other similar postprocess results.

This class is an abstraction for other objects to inherit from. To more conveniently compare different types of inheriting classes, `HailoObjects` store their object type from an enumerated list *hailo_object_t*.

The `class inheritance hierarchy` is as follows:



`Shared pointer handle`: **HailoObjectPtr**
`SOURCE`: core/hailo/general/hailo_objects.hpp

Constructor

```
HailoObject()
```

Functions

Function	Return Type	Description
`get_type()`	*hailo_object_t*	The type of the object from the list of enumerated types shown above.

 Confidential and Proprietary | Copyright © 2025 – Hailo Technologies Ltd.

HailoMainObject

Inherits from *HailoObject*

`HailoMainObject` represents a *HailoObject* that can hold other *HailoObject*s. For example a face detection can hold landmarks or age classification, gender classification etc...

`Shared pointer handle`: **HailoMainObjectPtr**

`SOURCE`: core/hailo/general/hailo_objects.hpp

Constructor

```
HailoMainObject()
```

 Confidential and Proprietary | Copyright © 2025 – Hailo Technologies Ltd.

Functions

Function	Return Type	Description
`add_object (HailoObjectPtr obj)`	void	Add a *HailoObject* to this *HailoMainObject*.
`add_tensor (HailoTensorPtr tensor)`	void	Add a *HailoTensor* to this *HailoMainObject*.
`remove_object (HailoObjectPtr obj)`	void	Remove a *HailoObject* from this *HailoMainObject*.
`remove_object(uint index)`	void	Remove a *HailoObject* from this *HailoMainObject* by index.
`remove_objects_typed (hailo_object_t type)`	void	Removes all the objects of a given type, attached to this *HailoMainObject*.
`get_tensor (std::string name)`	*HailoTensorPtr*	Get a tensor from this *HailoMainObject*.
`has_tensors()`	bool	Checks whether there are tensors attached to this *HailoMainObject*.
`get_tensors()`	std::vector <*HailoTensorPtr*>	Get a vector of the tensors attached to this *HailoMainObject*.
`get_tensors_by_name()`	std::map <std::string, *HailoTensorPtr*>	Get the map of tensor names to their corresponding *HailoTensorPtr* objects.
`clear_tensors()`	void	Clear all tensors attached to this *HailoMainObject*.
`get_objects()`	std::vector <*HailoObjectPtr*>	Get the objects attached to this *HailoMainObject*.
`get_objects_typed (hailo_object_t type)`	std::vector <*HailoObjectPtr*>	Get the objects of a given type, attached to this *HailoMainObject*.

 Confidential and Proprietary | Copyright © 2025 – Hailo Technologies Ltd.

HailoROI

Inherits from *HailoMainObject*

`HailoROI` represents an ROI (Region Of Interest): a part of an image that can hold other objects. Mostly inherited by other objects but isn't abstract. Can represent the whole image by giving the right HailoBBox.

`Shared pointer handle`: **HailoROIPtr**

`SOURCE`: core/hailo/general/hailo_objects.hpp

Constructor

```
HailoROI(HailoBBox bbox, std::string stream_id = "")
```

 Confidential and Proprietary | Copyright © 2025 – Hailo Technologies Ltd.

Functions

Function	Return Type	Description
`shared_from_this()`	std::shared_ptr <*HailoROI*>	Get a shared pointer to this instance.
`get_type()`	*hailo_object_t*	This *HailoObject*'s type: HAILO_ROI
`add_object (HailoObjectPtr obj)`	void	Add an object to the main object.
`add_unscaled_object (HailoObjectPtr obj)`	void	Adds an object to the main object. Ignores any potential scaling of ROIs
`get_bbox()`	*HailoBBox*	Get the bbox of this ROI.
`set_bbox(HailoBBox bbox)`	void	Set the bbox of this ROI.
`get_scaling_bbox()`	*HailoBBox*	Get the scaling bbox of this ROI, useful in case of nested ROIs.
`set_scaling_bbox (HailoBBox bbox)`	void	Set the scaling bbox of this ROI, useful in case of nested ROIs.
`clear_scaling_bbox()`	void	Clear the scaling bbox of this ROI.
`get_stream_id()`	std::string	Get the stream ID of this ROI.
`set_stream_id (std::string stream_id)`	std::string	Set the stream ID of this ROI.

 Confidential and Proprietary | Copyright © 2025 – Hailo Technologies Ltd.

HailoDetection

Inherits from *HailoROI*

`HailoDetection` represents a detection in an ROI. It is assumed that all numbers are normalized (between 0 and 1) so that objects remain in relative size for easy image resizing.

`Shared pointer handle`: **HailoDetectionPtr**

SOURCE: core/hailo/general/hailo_objects.hpp

Constructors

```
HailoDetection(HailoBBox bbox, const std::string &label, float confidence)
HailoDetection(HailoBBox bbox, int class_id, const std::string &label, float
↪confidence)
```

Functions

Function	Return Type	Description
`clone()`	std::shared_ptr<*HailoObject*>	Creates a deep copy of this *HailoDetection* object.
`get_type()`	*hailo_object_t*	This *HailoObject*'s type: HAILO_DETECTION
`get_confidence()`	float	This detection's confidence.
`get_label()`	std::string	This detection's label.
`get_class_id()`	int	This detection's class id.
`set_confidence(float conf)`	void	Sets the confidence score for this detection.
`set_label(std::string label)`	void	Sets the label for this detection.
`operator<(const HailoDetection &other)`	bool	Overload < operator, compares confidences.
`operator>(const HailoDetection &other)`	bool	Overload > operator, compares confidences.

 Confidential and Proprietary | Copyright © 2025 – Hailo Technologies Ltd.

HailoClassification

Inherits from *HailoObject*

`HailoClassification` represents a classification of an ROI. Classifications can have different `types`, for example a classification of type 'color' can have a `label` of red or blue.

`Shared pointer handle`: **HailoClassificationPtr**

SOURCE: core/hailo/general/hailo_objects.hpp

Constructors

```
HailoClassification(const std::string &classification_type, const std::string &
↪label, float confidence)
HailoClassification(const std::string &classification_type, int class_id,
↪std::string label, float confidence)
```

Functions

Function	Return Type	Description
`clone()`	std::shared_ptr<*HailoObject*>	Creates a deep copy of this *HailoClassification* object.
`get_type()`	*hailo_object_t*	This *HailoObject*'s type: HAILO_CLASSIFICATION
`get_confidence()`	float	This classification's confidence.
`get_label()`	std::string	This classification's label (e.g. "Horse", "Monkey", "Tiger" for type "Animals").
`get_classification_type()`	std::string	This classification's type (e.g. "age", "gender", "color", etc...).
`get_class_id()`	int	This classification's class id.

HailoLandmarks

Inherits from *HailoObject*

`HailoLandmarks` represents **a set** of landmarks on a given ROI. Like *HailoClassification*, *HailoLandmarks* can also have different `types`, for example a landmark can be of type "pose" or "facial landmarking". Each landmark in the set is represented as a *HailoPoint*.

`Shared pointer handle`: **HailoLandmarksPtr**

SOURCE: core/hailo/general/hailo_objects.hpp

 Confidential and Proprietary | Copyright © 2025 – Hailo Technologies Ltd.

Constructors

```
HailoLandmarks(std::string landmarks_name, float threshold = 0.0f, const std::vector
↪<std::pair<int, int>> pairs = {})
std::string landmarks_name, std::vector<HailoPoint> points, float threshold = 0.0f,
↪const std::vector<std::pair<int, int>> pairs = {})
```

Functions

Function	Return Type	Description
`clone()`	std::shared_ptr<*HailoObject*>	Creates a deep copy of this *HailoLandmarks* object.
`get_type()`	*hailo_object_t*	This *HailoObject*'s type: HAILO_LANDMARKS
`add_point(HailoPoint point)`	void	Add a point to this landmarks object.
`get_points()`	std::vector<*HailoPoint*>	Gets the set of points held by this Landmarks object.
`get_landmarks_type()`	std::string	This landmark's type (e.g. "pose estimation", "face landmark", etc...).
`get_pairs()`	std::vector<std::pair<int, int»	vector of pairs of joints that should be connected in overlay.
`get_threshold()`	float	Gets the threshold held by this Landmarks object.
`set_points (std::vector<HailoPoint> points)`	void	Set a new vector of points to this *HailoLandmarks* object.

HailoUniqueID

Inherits from *HailoObject*

`HailoUniqueID` represents a unique id of an ROI. Sometimes the user may want to give ROIs unique ids (for example, when tracking detections), and having a *HailoObject* abstraction makes adding and removing ids very simple (via `add_object()` and `remove_object()`). If no unique if is provided at construction, then a default -1 is used.

`Shared pointer handle`: **HailoUniqueIDPtr**

SOURCE: core/hailo/general/hailo_objects.hpp

 Confidential and Proprietary | Copyright © 2025 – Hailo Technologies Ltd.

Constructors

```
HailoUniqueID()
HailoUniqueID(int unique_id)
```

Functions

Function	Return Type	Description
`clone()`	std::shared_ptr<*HailoObject*>	Creates a deep copy of this *HailoUniqueID* object.
`get_type()`	*hailo_object_t*	This *HailoObject*'s type: HAILO_UNIQUE_ID
`get_id()`	int	Get the unique id.
`get_mode()`	int	Get the mode.

HailoMask

Inherits from *HailoObject*

`HailoMask` represents a mask of an ROI. Whenever the output of a postprocess is masks (tensors with result for every pixel) we will ROIs mask objects.

`Shared pointer handle`: **HailoMaskPtr**

`SOURCE`: core/hailo/general/hailo_objects.hpp

Constructors

```
HailoMask(int mask_width, int mask_height, float transparency)
```

Functions

Function	Return Type	Description
`get_width()`	int	get the mask width
`get_height()`	int	get the mask height
`get_transparency()`	float	get the desired drawing transparency

 Confidential and Proprietary | Copyright © 2025 – Hailo Technologies Ltd.

HailoDepthMask

Inherits from *HailoMask*.

`HailoDepthMask` represents a mask of an ROI, with float values for each pixel. The values represent depth between minimum and maximum values.

`Shared pointer handle`: **HailoDepthMaskPtr**

`SOURCE`: core/hailo/general/hailo_objects.hpp

Constructors

```
HailoDepthMask(std::vector<float> &&data_vec, int mask_width, int mask_height,
↪float transparency)
```

Functions

Function	Return Type	Description
`get_type()`	*hailo_object_t*	This *HailoObject*'s type: HAILO_DEPTH_MASK
`get_data()`	const std::vector <*float*>	get the mask data vector

HailoClassMask

Inherits from *HailoMask*

`HailoClassMask` represents a mask of an ROI, with uint8_t class id classification for each pixel.

`Shared pointer handle`: **HailoClassMaskPtr**

`SOURCE`: core/hailo/general/hailo_objects.hpp

Constructors

```
HailoClassMask(std::vector<uint8_t> &&data_vec, int mask_width, int mask_height,
↪float transparency)
```

Functions

Function	Return Type	Description
`get_type()`	*hailo_object_t*	This *HailoObject*'s type: HAILO_CLASS_MASK
`get_data()`	const std::vector<*uint8_t*>	get the mask data vector

 Confidential and Proprietary | Copyright © 2025 – Hailo Technologies Ltd.

HailoConfClassMask

Inherits from *HailoMask*

`HailoConfClassMask` represents a mask of an ROI, contains mask-class-id and confidence float value for each pixel.

`Shared pointer handle`: **HailoConfClassMaskPtr**

SOURCE: core/hailo/general/hailo_objects.hpp

Constructors

```
HailoConfClassMask(std::vector<float> &&data_vec, int mask_width, int mask_height,
↪float transparency, int class_id)
```

Functions

Function	Return Type	Description
`get_type()`	*hailo_object_t*	This *HailoObject*'s type: HAILO_CONF_CLASS_MASK
`get_data()`	const std::vector<*float*>	get the mask data vector
`get_class_id()`	int	get the mask class id

HailoMatrix

Inherits from *HailoObject*

`HailoMatrix` represents a matrix, contains float values. This matrix can be added to any HailoObject for different use cases.

`Shared pointer handle`: **HailoMatrixPtr**

SOURCE: core/hailo/general/hailo_objects.hpp

Constructors

```
HailoMatrix(float *data_ptr, uint32_t mat_height, uint32_t mat_width, uint32_t mat_
↪features = HailoMatrix::DEFAULT_NUMBER_OF_FEATURES)
```

 Confidential and Proprietary | Copyright © 2025 – Hailo Technologies Ltd.

Functions

Function	Return Type	Description
`clone()`	std::shared_ptr<*HailoObject*>	Creates a deep copy of this *HailoMatrix*object.
`get_type()`	*hailo_object_t*	This *HailoObject*'s type: HAILO_MATRIX
`width()`	const uint32_t	get matrix width
`height()`	const uint32_t	get matrix height
`features()`	const uint32_t	get matrix number of features
`size()`	const uint32_t	get number of elements in matrix
`shape()`	std::vectorstd::size_t	get the shape of the matrix
`get_data()`	std::vector<float>	get the constant reference to the matrix data

HailoUserMeta

Inherits from *HailoObject*
`HailoUserMeta` represents a sample metadata for users.
`Shared pointer handle`: **HailoUserMetaPtr**
`SOURCE`: core/hailo/general/hailo_objects.hpp

Constructors

```
HailoUserMeta()
HailoUserMeta(int user_int, std::string user_string, float user_float) : m_user_
↪int(user_int), m_user_string(user_string), m_user_float(user_float)
```

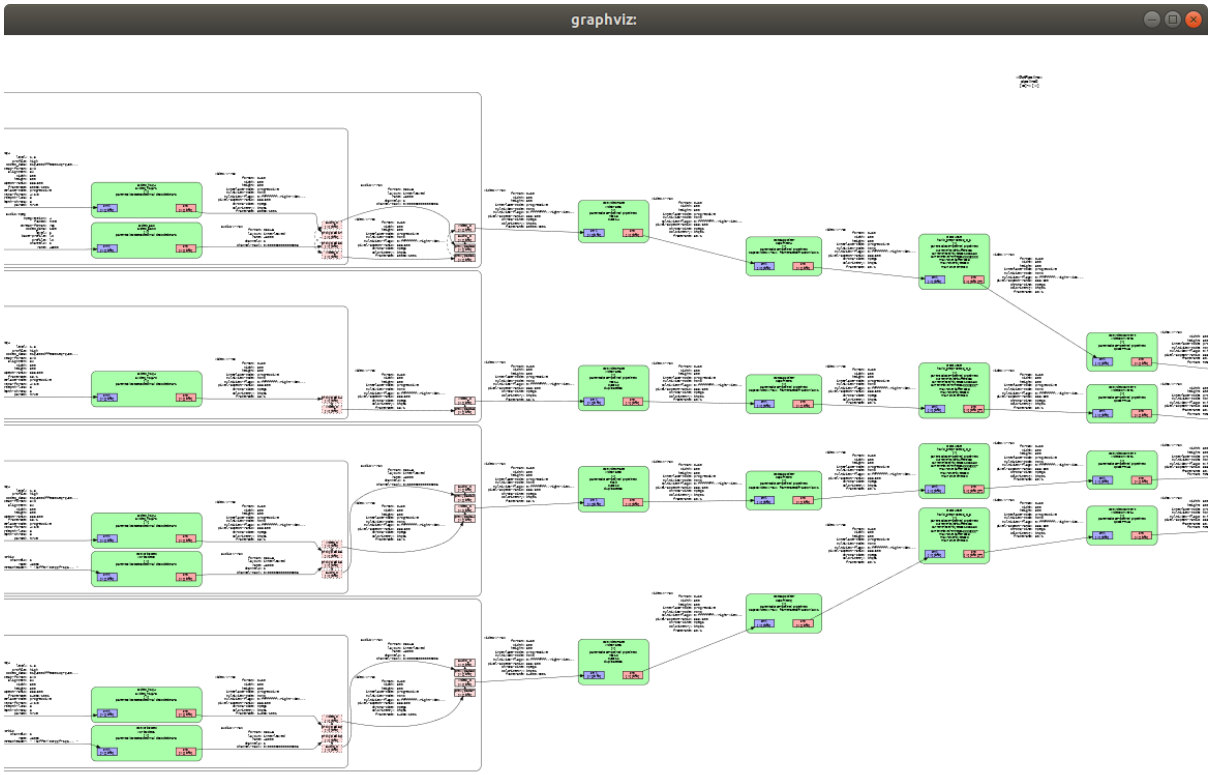
Functions

Function	Return Type	Description
`get_type()`	*hailo_object_t*	This *HailoObject*'s type: HAILO_USER_META
`get_user_float()`	folat	get the user-defined floating-point value in a thread-safe manner.
`get_user_string()`	std::string	get the user-defined string in a thread-safe manner.
`get_user_int()`	int	get the user-defined integer value in a thread-safe manner
`set_user_float(float user_float)`	void	set the user-defined floating-point value in a thread-safe manner.
`set_user_string(std::string user_string)`	void	set the user-defined string in a thread-safe manner.
`set_user_int(int user_int)`	void	set the user-defined integer value in a thread-safe manner.

 Confidential and Proprietary | Copyright © 2025 – Hailo Technologies Ltd.

4.3.3. Debugging

GstShark



GstShark is an open-source project from RidgeRun that provides benchmarks and profiling tools for GStreamer 1.7.1 (and above). It includes tracers for generating debug information plus some tools to analyze the debug information. GstShark provides easy to use and useful tracers, paired with analysis tools to enable straightforward optimizations.

GstShark leverages GStreamer's built in tracing system and adds plotting tools to simplify the process of understanding the bottlenecks in the user's pipeline.

As part of the TAPPAS framework, we have expanded the available profiling mechanisms to include TAPPAS related tracers and a new plotting tool.

The profiling tool provides 3 general features that can be used to debug the pipeline:

- Tracers log printouts - At the most basic level, printouts are available from the traces of the different measurements made. An experienced user will be able to make observations at runtime. Printouts can be found in the debug file: GST_DEBUG_FILE=$TAPPAS_WORKSPACE/tappas_traces.log.
- Graphic visualization - Shown above, gst-shark can generate a pipeline graph that shows how elements are connected and what caps were negotiated between them. This is a very convenient feature for observing the pipeline in a more comfortable way. The graph is generated at runtime so it is very effective for seeing and debugging how elements were actually connected and in what formats the data ended up in.
- Plotting tool - A python script that generates a graph plot for each tracer metric enabled. This is a powerful tool to visualize each metric that can be used for deeper debugging.

 Confidential and Proprietary | Copyright © 2025 – Hailo Technologies Ltd.

Install

Follow RidgeRun tutorial: GstShark

Bash Shortcuts

Please refer to the following section: *Using GstShark in yocto-compiled images*

```
vim ~/.bashrc
```

```
# set gstreamer debug
gst_set_debug() {
 export HAILO_PROFILE_LOCATION=/tmp/profile
 export GST_DEBUG="GST_TRACER:7"
 export GST_DEBUG_FILE=$TAPPAS_WORKSPACE/tappas_traces.log
 export GST_TRACERS="cpuusage;proctime;interlatency;scheduletime;bitrate;
↪framerate;queuelevel;threadmonitor;numerator;buffer;detections;graphic"
 export GST_DEBUG_NO_COLOR=1
 echo 'Options for TRACERS:"'
 echo 'export GST_TRACERS="cpuusage;proctime;interlatency;scheduletime;bitrate;
↪framerate;queuelevel;threadmonitor;numerator;buffer;detections;graphic"'
}

# set trace to collect graphic data of gstreamer pipeline
gst_set_graphic() {
 export HAILO_PROFILE_LOCATION=/tmp/profile
 export GST_DEBUG_DUMP_DOT_DIR=/tmp/
 export GST_DEBUG="GST_TRACER:7"
 export GST_TRACERS="graphic"
 echo 'export GST_TRACERS="graphic"'
}

# unset gstreamer debug
gst_unset_debug() {
 unset GST_TRACERS
 unset GST_DEBUG
 unset GST_DEBUG_DUMP_DOT_DIR
 unset HAILO_PROFILE_LOCATION
 unset GST_DEBUG_FILE
 unset GST_DEBUG_NO_COLOR
}

# plot the gst-shark dump files
gst_plot_debug() {
 export HAILO_PROFILE_LOCATION=/tmp/profile
 split_traces_dir=$TAPPAS_WORKSPACE/tappas_traces_$(date +%d.%m.%Y_%H:%M:%S)
 $TAPPAS_WORKSPACE/sources/gst-shark/scripts/graphics/split_traces.sh $split_
↪traces_dir
 python3 $TAPPAS_WORKSPACE/sources/gst-shark/scripts/graphics/plot_all_to_html.
↪py -p $split_traces_dir
 echo 'In order to plot the graphic pipeline graph, run:'
 echo "dot $HAILO_PROFILE_LOCATION/graphic/pipeline_<timestamp>.dot -T x11"
}
```

Note that 4 functions were added: two sets, an unset, and a plot function. The set functions enable gst-shark by setting environment variables, the chief of which is GST_TRACERS. This enables the different trace hooks in the pipeline. The available tracers are listed in the echo command at the end of each set. You can enable any combination of the available tracers, just chain them together with a ';' (notice that the difference between gst_set_debug and gst_set_graphic is that gst_set_debug enables all tracers whereas gst_set_graphic only enables the graphic tracer that draws the pipeline graph). HAILO_PROFILE_LOCATION and GST_DEBUG_DUMP_DOT_DIR set locations where the dump files are stored,

 Confidential and Proprietary | Copyright © 2025 – Hailo Technologies Ltd.

the first sets where the tracer dumps are (used for gst-plot), and the latter where the dot file is saved (the graphic pipeline graph). Unset disables all tracers, and gst_plot_debug runs plot script.

Using GstShark in Yocto-Compiled Images

Enable TAPPAS tracers:

- Export the following environment variables:

```
export HAILO_PROFILE_LOCATION=/tmp/profile
export GST_DEBUG="GST_TRACER:7"
export GST_DEBUG_NO_COLOR=1
```

- Select the tracers by setting the GST_TRACERS environment variable to the list of tracers, separated by ; as in the example:

```
export GST_TRACERS="scheduletime;bitrate;threadmonitor;numerator;buffer;
↪detections"
```

- You should export only some of the tracers, exporting too many tracers may fail the embedded device. All tracers explained: *Understanding GstShark tappas plotted graphs*
- If you want to save the output to a file, run the following command before running the app:

```
export GST_DEBUG_FILE=<file_path>
```

- If you want to plot the tracers output, use a strong machine (not an embedded device) with a full tappas installation and copy there the output file, then run the following commands:

```
export GST_DEBUG_FILE=<file_path>
gst_plot_debug
```

Using GstShark

Let's say there is a GStreamer app to be profiled, Start by enabling gst-shark:

Then just run your app, it will be possible to see all kinds of tracer prints on the debug output file: GST_DEBUG_FILE=$TAPPAS_WORKSPACE/tappas_traces.log.

After you have run a gstreamer pipeline with tracers enabled, plot them using the plot script, just run:

```
gst_plot_debug
```

It will print to the console the path of the html file that contains the plots, which can be opened in the browser. In addition it will print the command to open the pipeline graph, which can be run in a terminal to open the graph.

Understanding GstShark TAPPAS Plotted Graphs

Each graph inspects a different metric of the pipeline, it is recommended to read more about what each one represents here:

- CPU Usage (cpuusage) - Measures the CPU usage every second. In multiprocessor systems this measurements are presented per core.
- Processing Time (proctime) - Measures the time an element takes to produce an output given the corresponding input.
- InterLatency (interlatency) - Measures the latency time at different points in the pipeline.
- Schedule Time (scheduling) - Measures the amount of time between two consecutive buffers in a sink pad.
- Buffer (buffer) - Prints information of every buffer that passes through every sink pad in the pipeline.

 Confidential and Proprietary | Copyright © 2025 – Hailo Technologies Ltd.

- Bitrate (bitrate) - Measures the current stream bitrate in bits per second.
- Framerate (framerate) - Measures the amount of frames that go through a src pad every second.
- Queue Level (queuelevel) - Measures the amount of data queued in every queue element in the pipeline.
- Thread Monitor (threadmonitor) - Measures the CPU usage of every thread in the pipeline.
- Numerator (numerator) - Numerates the buffers by setting the field "offset" of the buffer metadata. This trace is different from the others because it does not collect any data, it just numerates the buffers.
- Detections (detections) - Prints information about the objects detected in every buffer that passes through every pad in the pipeline. This trace only works with the TAPPAS framework since it collects the TAPPAS detection objects.
- Graphic (graphics) - Records a graphical representation of the current pipeline.

Note: When using the Thread Monitor tracer, provide meaningful names to the queues because the names of the threads in the graph will be based on the names of the queues. This will help easily identify the threads and understand their purpose when analyzing the trace. In addition, due to the way this tracer works, it is important to keep the names of the queues shorter than 16 characters. If the names are longer than this, the thread names in the graph will be truncated.

Modify Buffering Mode and Size

```
$ export GST_SHARK_FILE_BUFFERING=0
```

With the no buffering mode every I/O operation is written as soon as possible.

The following command is an example of how to define the environment variable that will change the buffering mode to full buffering and the buffering size, this command uses a positive integer value for the size:

```
$ export GST_SHARK_FILE_BUFFERING=1024
```

Individual Element Tracing (filter)

The individual element tracing, or filter parameter, allows the user to choose which elements get included in the tracing. The value to be set in the filter is a Glib Compatible Regular Expression, meaning that elements to be traced can be grouped by using a regex that matches with their name.

The filtering applies to the element name, NOT the factory. This is, if the element is specified as "identity name=myelem", it should be referred to as "myelem" and not to "identity"

Print the amount of frames that flow every 5 seconds through the different src pads in the pipe:

```
GST_TRACERS="framerate(period=5)" GST_DEBUG=GST_TRACER:7
```

Print the amount of bits that flow every 3 seconds through the different src pads in the pipe:

```
GST_TRACERS="bitrate(period=3)" GST_DEBUG=GST_TRACER:7
```

Print the amount of frames that flow every 5 seconds and bits that flow every 3 seconds through the different src pads in the pipe:

```
GST_TRACERS="framerate(period=5);bitrate(period=3)" GST_DEBUG=GST_TRACER:7
```

Print the amount of frames that flow every 5 through the identity:

```
GST_TRACERS="framerate(period=5,filter=identity);bitrate(period=3)" GST_
↪DEBUG=GST_TRACER:7
```

 Confidential and Proprietary | Copyright © 2025 – Hailo Technologies Ltd.

Using gst-instruments

gst-instruments is a set of performance profiling and data flow inspection tools for GStreamer pipelines.

- `gst-top-1.0` at the start of the pipeline will analyze and profile the run. (gst-top-1.0 gst-launch-1.0 ! audiotestsrc ! autovideosink)
- `gst-report-1.0` - generates performance report for input trace file.
- `gst-report-1.0 --dot gst-top.gsttracee | dot -Tsvg > perf.svg` - generates performance graph in DOT format.

Read more in gst-instruments github page

4.3.4. Writing Your Own Postprocess

Overview

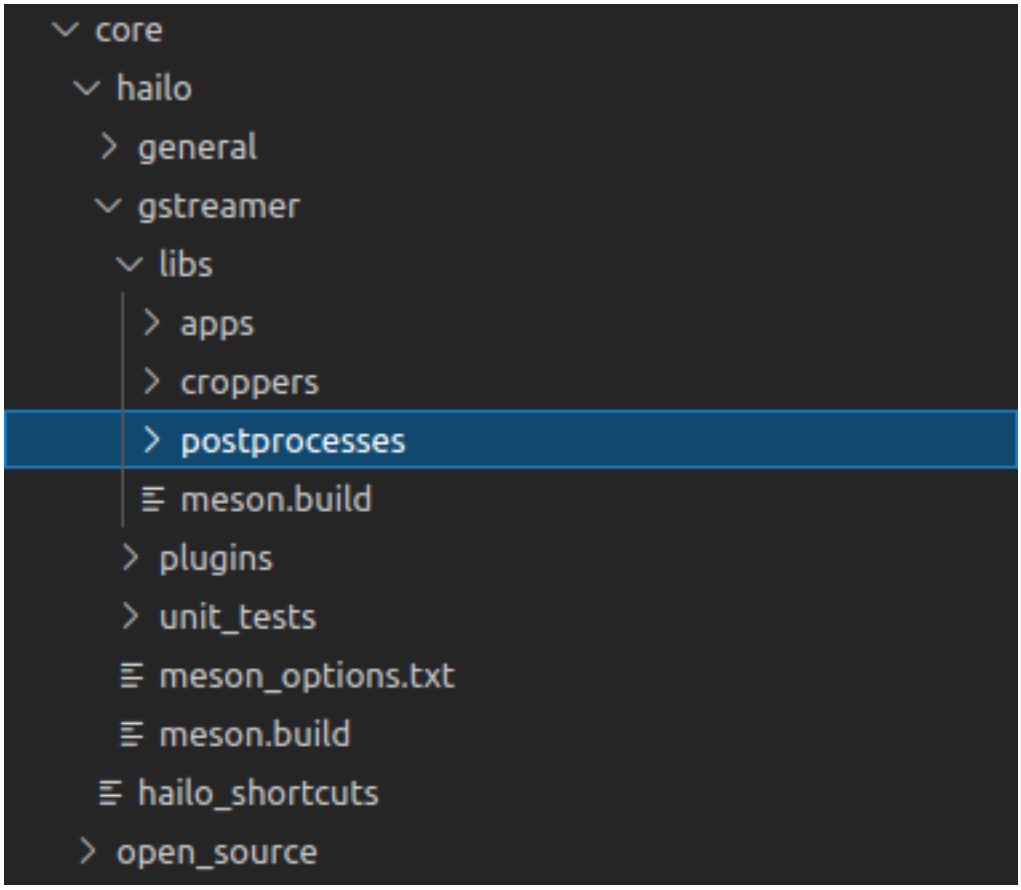
When adding a network to the TAPPAS that is not already supported, then most likely it will need to implement a new post-process and drawing filter. Fortunately with the use of the *hailofilter*, there is no need to create any new GStreamer elements, just provide the .so (compiled shared object binary) that applies to the filter! This guide will go over how to create such an .so and what mechanisms/structures are available to the user as they create their postprocess.

Getting Started

File Location

When creating or working with postprocess, it is important to know where to find all the relevant source files that already exist, and how to add new ones. From the TAPPAS home directory, one can find the `core/` folder. Inside this `core/` directory are a few subdirectories that host different types of source files. The `open_source` folder contains source files from 3rd party libraries (opencv, xtensor, etc..), while the `hailo` folder contains source files for all kinds of Hailo tools, such as the Hailo Gstreamer elements, the different metas provided, and the source files for the postprocesses of the networks that were already provided in the TAPPAS. Inside this directory is one titled `general/`, which contains sources for *the different object classes* (detections, classifications, etc..) available. Next to `general` is a directory titled `gstreamer/`, and inside that are two folders of interest: `libs/` and `plugins/`. The former contains the source code for all the postprocess and drawing functions packaged in the TAPPAS, while the latter contains source code for the different Hailo GStreamer elements, and the different metas available. This guide will mostly focus on this `core/hailo/` directory, as it has everything needed to create and compile a new .so! It is recommended that users spend time familiarizing themselves with these locations, and then when ready to continue enter the `postprocesses/` directory:

 Confidential and Proprietary | Copyright © 2025 – Hailo Technologies Ltd.



Preparing the Header File: Default Filter Function

The new postprocess can be created here in the `postprocesses/` folder. Create a new header file named `my_post.hpp`.In the first lines we want to import useful classes to our postprocess, so add the following includes:

```
#pragma once
#include "hailo_objects.hpp"
#include "hailo_common.hpp"
```

`"hailo_objects.hpp"` contains classes that represent the *different inputs (tensors) and outputs* (detections, classifications, etc...) that your postprocess might handle. You can find the header in core/hailo/general/hailo_objects.hpp. Your main point of entry for data in the postprocess is the `HailoROI`, which can have a tensor or a number of tensors attached. `"hailo_common.hpp"` provides common useful functions for handling these classes. Wrap up the header file by adding a function prototype for your filter, the whole header file should look like:

```
#pragma once
#include "hailo_objects.hpp"
#include "hailo_common.hpp"

__BEGIN_DECLS
void filter(HailoROIPtr roi);
__END_DECLS
```

The process is now complete. The `hailofilter` element does not expect much, just that the above `filter` function be provided. Adding *Multiple Filters in One .so* will be described later. Note that the `filter` function takes a `HailoROIPtr` as a parameter; this will provide you with the `HailoROI` of each passing image.

 Confidential and Proprietary | Copyright © 2025 – Hailo Technologies Ltd.

Implementing a Filter()

Start implementing the actual filter to see how to access and work with tensors. Start by creating a new file called `my_post.cpp`. Open it and include the following:

```
#include <iostream>
#include "my_post.hpp"
```

The `<iostream>` will allow printing to the console, and the `"my_post.hpp"` includes the header file we just wrote.
For now add the following implementation for the filter() so that we have a working postprocess we can test:

```
// Default filter function
void filter(HailoROIPtr roi)
{
 std::cout << "My first postprocess!" << std::endl;
}
```

This is sufficient for compiling and running a pipeline. Next we will describe how to add the postprocess to the meson project so that it compiles.

Compiling and Running

Building with Meson

Meson is an open source build system that places an emphasis on speed and ease of use. GStreamer uses meson for all sub-projects to generate build instructions to be executed by ninja, another build system focuses specifically on speed that requires a higher level build system (ie: meson) to generate its input files. Like GStreamer, TAPPAS also uses meson, and compiling new projects requires adjusting the `meson.build` files. Here we will describe how to add the user's. In the `libs/postprocesses` path will be found a meson.build, open it and add the following entry for our postprocess:

```
################################################
# MY POST SOURCES
################################################
my_post_sources = [
 'my_post.cpp',
]

my_post_lib = shared_library('my_post',
 my_post_sources,
 include_directories: [hailo_general_inc] + xtensor_inc,
 dependencies : post_deps,
 gnu_symbol_visibility : 'default',
 install: true,
 install_dir: post_proc_install_dir,
)
```

This will provide meson with all the information required to compile the postprocess. In short, we are providing paths to cpp compilers, linked libraries, included directories and dependencies. All these path variables come from the parent meson project, the meson file can be read to see what packages and directories are available at core/hailo/meson.build.

 Confidential and Proprietary | Copyright © 2025 – Hailo Technologies Ltd.

Compiling the .so

We are now ready to compile the postprocess. To help streamline this process a script has been provided that will handle most of the work. The script can be found at scripts/gstreamer/install_hailo_gstreamer.sh. It includes some flags that allows the user to do more specific operations, but they are not needed right now.

From the TAPPAS home directory folder you can run:

```
./scripts/gstreamer/install_hailo_gstreamer.sh
```

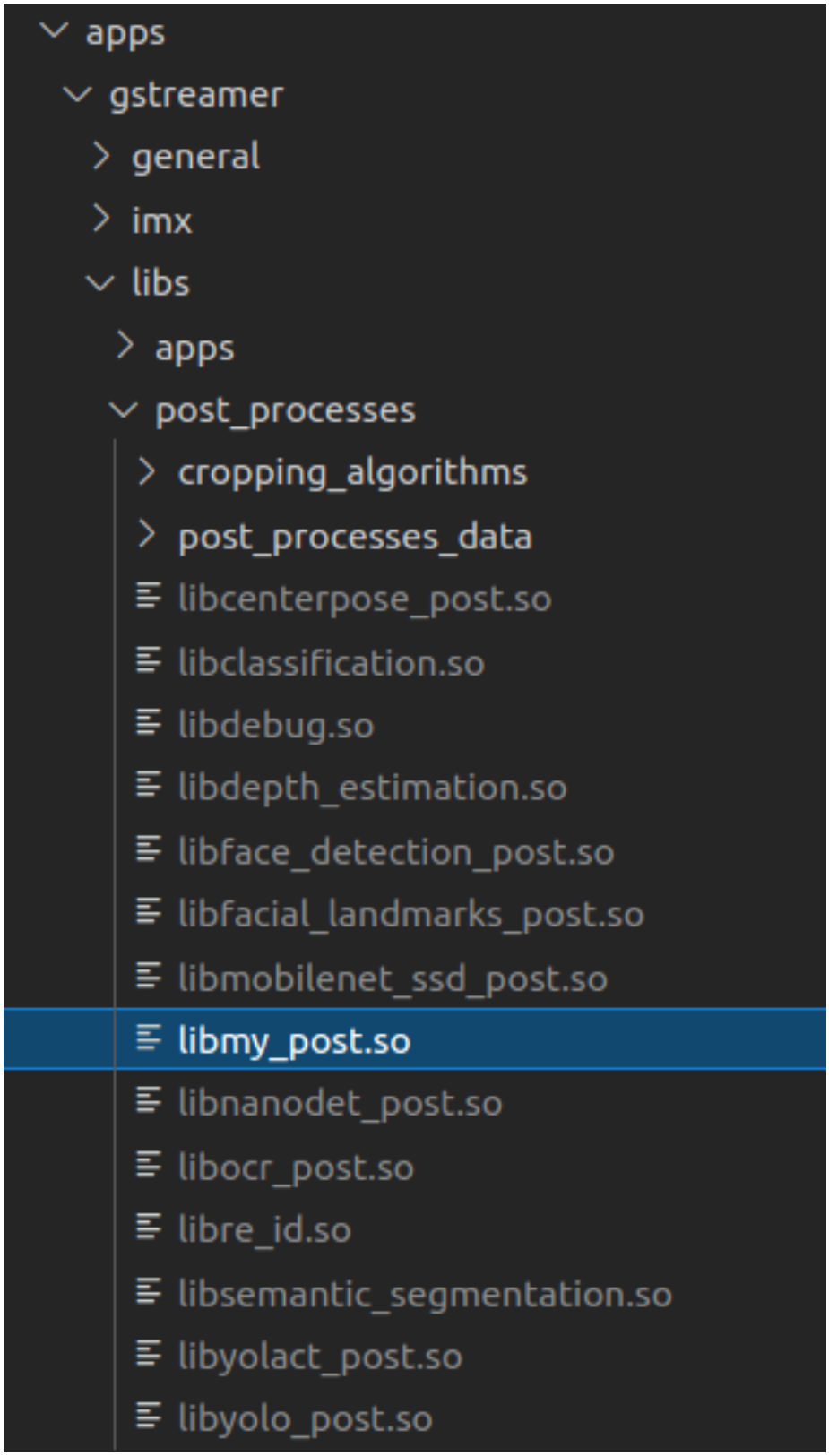
```
hailo@hailo_tappas:/local/workspace/tappas$ ./scripts/gstreamer/install_hailo_gstreamer.sh
/local/workspace/tappas/core/hailo/gstreamer /local/workspace/tappas
Directory already configured.

Just run your build command (e.g. ninja) and Meson will regenerate as necessary.
If ninja fails, run "ninja reconfigure" or "meson --reconfigure"
to force Meson to regenerate.

If build failures persist, run "meson setup --wipe" to rebuild from scratch
using the same options as passed when configuring the build.
To change option values, run "meson configure" instead.
ninja: Entering directory `build.release'
[0/1] Regenerating build files.
The Meson build system
Version: 0.58.0
Source dir: /local/workspace/tappas/core/hailo/gstreamer
Build dir: /local/workspace/tappas/core/hailo/gstreamer/build.release
Build type: native build
Project name: gst-hailo-tools
Project version: 3.16.0
C compiler for the host machine: gcc-9 (gcc 9.4.0 "gcc-9 (Ubuntu 9.4.0-1ubuntu1~18.04) 9.4.0")
C linker for the host machine: gcc-9 ld.bfd 2.30
C++ compiler for the host machine: g++-9 (gcc 9.4.0 "g++-9 (Ubuntu 9.4.0-1ubuntu1~18.04) 9.4.0")
C++ linker for the host machine: g++-9 ld.bfd 2.30
Host machine cpu family: x86_64
Host machine cpu: x86_64
Configuring config.h using configuration
Message: GCC >= 9.0.0 detected, applying extra arguments.
Dependency gstreamer-1.0 found: YES 1.14.5 (cached)
Dependency gstreamer-base-1.0 found: YES 1.14.5 (cached)
Dependency gstreamer-app-1.0 found: YES 1.14.5 (cached)
Dependency gstreamer-video-1.0 found: YES 1.14.5 (cached)
Dependency opencv4 found: YES 4.5.2 (cached)
Dependency glib-2.0 found: YES 2.56.4 (cached)
Dependency gmodule-2.0 found: YES 2.56.4 (cached)
Dependency gobject-2.0 found: YES 2.56.4 (cached)
Dependency pygobject-3.0 found: YES 3.26.1 (cached)
Program python3.6 found: YES (/usr/bin/python3.6)
Found pkg-config: /usr/bin/pkg-config (0.29.1)
Dependency python found: YES (pkgconfig)
Dependency python found: YES (pkgconfig)
Dependency blas found: YES 3.7.1 (cached)
Dependency lapack found: YES 3.7.1 (cached)
Library dl found: YES
Library stdc++fs found: YES
Dependency threads found: YES unknown (cached)
Build targets in project: 29
```

If all runs correctly a green `YES`, and our .so should appear in `apps/libs/post_processes/`!

 Confidential and Proprietary | Copyright © 2025 – Hailo Technologies Ltd.



 Confidential and Proprietary | Copyright © 2025 – Hailo Technologies Ltd.

Running the .so

Now that the user has successfully compiled their first postprocess, they can continue to run the postprocess and view the results. Since it is still generic, run this test pipeline in the terminal to see if it works:

```
gst-launch-1.0 videotestsrc ! hailofilter so-path=$TAPPAS_WORKSPACE/apps/libs/post_
↪processes/libmy_post.so ! fakesink
```

Note in the above pipeline that we gave the `hailofilter` the path to `libmy_post.so` in the `so-path` property. So now every time a buffer is received in that `hailofilter`'s sink pad, it calls the `filter()` function in `libmy_post.so`. The resulting app should print our chosen text `"My first postprocess!"` in the console:

```
hailo@hailo_tappas:/local/workspace/tappas$ gst-launch-1.0 videotestsrc ! hailofilter2 so-path=$TAPPAS_WORKSPACE/apps/gstreamer/x86/libs/libmy_post.so ! fakesink
Setting pipeline to PAUSED ...
Pipeline is PREROLLING ...
My first postprocess!
Pipeline is PREROLLED ...
Setting pipeline to PLAYING ...
New clock: GstSystemClock
My first postprocess!
My first postprocess!
My first postprocess!
My first postprocess!
My first postprocess!
My first postprocess!
My first postprocess!
```

Filter Basics

Working with Tensors

Printing statements on every buffer is useful, however we would like a postprocess that can actually do operations on inference tensors. We will now describe how this can be achieved. Go back to `my_post.cpp` and replace the print statement with the following:

```
// Get the output layers from the hailo frame.
std::vector<HailoTensorPtr> tensors = roi->get_tensors();
```

The `HailoROI` has two ways of providing the output tensors of a network: via the `get_tensors()` and `get_tensor(std::string name)` functions. The first (which is used here) returns an `std::vector` of `HailoTensorPtr` objects. These are an `std::shared_ptr` to a `HailoTensor`: a class that represents an output tensor of a network. `HailoTensor` holds all kinds of important tensor metadata besides the data itself; such as the width, height, number of channels, and even quantization parameters. A full implementation for this class can be viewed at core/hailo/general/hailo_tensors.hpp. `get_tensor(std::string name)` also returns a `HailoTensorPtr`, but only the one with the given name output layer name. This can be convenient for performing operations on specific layers whose names are known in advance. Now that we have a vector of `HailoTensorPtr` objects, lets examine the information that can be obtained from it. Add the following lines to our `filter()` function:

```
// Get the first output tensor
HailoTensorPtr first_tensor = tensors[0];
std::cout << "Tensor: " << first_tensor->name();
std::cout << " has width: " << first_tensor->shape()[0];
std::cout << " height: " << first_tensor->shape()[1];
std::cout << " channels: " << first_tensor->shape()[2] << std::endl;
```

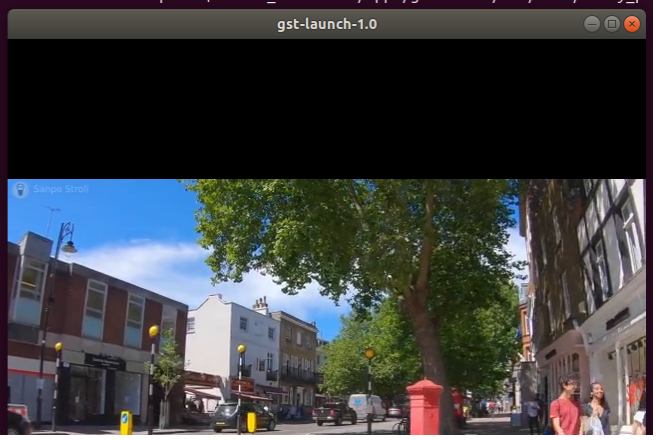
Recompile with the same *script we used earlier*. Run a test pipeline, and this time see actual parameters of the tensor printed out:

```
gst-launch-1.0 filesrc location=$TAPPAS_WORKSPACE/apps/detection/resources/
↪detection.mp4 name=src_0 ! decodebin ! videoscale ! video/x-raw, pixel-aspect-
↪ratio=1/1 ! videoconvert ! queue ! hailonet hef-path=$TAPPAS_WORKSPACE/apps/
↪detection/resources/yolov5m_wo_spp.hef is-active=true ! queue leaky=no max-size-
↪buffers=30 max-size-bytes=0 max-size-time=0 ! hailofilter so-path=$TAPPAS_
↪WORKSPACE/apps/libs/post_processes/libmy_post.so qos=false ! videoconvert !
↪fpsdisplaysink video-sink=ximagesink name=hailo_display sync=true text-
↪overlay=false
```

(continues on next page)

(continued from previous page)

```
hailo@hailo_tappas:/local/workspace/tappas$ gst-launch-1.0 filesrc location=$TAPPAS_WORKSPACE/apps/gstreamer/x86/detection/resources/d
aspect-ratio=1/1 ! videoconvert ! queue ! hailonet hef-path=$TAPPAS_WORKSPACE/apps/gstreamer/x86/detection/resources/yolov5m.hef debug
eaky=no max-size-buffers=30 max-size-bytes=0 max-size-time=0 ! hailofilter2 so-path=$TAPPAS_WORKSPACE/apps/gstreamer/x86/libs/libmy_po
name=hailo_display sync=true text-overlay=false
Setting pipeline to PAUSED ...
Pipeline is PREROLLING ...
Redistribute latency...
Redistribute latency...
Tensor: yolov5m/conv73 has width: 80 height: 80 channels: 255
Pipeline is PREROLLED ...
Setting pipeline to PLAYING ...
New clock: GstSystemClock
Tensor: yolov5m/conv73 has width: 80 height: 80 channels: 255
Tensor: yolov5m/conv73 has width: 80 height: 80 channels: 255
Tensor: yolov5m/conv73 has width: 80 height: 80 channels: 255
Tensor: yolov5m/conv73 has width: 80 height: 80 channels: 255
Tensor: yolov5m/conv73 has width: 80 height: 80 channels: 255
Tensor: yolov5m/conv73 has width: 80 height: 80 channels: 255
Tensor: yolov5m/conv73 has width: 80 height: 80 channels: 255
Tensor: yolov5m/conv73 has width: 80 height: 80 channels: 255
Tensor: yolov5m/conv73 has width: 80 height: 80 channels: 255
Tensor: yolov5m/conv73 has width: 80 height: 80 channels: 255
Tensor: yolov5m/conv73 has width: 80 height: 80 channels: 255
Tensor: yolov5m/conv73 has width: 80 height: 80 channels: 255
Tensor: yolov5m/conv73 has width: 80 height: 80 channels: 255
Tensor: yolov5m/conv73 has width: 80 height: 80 channels: 255
Tensor: yolov5m/conv73 has width: 80 height: 80 channels: 255
```



With a `HailoTensorPtr` the user has everything needed to perform postprocess operations. The actual tensor values can be accessed from the `HailoTensorPtr` with:

```
auto first_tensor_data = first_tensor->data();
```

Remember at this point the data is of type `uint8_t`, for full precision you will have to dequantize the tensor to a `float`. To aid this the quantization parameters (scale and zero point) are stored in the `HailoTensorPtr` and can be applied through `tensor->fix_scale(uint8_t num)`.

Attaching Detection Objects to the Frame

Now that you know how to create a basic filter and access your inference tensor, we will learn how to add a detection object to the `hailo_frame`.Remove the prints from the `filter()` function and replace them with the following function call:

```
std::vector<HailoDetectionPtr> detections = demo_detection_objects();
```

Here the function is being called `demo_detection_objects()` which will return some detection objects. Copy the following function definition into `my_post.cpp`:

```
std::vector<HailoDetection> demo_detection_objects()
{
 std::vector<HailoDetection> objects; // The detection objects we will eventually
↪return
 HailoDetection first_detection = HailoDetection(HailoBBox(0.2, 0.2, 0.2, 0.2),
↪"person", 0.99);
 HailoDetection second_detection = HailoDetection(HailoBBox(0.6, 0.6, 0.2, 0.2),
↪"person", 0.89);
 objects.push_back(first_detection);
 objects.push_back(second_detection);

 return objects;
}
```

In this function two instances of `HailoDetection` are being created and pushed into a vector that we return. Note that when creating a `HailoDetection`, we give a series of parameters. The expected parameters are as follows:

```
HailoDetection(HailoBBox bbox, const std::string &label, float confidence)
```

 Confidential and Proprietary | Copyright © 2025 – Hailo Technologies Ltd.

Where `HailoBBox` is a class that represents a bounding box, it is initialized as `HailoBBox(float xmin, float ymin, float width, float height)`.
NOTE: It is assumed that the `xmin, ymin, width, and height` given are a **percentage of the image size** (meaning, if the box is **half** as wide as the width of the image, then `width=0.5`). This protects the pipeline's ability to resize buffers without compromising the correct relative size of the detection boxes.

Looking back at the demo function we just introduced, we are adding two instances of `HailoDetection`: `first_detection` and `second_detection`. According to the parameters we saw, `first_detection` has an `xmin` 20% along the x axis, and a `ymin` 20% down the y axis. The `width` and `height` are also 20% of the image. The last two parameters, `label` and `confidence`, show that this instance has a 99% `confidence` for `label` person.

Now that we have a couple of `HailoDetections` available, add to them the original `HailoROIPtr`. There is a helper function we need in the core/hailo/general/hailo_common.hpp file that we included earlier in `my_post.hpp`.
This file has other features that will be useful, so it is recommended to keep the file readily available.
With the include in place, add the following function call to the end of the `filter()` function:

```
// Update the frame with the found detections.
hailo_common::add_detections(roi, detections);
```

This function takes a `HailoROIPtr` and a `HailoDetection` vector, then adds each `HailoDetection` to the `HailoROIPtr`. Now that our detections have been added to the `hailo_frame` our postprocess is done!
To recap, our whole `my_post.cpp` should look like this:

```
#include <iostream>
#include "my_post.hpp"

std::vector<HailoDetection> demo_detection_objects()
{
 std::vector<HailoDetection> objects; // The detection objects we will eventually
↪return
 HailoDetection first_detection = HailoDetection(HailoBBox(0.2, 0.2, 0.2, 0.2),
↪"person", 0.99);
 HailoDetection second_detection = HailoDetection(HailoBBox(0.6, 0.6, 0.2, 0.2),
↪"person", 0.89);
 objects.push_back(first_detection);
 objects.push_back(second_detection);
 return objects;
}

// Default filter function
void filter(HailoROIPtr roi)
{
  std::vector<HailoTensorPtr> tensors = roi->get_tensors();

  std::vector<HailoDetection> detections = demo_detection_objects();
  hailo_common::add_detections(roi, detections);
}
```

Recompile again and run the test pipeline, if all is correct then you should see the original video run with no problems. If you are unable to see any detections this is because they are attached to each buffer, however no overlay is drawing them onto the image itself. To see how our detection boxes can be drawn, read further in *Next Steps Drawing*.

 Confidential and Proprietary | Copyright © 2025 – Hailo Technologies Ltd.

Next Steps

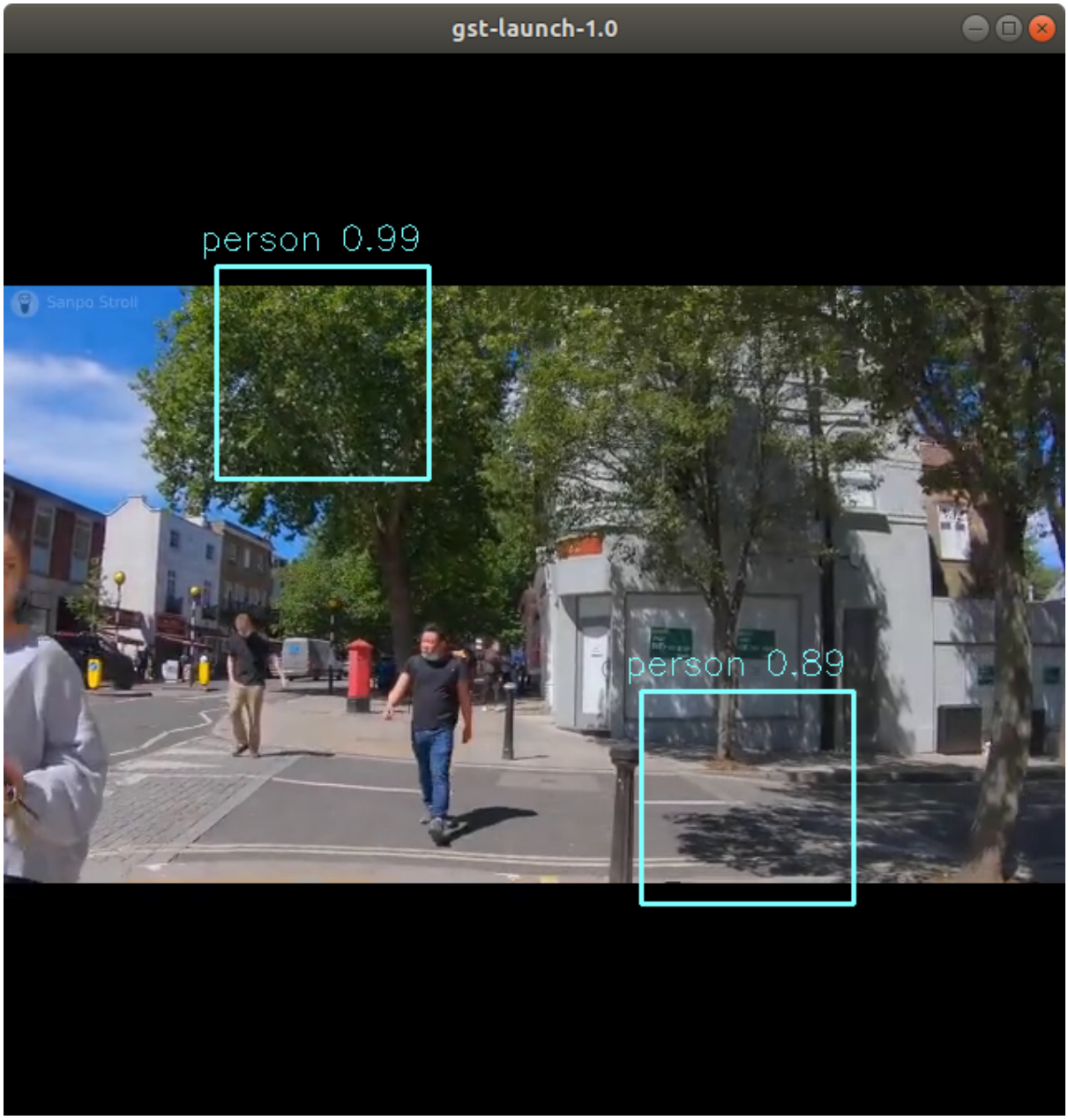
Drawing

At this point we have a working postprocess that attaches two detection boxes to each passing buffer. But to get the GStreamer pipeline to draw those boxes onto the image, We provide a GStreamer element - *hailooverlay* - that draws any Hailo provided output classes (detections, classifications, landmarks, etc..) on the buffer, to do this include it in your pipeline.
The element should be added in the pipeline after the `hailofilter` element with our postprocess.
Now the pipeline should look like:

```
gst-launch-1.0 filesrc location=$TAPPAS_WORKSPACE/apps/detection/resources/
↪detection.mp4 name=src_0 ! decodebin ! videoscale ! video/x-raw, pixel-aspect-
↪ratio=1/1 ! videoconvert ! queue ! hailonet hef-path=$TAPPAS_WORKSPACE/apps/
↪detection/resources/yolov5m_wo_spp.hef is-active=true ! queue leaky=no max-size-
↪buffers=30 max-size-bytes=0 max-size-time=0 ! hailofilter so-path=$TAPPAS_
↪WORKSPACE/apps/libs/post_processes/libmy_post.so qos=false ! queue !
↪hailooverlay ! videoconvert ! fpsdisplaysink video-sink=ximagesink name=hailo_
↪display sync=true text-overlay=false
```

Run the expanded pipeline above to see the original video, but this time with the two detection boxes we added.

 Confidential and Proprietary | Copyright © 2025 – Hailo Technologies Ltd.



Both boxes will be labeled as `person`, and each is shown with the assigned `confidence`. Obviously, the two boxes don't move or match any object in the video; this is because for the benefit of this tutorial the values are hardcoded. It is up to the user to extract the correct numbers from the inferred tensor of their network, as can be seen among the postprocesses already implemented in the TAPPAS each network can be different. This guide provides a strong starting point for further development.

Multiple Filters in One .so

While the `hailofilter` always calls on a `filter()` function by default, the user can provide the element access to other functions in `.so` to call instead. This may be of interest for developing a postprocess that applies to mutliple networks, but each network needs slightly different starting parameters (in the TAPPAS case, mutliple flavors of the Yolo detection network are handled via the same .so). This can be achieved by declaring the extra functions in the header file, then pointing the `hailofilter` to that function via the `function-name` property. Taking the Yolo networks as an example, open up libs/postprocesses/detection/yolo_postprocess.hpp to see what functions are made available to the `hailofilter`:

 Confidential and Proprietary | Copyright © 2025 – Hailo Technologies Ltd.

```
#pragma once
#include "hailo_objects.hpp"
#include "hailo_common.hpp"


__BEGIN_DECLS
void filter(HailoROIPtr roi);
void yolov5(HailoROIPtr roi, void *params_void_ptr);
void yolox(HailoROIPtr roi, void *params_void_ptr);
void yoloxx(HailoROIPtr roi, void *params_void_ptr);
void yolov3(HailoROIPtr roi, void *params_void_ptr);
void yolov4(HailoROIPtr roi, void *params_void_ptr);
void tiny_yolov4_license_plates(HailoROIPtr roi, void *params_void_ptr);
void yolov5_no_persons(HailoROIPtr roi, void *params_void_ptr);
void yolov5_no_faces(HailoROIPtr roi, void *params_void_ptr);
void yolov5_counter(HailoROIPtr roi, void *params_void_ptr);
void yolov5_vehicles_only(HailoROIPtr roi, void *params_void_ptr);
void yolov5_personface(HailoROIPtr roi, void *params_void_ptr);
void yolov5_personface_letterbox(HailoROIPtr roi, void *params_void_ptr);
void yolov5_no_faces_letterbox(HailoROIPtr roi, void *params_void_ptr); //should
 ↪add to python too
void yolov5_adas(HailoROIPtr roi, void *params_void_ptr);
__END_DECLS
```

Any of the functions declared here can be given as a `function-name` property to the `hailofilter` element. Condsider this pipeline for running the `Yolov5` network:

```
gst-launch-1.0 filesrc location=/local/workspace/tappas/apps/detection/resources/
 ↪detection.mp4 name=src_0 ! decodebin ! videoscale ! video/x-raw, pixel-aspect-
 ↪ratio=1/1 ! videoconvert ! queue leaky=no max-size-buffers=30 max-size-bytes=0 max-
 ↪size-time=0 ! hailonet hef-path=/local/workspace/tappas/apps/detection/
 ↪resources/yolov5m_wo_spp.hef is-active=true ! queue leaky=no max-size-buffers=30
 ↪max-size-bytes=0 max-size-time=0 ! hailofilter function-name=yolov5 so-path=/
 ↪local/workspace/tappas/apps/libs/post_processes//libyolo_post.so qos=false !
 ↪queue leaky=no max-size-buffers=30 max-size-bytes=0 max-size-time=0 !
 ↪hailooverlay ! videoconvert ! fpsdisplaysink video-sink=xvimagesink name=hailo_
 ↪display sync=false text-overlay=false
```

The `hailofilter` above that performs the post-process points to `libyolo_post.so` in the `so-path`, but it also includes the property `function-name=yolov5`. This lets the `hailofilter` know that instead of the default `filter()` function it should call on the `yolov5` function instead.

4.3.5. Writing Your Own Python Postprocess

Overview

To add a network to the TAPPAS that is not already supported, requires the implementation of a new post-process. Fortunately with the use of the *hailopython*, there is no need to create any new GStreamer elements, only to provide a Python module that applies your post-processing! This section will review how to create a python module and what mechanisms/structures are available for creating a post-process.

 Confidential and Proprietary | Copyright © 2025 – Hailo Technologies Ltd.

Getting Started

hailopython requires a module and a Python function.

Python Module Template

Below is a template for a Python module for the hailopython element.

```
import hailo

# Importing VideoFrame before importing GST is must
from gsthailo import VideoFrame
from gi.repository import Gst

# Create 'run' function, that accepts one parameter - VideoFrame, more about
↪VideoFrame later.
# `run` is default function name if no name is provided
def run(video_frame: VideoFrame):
  print("My first Python postprocess!")

  return Gst.FlowReturn.OK
```

To call it, create a pipeline with hailopython:

```
gst-launch-1.0 videotestsrc ! hailopython module=$PATH_TO_MODULE/my_module.py !
↪autovideosink
```

Extracting the Tensors

One of the first steps in each post-process is to get the output tensors. This can be done by one of two methods `video_frame.roi.get_tensor(tensor_name)` or `video_frame.roi.get_tensors()`

```
def run(video_frame: VideoFrame):
  for tensor in video_frame.roi.get_tensors():
    print(tensor.name())
  my_tensor = roi.get_tensor("output_layer_name")
  print(f"shape is {my_tensor.height()}X{my_tensor.width()}X{my_tensor.features()}
↪)
```

After doing this it is possible to convert this object of type HailoTensor to a numpy array on which perform post-processing operations can be performed more conveniently. This is a fairly simple step, just use np.array on a given HailoTensor.

> Notice that np.array has a parameter that determines whether to copy the memory or use the original buffer.

```
def run(video_frame: VideoFrame):
  my_tensor = roi.get_tensor("output_layer_name")
  # To create a numpy array with new memory
  my_array = np.array(my_tensor)
  # To create a numpy array with original memory
  my_array = np.array(my_tensor, copy=False)
```

There are some other methods in HailoTensor, that can be performed `dir(my_tensor)` or `help(my_tensor)`.

 Confidential and Proprietary | Copyright © 2025 – Hailo Technologies Ltd.

Adding Results

After processing the net results and obtaining the post-processed results, they can be used as preferred. Below demonstrates how to add them to the original image in order to draw them later with the hailooverlay element. In order to add the post-processed result to the original image - use the `video_frame.roi.add_object` method. This method adds a HailoObject object to the image. There are several types of objects that are currently supported: hailo.HailoClassification - Classification of the image. hailo.HailoDetection - Detection in the image. hailo.HailoLandmarks - Landmarks in the image.

It is possible to create one of these objects and then add it with the roi.add_object method.

```
def run(video_frame: VideoFrame):
  classification = hailo.HailoClassification(type='animal', index=1, label='horse',
↪ confidence=0.67)

  # You can also create a classification without class id (index).
  classification = hailo.HailoClassification(type='animal', label='horse',
↪confidence=0.67)

  video_frame.roi.add_object(classification)
```

One can also add objects to detections:

```
def run(video_frame: VideoFrame):
  # Adds a person detection in the bottom right quarter of the image. (normalized only)
  person_bbox = hailo.HailoBBox(xmin=0.5, ymin=0.5, width=0.5, height=0.5)
  person = hailo.HailoDetection(bbox=person_bbox, label='person', confidence=0.97)
  video_frame.roi.add_object(person)

  # Now, Adds a face to the person, at the top of the person. (normalized only)
  face_bbox  = hailo.HailoBBox(xmin=0.0, ymin=0.0, width=1, height=0.2)
  face = hailo.HailoDetection(bbox=face_bbox, label='face', confidence=0.84)
  person.add_object(face)
  # No need to add the face to the roi because it is already in the person that is in the
↪roi.
```

Next Steps

Drawing

In order to draw the post-processed results on the original image use the hailooverlay element. It is already familiar with our HailoObject types and knows how to draw classifications, detections, and landmarks onto the image.

```
gst-launch-1.0 filesrc location=$TAPPAS_WORKSPACE/apps/detection/detection.mp4
↪name=src_0 ! decodebin \
! videoscale ! video/x-raw, pixel-aspect-ratio=1/1 ! videoconvert ! queue leaky=no
↪max-size-buffers=30 \
max-size-bytes=0 max-size-time=0 ! hailonet hef-path=$TAPPAS_WORKSPACE/apps/
↪detection/yolov5m_wo_spp.hef \
is-active=true ! queue leaky=no max-size-buffers=30 max-size-bytes=0 \
max-size-time=0 ! hailopython module=$TAPPAS_WORKSPACE/apps/detection/my_module.
↪py qos=false ! queue \
leaky=no max-size-buffers=30 max-size-bytes=0 max-size-time=0 ! hailooverlay
↪qos=false ! videoconvert ! \
fpsdisplaysink video-sink=ximagesink name=hailo_display sync=true text-
↪overlay=false
```

This is the standard detection pipeline with a python module for post-processing.

 Confidential and Proprietary | Copyright © 2025 – Hailo Technologies Ltd.

Multiple Functions in One Python Module

There is an option to write several post-process functions in the same module. In order to run each of them just add the `function` property to the `hailopython` element:

```
import hailo

# Importing VideoFrame before importing GST is must
from gsthailo import VideoFrame
from gi.repository import Gst


def post_process_function(video_frame: VideoFrame):
  print("My first Python postprocess!")

def other_post_function(video_frame: VideoFrame):
  print("Other Python postprocess!")
```

```
gst-launch-1.0 videotestsrc ! hailopython module=$PATH_TO_MODULE/my_module.py
↪function=other_post_function ! autovideosink
```

VideoFrame Class

In addition to providing `buffer` and `HailoROI` access functions, the `VideoFrame` module provides helper functions for accessing the buffer through NumPy

List all Available Methods and Members

Running the following command would display a list of methods and members available:

```
python3 -c "import hailo; help(hailo)"
```

4.3.6. Retraining TAPPAS Models

Instructions

If you wish to use a TAPPAS pipeline/demo with your own model, the easiest solution is:

1. **Read the relevant TAPPAS pipeline/demo README page:**
    - It lists the ModelZoo models that are used on this pipeline
    - If Model Zoo retraining dockers are available for those models, links are given; In addition it describes how to reconfigure TAPPAS for the retrained models
2. Follow the retraining docker's instructions to retrain the model with your own data
3. Follow the retraining docker's instructions to compile the network
4. Reconfigure TAPPAS scripts to run your new .hef file(s)

 Confidential and Proprietary | Copyright © 2025 – Hailo Technologies Ltd.

Notes

- Easy post-processing JSON reconfiguration is currently only available for YOLO architecures. For other architectures, recreating post-processing .so file is required. The relevant code and header files are mentioned on the README pages.
- Models that use on-chip RGBX->RGB layers does not appear yet on the ModelZoo. Therefore, many models that are used for iMX demos (and others; see on the README of each app) should be added those layers manually to create models that will fit TAPPAS:
  - Use the non-rgbx network from the above table for retraining and compilation, with one simple modification - On the model alls file (on hailo_model_zoo/cfg/alls), add **right after normalization command**:

```
reshape_rgb = input_conversion(input_layer1, tf_rgbx_to_hailo_rgb)
```

 Confidential and Proprietary | Copyright © 2025 – Hailo Technologies Ltd.

5. Elements

5.1. Hailo Cropper

5.1.1. Overview

`HailoCropper` is an element providing cropping functionality, designed for application with cascading networks, meaning doing one task based on a previous task. It has 1 sink and 2 sources. `HailoCropper` receives a frame on its sink pad, then invokes it's `prepare_crops` method that returns the vector of crop regions of interest (crop_roi), For each crop_roi it creates a cropped image (representing it's x, y, width, height in the full frame). The cropped images are then sent to the second src. From the first src we push the original frame that the detections were cropped from.

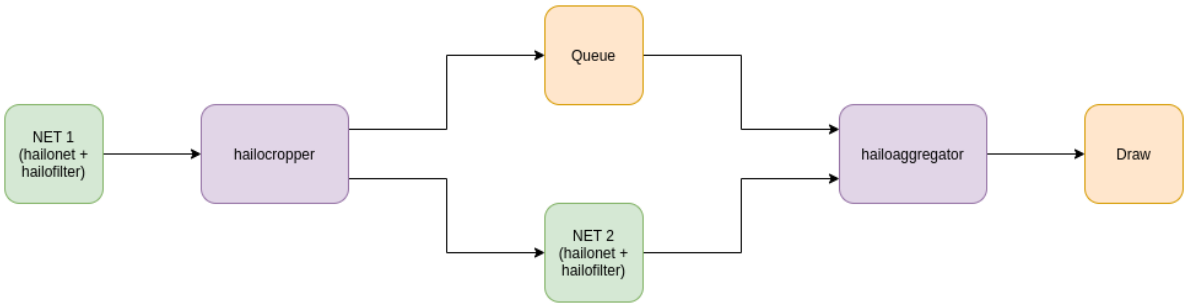
By default, `HailoCropper` receives a video frame that has detections (means a previous `HailoNet` + `HailoFilter` ran) on it's sinkpad. For each detection it creates a cropped image (using a specific algorithm to create a scaled image with the same aspect ratio). This is used by the cascading networks app Face Landmarks based on Face Detection.

Derived classes can override the default `prepare_crops` behaviour and decide where to crop and how many times. *hailotilecropper* element does this exact thing when splitting the frame into tiles by rows and columns.

Parameters

There is only one property for this element other than the common 'name' and 'parent'. The name of this boolean property is 'internal-offset' and it is used to determine whether we use the original offset* of the buffer or overwrite it with our own offset. The offset of the buffer is given to the original buffer and all the crops, and used by the hailoaggregator, to make sure the cropped detections we are 'muxing' with the original buffer are actually from the same buffer.*Offset is an attribute of buffer that determines on what offset this buffer is since the start of the pipeline run, represented by number of buffers. It's similar to frame-id in video. On some videos the offset attribute is not created by the filesrc element and it is set to -1 (casted to uint64), therefore if we want to use it to determine what the current frame is, we should somehow track the number of buffers and set this offset accordingly.

5.1.2. Example



 Confidential and Proprietary | Copyright © 2025 – Hailo Technologies Ltd.

5.1.3. Hierarchy

```
GObject
 +----GInitiallyUnowned
    +----GstObject
       +----GstElement
          +----GstHailoCropper

Pad Templates:
 SRC template: 'src'
  Availability: Always
  Capabilities:
   ANY

 SINK template: 'sink'
  Availability: Always
  Capabilities:
   ANY

Element has no clocking capabilities.
Element has no URI handling capabilities.

Pads:
 SINK: 'sink'
  Pad Template: 'sink'
 SRC: 'src_0'
  Pad Template: 'src'
 SRC: 'src_1'
  Pad Template: 'src'

Element Properties:
 name         : The name of the object
            flags: readable, writable
            String. Default: "hailocropper0"
 parent       : The parent of the object
            flags: readable, writable
            Object of type "GstObject"
 internal-offset   : Whether to use Gstreamer offset of internal offset.
            flags: readable, writable, controllable
            Boolean. Default: false
```

5.1.4. Hailo-15

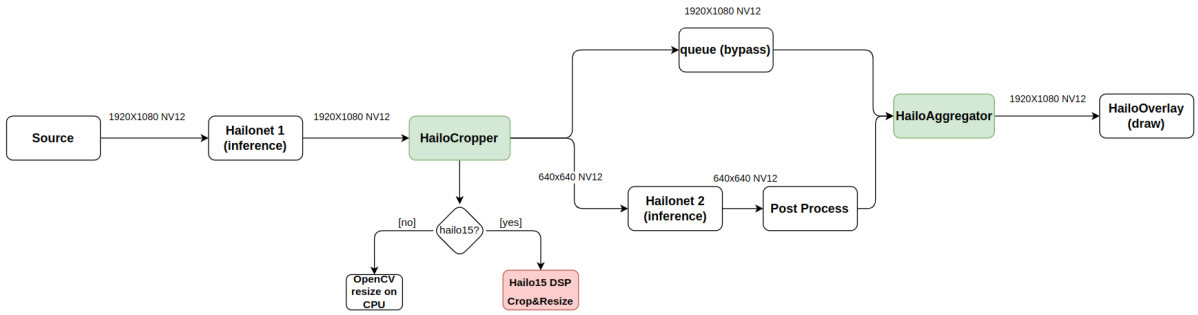
HailoCropper can utilize the on-chip DSP (Digital Signal Processor), to perform resize and crop operations.

The DSP is used by default on the Hailo-15 machine, and can be disabled by setting the `use-dsp` property. When disabled OpenCV will be used (on the CPU) to perform the resize and crop operations.

HailoCropper holds a buffer pool (GstBufferPool) that manages the buffers, used by the DSP. The buffer pool is responsible for allocating and freeing the buffers when the reference count of a buffer reaches 0. Buffer pool size (maximum buffers that can be allocated simultaneously in the pool) can be controlled using the `pool-size` property.

It is recommended to make sure that a buffer is contiguous in memory, before being sent to the DSP. Non-contiguous buffers will be copied to a new buffer, before being sent to the DSP.

 Confidential and Proprietary | Copyright © 2025 – Hailo Technologies Ltd.



5.2. Hailo Overlay

5.2.1. Overview

HailoOverlay is a drawing element that can draw postprocessed results on an incoming video frame. This element supports the following results:

- Detection - Draws the rectengle over the frame, with the label and confidence (rounded).
- Classification - Draws a classification over the frame, at the top left corner of the frame.
- Landmarks - Draws a set of points on the given frame at the wanted coordintates.
- Tiles - Can draw tiles as a thin rectengle.

Parameters

As a member of the GstBaseTransform hierarchy, the hailooverlay element supports qos (Quality of Service). Although qos typically tries to guarantee some level of performance, it can lead to frames dropping. For this reason it is advised to always set `qos=false` to avoid either tensors being dropped or not drawn.

5.2.2. Hierarchy

```
GObject
 +----GInitiallyUnowned
    +----GstObject
       +----GstElement
          +----GstBaseTransform
             +----GstHailoOverlay

Pad Templates:
 SINK template: 'sink'
  Availability: Always
  Capabilities:
   video/x-raw
         format: { (string)RGB }
          width: [ 1, 2147483647 ]
         height: [ 1, 2147483647 ]
      framerate: [ 0/1, 2147483647/1 ]

 SRC template: 'src'
  Availability: Always
  Capabilities:
   video/x-raw
         format: { (string)RGB }
```

(continues on next page)

 Confidential and Proprietary | Copyright © 2025 – Hailo Technologies Ltd.

(continued from previous page)

```
          width: [ 1, 2147483647 ]
         height: [ 1, 2147483647 ]
       framerate: [ 0/1, 2147483647/1 ]

Element has no clocking capabilities.
Element has no URI handling capabilities.

Pads:
 SINK: 'sink'
  Pad Template: 'sink'
 SRC: 'src'
  Pad Template: 'src'

Element Properties:
 name        : The name of the object
             flags: readable, writable
             String. Default: "hailooverlay0"
 parent      : The parent of the object
             flags: readable, writable
             Object of type "GstObject"
 qos         : Handle Quality-of-Service events
             flags: readable, writable
             Boolean. Default: false
```

5.3. Hailo DeviceStats

5.3.1. Overview

Hailodevicestats is an element that samples power and temperature. It doesn't have any pads, it just has to be part of the pipeline. An example for using this element could be found under the `detection` / `multi-stream_multidevice` app.

Parameters

Determine the time period between samples with the `interval` property.

Choose device with the `device-id` property.

5.3.2. Hierarchy

```
GObject
 +----GInitiallyUnowned
    +----GstObject
       +----GstElement
          +----GstHailoDeviceStats


Factory Details:
 Rank          primary (256)
 Long-name        hailodevicestats element
 Klass          Hailo/Device
 Description       Log Hailo-8 device statistics
 Author         Omer Salem <omers@hailo.ai>
```

(continues on next page)

 Confidential and Proprietary | Copyright © 2025 – Hailo Technologies Ltd.

(continued from previous page)

```
Pad Templates:

 none


Element has no clocking capabilities.


Element has no URI handling capabilities.


Pads:

 none


Properties:

 name         : The name of the object
            flags: readable, writable
            String. Default: "hailodevicestats0"

 parent       : The parent of the object
            flags: readable, writable
            Object of type "GstObject"

 interval      : Time period between samples, in seconds
            flags: readable, writable
            Unsigned Integer. Range: 0 - 4294967295 Default: 1

 device-id      : Device ID ([<domain>]:<bus>:<device>.<func>, same as in lspci
↪command)
            flags: readable, writable
            String. Default: null

 silent       : Should print statistics
            flags: readable, writable
            Boolean. Default: false

 power-measurement  : Current power measurement of device
            flags: readable
            Float. Range:        0 -   3.402823e+38 Default:        0

 temperature     : Current temperature of device
            flags: readable
            Float. Range:        0 -   3.402823e+38 Default:        0
```

5.4. Hailo Filter

5.4.1. Overview

Hailofilter is an element which enables the user to apply a postprocess operation on hailonet's output tensors. It provides an entry point for a compiled .so file that the user writes, inside of which they will have access to the original image frame, the tensors output by the network for that frame, and any metadata attached. At first the hailofilter will read the buffer from the sink pad, then apply the filter defined in the provided .so, until finally sending the filtered buffer along the source pad to continue down the pipeline.

 Confidential and Proprietary | Copyright © 2025 – Hailo Technologies Ltd.

Parameters

The most important parameter here is the `so-path`. Here the user provides the path to your compiled .so that applies your wanted filter. By default, the hailofilter will call on a filter() function within the .so as the entry point. If your .so has multiple entry points, for example in the case of slightly different network flavors, then you can chose which specific filter function to apply via the `function-name` parameter. As a member of the GstVideoFilter hierarchy, the hailofilter element supports qos (Quality of Service). Although qos typically tries to garuantee some level of performance, it can lead to frames dropping. For this reason it is advised to always set `qos=false` to avoid either tensors being dropped or not drawn.

5.4.2. Hierarchy

```
GObject
 +----GInitiallyUnowned
    +----GstObject
       +----GstElement
          +----GstBaseTransform
             +----GstHailofilter

Pad Templates:
 SRC template: 'src'
  Availability: Always
  Capabilities:
   ANY

 SINK template: 'sink'
  Availability: Always
  Capabilities:
   ANY

Element has no clocking capabilities.
Element has no URI handling capabilities.

Pads:
 SINK: 'sink'
  Pad Template: 'sink'
 SRC: 'src'
  Pad Template: 'src'

Element Properties:
 name         : The name of the object
            flags: readable, writable
            String. Default: "hailofilter-0"
 parent       : The parent of the object
            flags: readable, writable
            Object of type "GstObject"
 qos          : Handle Quality-of-Service events
            flags: readable, writable
            Boolean. Default: false
 so-path      : Location of the so file to load
            flags: readable, writable, changeable only in NULL or READY state
            String. Default: null
 function-name    : function-name
            flags: readable, writable, changeable only in NULL or READY state
            String. Default: "filter"
 use-gst-buffer    : use function with access to the Gst Buffer
            flags: readable, writable, controllable
            Boolean. Default: false
```

 Confidential and Proprietary | Copyright © 2025 – Hailo Technologies Ltd.

5.5. Hailo Tracker

5.5.1. Overview

HailoTracker is an element which enables the user to track *HailoDetection* objects using the Joint-Detection-and-Embedding (JDE) Tracking algorithm. JDE is a fast and high-performance multiple-object tracking algorithm that associates between detections in consecutive frames based on size/location/movement. Associations are made by linear assignment via a Kalman Filter.
For more information on JDE Tracking and Kalman Filtering read here.

Parameters

The hailotracker element provides a series of properties that allow you to adjust the tracking algorithm. The most important property to set is `class-id`: this determines if the tracker will track all *HailoDetection* objects indiscriminately of class or focus only on detections of a specific class id (the default behavior is to track across-classes).

5.5.2. Hierarchy

```
GObject
 +----GInitiallyUnowned
    +----GstObject
       +----GstElement
          +----GstBaseTransform
             +----GstVideoFilter
                +----GstHailoTracker

Pad Templates:
 SRC template: 'src'
  Availability: Always
  Capabilities:
   ANY

 SINK template: 'sink'
  Availability: Always
  Capabilities:
   ANY

Element has no clocking capabilities.
Element has no URI handling capabilities.

Pads:
 SINK: 'sink'
  Pad Template: 'sink'
 SRC: 'src'
  Pad Template: 'src'

Element Properties:
 name        : The name of the object
              flags: readable, writable
              String. Default: "hailotracker0"
 parent      : The parent of the object
              flags: readable, writable
              Object of type "GstObject"
 qos         : Handle Quality-of-Service events
              flags: readable, writable
              Boolean. Default: true
 debug       : Enable debug mode.
```

(continues on next page)

 Confidential and Proprietary | Copyright © 2025 – Hailo Technologies Ltd.

(continued from previous page)

```
            flags: readable, writable, controllable
            Boolean. Default: false
class-id       : The class id of the class to track. Default -1 crosses classes.
            flags: readable, writable, changeable only in NULL or READY state
            Integer. Range: -2147483648 - 2147483647 Default: -1
kalman-dist-thr   : Threshold used in Kalman filter to compare Mahalanobis cost
↪matrix. Closer to 1.0 is looser.
            flags: readable, writable, controllable
            Float. Range:        0 -        1 Default:       0.7
iou-thr       : Threshold used in Kalman filter to compare IOU cost matrix. Closer to
↪1.0 is looser.
            flags: readable, writable, controllable
            Float. Range:        0 -        1 Default:       0.8
init-iou-thr     : Threshold used in Kalman filter to compare IOU cost matrix of newly
↪found instances. Closer to 1.0 is looser.
            flags: readable, writable, controllable
            Float. Range:        0 -        1 Default:       0.9
keep-tracked-frames : Number of frames to keep without a successful match before a
↪'tracked' instance is considered 'lost'.
            flags: readable, writable, controllable
            Integer. Range: 0 - 2147483647 Default: 2
keep-new-frames   : Number of frames to keep without a successful match before a 'new'
↪instance is removed from the tracking record.
            flags: readable, writable, controllable
            Integer. Range: 0 - 2147483647 Default: 2
keep-lost-frames   : Number of frames to keep without a successful match before a 'lost
↪' instance is removed from the tracking record.
            flags: readable, writable, controllable
            Integer. Range: 0 - 2147483647 Default: 2
```

5.6. Hailo Tile Aggregator

5.6.1. Overview

`HailoTileAggregator` is a derived element of *hailoAggregator*. A complement to the *HailoTileCropper*, the two elements work together to form a versatile tiling apps.

The element extends two methods of the parent element:

- `handle_sub_frame_roi`: Functionality to perform for each incoming sub frame. .. code-block:

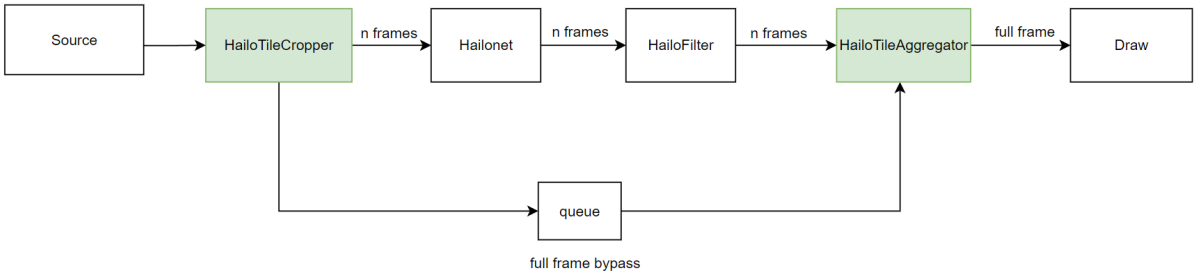
  ```
  Performs ``remove_exceeded_bboxes`` (remove boxes close to boundary - using
  ↪given border_threshold) and then parent element performs flatten detections.
  ```

- `post_aggregation`: Functionality to perform after all frames are aggregated succesfully. .. code-block:

  ```
  Performs ``remove_large_landscape`` and ``NMS``.
  ```

 Confidential and Proprietary | Copyright © 2025 – Hailo Technologies Ltd.

5.6.2. Example



5.6.3. Hierarchy

```
GObject
 +----GInitiallyUnowned
    +----GstObject
       +----GstElement
          +----GstHailoAggregator
             +----GstHailoTileAggregator

Pad Templates:
 SRC template: 'src'
  Availability: Always
  Capabilities:
   ANY

 SINK template: 'sink'
  Availability: Always
  Capabilities:
   ANY

Element has no clocking capabilities.
Element has no URI handling capabilities.

Pads:
 SINK: 'sink_0'
  Pad Template: 'sink'
 SINK: 'sink_1'
  Pad Template: 'sink'
 SRC: 'src'
  Pad Template: 'src'

Element Properties:
 name         : The name of the object
            flags: readable, writable
            String. Default: "hailotileaggregator0"
 parent       : The parent of the object
            flags: readable, writable
            Object of type "GstObject"
 flatten-detections  : perform a 'flattening' functionality on the detection metadata
↪when receiving each frame
            flags: readable, writable, changeable only in NULL or READY state
            Boolean. Default: false
 iou-threshold    : threshold
            flags: readable, writable, changeable only in NULL or READY state
            Float. Range:        0 -        1 Default:        0.3
 border-threshold   : border threshold
            flags: readable, writable, changeable only in NULL or READY state
```

(continues on next page)

 Confidential and Proprietary | Copyright © 2025 – Hailo Technologies Ltd.

(continued from previous page)

```
            Float. Range:        0 -        1 Default:        0.1
 remove-large-landscape: remove large landscape objects when running in multi-scale
↪mode
            flags: readable, writable, changeable only in NULL or READY state
```

5.7. Hailo Tile Cropper

5.7.1. Overview

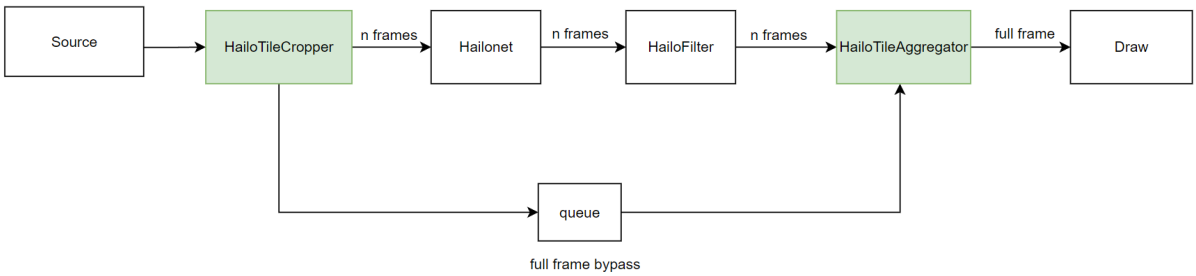
`HailoTileCropper` is a derived element of *hailoCropper*. It overrides the default `prepare_crops` behaviour to return a vector of tile regions of interest, and allows splitting the incoming frame into tiles by rows and columns. Each tile stores their x, y, width, and height (with overlap between tiles included) in the full frame. Just like the base HailoCropper, the full original frame is sent to the first src pad while all the cropped images are sent to the second.

hailoaggregator wiil aggregate the cropped tiles and stitch them back to the original resolution.

Parameters

- tiles-along-x-axis : Number of tiles along x axis (columns) - default 2
- tiles-along-y-axis : Number of tiles along x axis (rows) - default 2
- overlap-x-axis : Overlap in percentage between tiles along x axis (columns) - default 0
- overlap-y-axis : Overlap in percentage between tiles along y axis (rows) - default 0
- tiling-mode : Tiling mode (0 - single-scale, 1 - multi-scale) - default 0
- scale-level : Scales (layers of tiles) in addition to the main layer 1: [(1 X 1)] 2: [(1 X 1), (2 X 2)] 3: [(1 X 1), (2 X 2), (3 X 3)]] - default 2

5.7.2. Example



5.7.3. Hierarchy

```
GObject
 +----GInitiallyUnowned
    +----GstObject
       +----GstElement
          +----GstHailoBaseCropper
             +----GstHailoTileCropper
Pad Templates:
 SRC template: 'src'
  Availability: Always
```

(continues on next page)

(continued from previous page)

```
  Capabilities:
   video/x-raw
         format: { (string)RGB, (string)YUY2 }
          width: [ 1, 2147483647 ]
         height: [ 1, 2147483647 ]
       framerate: [ 0/1, 2147483647/1 ]

 SINK template: 'sink'
  Availability: Always
  Capabilities:
   video/x-raw
         format: { (string)RGB, (string)YUY2 }
          width: [ 1, 2147483647 ]
         height: [ 1, 2147483647 ]
       framerate: [ 0/1, 2147483647/1 ]

Element has no clocking capabilities.
Element has no URI handling capabilities.

Pads:
 SINK: 'sink'
  Pad Template: 'sink'
 SRC: 'src_0'
  Pad Template: 'src'
 SRC: 'src_1'
  Pad Template: 'src'

Element Properties:
 name         : The name of the object
             flags: readable, writable
             String. Default: "hailotilecropper0"
 parent       : The parent of the object
             flags: readable, writable
             Object of type "GstObject"
 internal-offset   :
             Whether to use Gstreamer offset of internal offset.
             NOTE: If using file sources, Gstreamer does not generate offsets for
↪buffers,
             so this property should be set to true in such cases.
             flags: readable, writable, controllable
             Boolean. Default: false
 tiles-along-x-axis  : Number of tiles along x axis (columns)
             flags: readable, writable, changeable only in NULL or READY state
             Unsigned Integer. Range: 1 - 20 Default: 2
 tiles-along-y-axis  : Number of tiles along x axis (rows)
             flags: readable, writable, changeable only in NULL or READY state
             Unsigned Integer. Range: 1 - 20 Default: 2
 overlap-x-axis    : Overlap in percentage between tiles along x axis (columns)
             flags: readable, writable, changeable only in NULL or READY state
             Float. Range:        0 -        1 Default:        0
 overlap-y-axis    : Overlap in percentage between tiles along y axis (rows)
             flags: readable, writable, changeable only in NULL or READY state
             Float. Range:        0 -        1 Default:        0
 tiling-mode     : Tiling mode
             flags: readable, writable
             Enum "GstHailoTileCropperTilingMode" Default: 0, "single-scale"
               (0): single-scale   - Single Scale
               (1): multi-scale    - Multi Scale
 scale-level     : 1: [(1 X 1)] 2: [(1 X 1), (2 X 2)] 3: [(1 X 1), (2 X 2), (3 X 3)]]
             flags: readable, writable, changeable only in NULL or READY state
             Unsigned Integer. Range: 1 - 3 Default: 2
```

 Confidential and Proprietary | Copyright © 2025 – Hailo Technologies Ltd.

5.8. Hailo Python

5.8.1. Overview

HailoPython is an element which enables the user to apply processing operations to an image via python. It provides an entry point for a python module that the user writes, inside of which they will have access to the Hailo raw-output (output tensors) and postprocessed-outputs (detections, classifications etc..) as well as the gstreamer buffer. The python function will be called for each buffer going through the hailopython element.

Parameters

The two parameters that define the function to call are `module` and `function` for the module path and function name respectively. In addition, as a member of the GstVideoFilter hierarchy, the hailofilter element supports qos (Quality of Service). Although qos typically tries to guarantee some level of performance, it can lead to frames dropping. For this reason it is advised to always set `qos=false` to avoid either tensors being dropped or not drawn.

5.8.2. Hierarchy

```
GObject
 +----GInitiallyUnowned
    +----GstObject
       +----GstElement
          +----GstBaseTransform
             +----GstVideoFilter
                +----GstHailoPython

Pad Templates:
 SRC template: 'src'
  Availability: Always
  Capabilities:
   video/x-raw
         format: { (string)RGB, (string)YUY2 }
          width: [ 1, 2147483647 ]
         height: [ 1, 2147483647 ]
      framerate: [ 0/1, 2147483647/1 ]

 SINK template: 'sink'
  Availability: Always
  Capabilities:
   video/x-raw
         format: { (string)RGB, (string)YUY2 }
          width: [ 1, 2147483647 ]
         height: [ 1, 2147483647 ]
      framerate: [ 0/1, 2147483647/1 ]

Element has no clocking capabilities.
Element has no URI handling capabilities.

Pads:
 SINK: 'sink'
  Pad Template: 'sink'
 SRC: 'src'
  Pad Template: 'src'

Element Properties:
 name        : The name of the object
            flags: readable, writable
            String. Default: "hailopython0"
```

(continues on next page)

 Confidential and Proprietary | Copyright © 2025 – Hailo Technologies Ltd.

(continued from previous page)

```
parent          : The parent of the object
                flags: readable, writable
                Object of type "GstObject"
qos             : Handle Quality-of-Service events
                flags: readable, writable
                Boolean. Default: true
module          : Python module name
                flags: readable, writable
                String. Default: "/local/workspace/tappas/processor.py"
function        : Python function name
                flags: readable, writable
                String. Default: "run"
```

5.9. Hailo Net

For documentation, please refer to HailoRT

5.10. Hailo Muxer

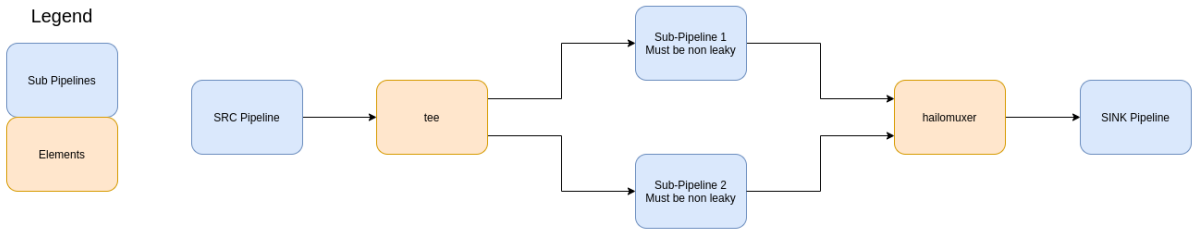
5.10.1. Overview

HailoMuxer is an element designed for our new multi-device application. It muxes 2 similar streams into 1 stream, holding both stream's metadata. It has 2 src elements and 1 sink, and whenever there are buffers on both src pads, it takes only 1 of the buffers and passes it on, with both buffer's metadata.

Parameters

There are no unique properties to hailomuxer. The only parameters are the baseclass parameters, which are 'name' and 'parent'.

5.10.2. Example



 Confidential and Proprietary | Copyright © 2025 – Hailo Technologies Ltd.

5.10.3. Hierarchy

```
GObject
 +----GInitiallyUnowned
    +----GstObject
       +----GstElement
          +----GstHailoMuxer

Pad Templates:
 SRC template: 'src'
  Availability: Always
  Capabilities:
   ANY

 SINK template: 'sink_%u'
  Availability: On request
  Capabilities:
   ANY

Element has no clocking capabilities.
Element has no URI handling capabilities.

Pads:
 SRC: 'src'
  Pad Template: 'src'

Element Properties:
 name        : The name of the object
            flags: readable, writable
            String. Default: "hailomuxer0"
 parent      : The parent of the object
            flags: readable, writable
            Object of type "GstObject"
```

5.11. Hailo Aggregator

5.11.1. Overview

HailoAggregator is an element designed for applications with cascading networks or cropping functionality, meaning to perform one task based on a previous task. It is a complement to the *HailoCropper*, the two elements work together to form versatile apps. It has 2 sink pads and 1 source: the first sinkpad receives the original frame from an upstream hailocropper, while the other receives cropped buffers from the other hailocropper. The HailoAggregator waits for all crops of a given orignal frame to arrive, then sends the original buffer with the combined metadata of all collected crops.

HailoAggregator also performs a 'flattening' functionality on the detection metadata when receiving each frame: detections are taken from the cropped frame, copied to the main frame and re-scaled/moved to their corresponding location in the main frame (x,y,width,height). As an example:

- Face Landmarks based on Face Detection - HailoCropper crops each face detection -> HailoNet + FaceLandmarks post for each face -> HailoAggregator aggregates the frames back.
- Tiling - *hailotilecropper* crops the image to tiles -> HailoNet + Detection post for each tile -> HailoAggregator aggregates the frames back and 'flatten' the detection objects in the metadata.

HailoAggregator exports two methods to extend or override in derived elements:

- `handle_sub_frame_roi`: Functionality to perform for each incoming sub frame.

 Confidential and Proprietary | Copyright © 2025 – Hailo Technologies Ltd.

```
Calls `flattening` method.
```

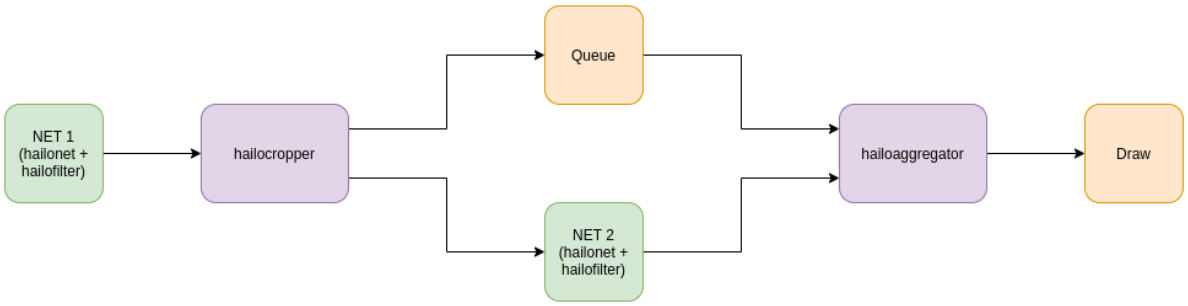
- `post_aggregation`: Functionality to perform after all frames are aggregated succesfully.

```
Base implementation does nothing.
```

Parameters

There are no unique properties to hailoaggregator. The only parameters are the baseclass parameters, which are 'name' and 'parent'.

5.11.2. Example



5.11.3. Hierarchy

```
GObject
 +----GInitiallyUnowned
    +----GstObject
       +----GstElement
          +----GstHailoAggregator

Pad Templates:
 SRC template: 'src'
  Availability: Always
  Capabilities:
   ANY

 SINK template: 'sink'
  Availability: Always
  Capabilities:
   ANY

Element has no clocking capabilities.
Element has no URI handling capabilities.

Pads:
 SINK: 'sink_0'
  Pad Template: 'sink'
 SINK: 'sink_1'
  Pad Template: 'sink'
 SRC: 'src'
  Pad Template: 'src'

Element Properties:
 name         : The name of the object
```

(continues on next page)

 Confidential and Proprietary | Copyright © 2025 – Hailo Technologies Ltd.

(continued from previous page)

```
            flags: readable, writable
            String. Default: "hailoaggregator0"
 parent       : The parent of the object
            flags: readable, writable
            Object of type "GstObject"
 flatten-detections : perform a 'flattening' functionality on the detection metadata
            when receiving each frame.
            flags: readable, writable, changeable only in NULL or READY state
            Boolean. Default: false
```

5.12. Hailo Gallery

5.12.1. Overview

HailoGallery is an element which enables the user to save and compare embeddings (HailoMatrix) that represents recogintion, in order to track objects across multiple streams. It is also enables saving and loading these embeddings into a local JSON file (database like), in order to track pre-saved objects.

Parameters

The hailogallery element provides a series of properties that allow you to adjust the gallery comparison algorithm. The most important property to set is `class-id`: this determines if the gallery will track all *HailoDetection* objects indiscriminately of class or focus only on detections of a specific class id (the default behavior is to track across-classes).

5.12.2. Hierarchy

```
GObject
+----GInitiallyUnowned
   +----GstObject
      +----GstElement
         +----GstBaseTransform
            +----GstHailoGallery

Pad Templates:
 SRC template: 'src'
  Availability: Always
  Capabilities:
   ANY

 SINK template: 'sink'
  Availability: Always
  Capabilities:
   ANY

Element has no clocking capabilities.
Element has no URI handling capabilities.

Pads:
 SINK: 'sink'
  Pad Template: 'sink'
 SRC: 'src'
  Pad Template: 'src'

Element Properties:
 name         : The name of the object
```

(continues on next page)

(continued from previous page)

```
            flags: readable, writable
            String. Default: "hailogallery0"
parent        : The parent of the object
            flags: readable, writable
            Object of type "GstObject"
qos           : Handle Quality-of-Service events
            flags: readable, writable
            Boolean. Default: false
class-id      : The class id of the class to update into the gallery. Default -1
↪crosses classes.
            flags: readable, writable, changeable only in NULL or READY state
            Integer. Range: -2147483648 - 2147483647 Default: -1
similarity-thr    : Similarity threshold used in Gallery to find New ID's. Closer to 1.
↪0 is less similar.
            flags: readable, writable, controllable
            Float. Range:        0 -        1 Default:      0.15
gallery-queue-size  : Number of Matrixes to save for each global ID
            flags: readable, writable, controllable
            Integer. Range: 0 - 2147483647 Default: 100
load-local-gallery  : Load Gallery from JSON file
            flags: readable, writable, controllable
            Boolean. Default: false
save-local-gallery  : Save Gallery to JSON file
            flags: readable, writable, controllable
            Boolean. Default: false
gallery-file-path   : Gallery JSON file path to load
            flags: readable, writable, controllable
            String. Default: null
```

5.13. Hailo RoundRobin

5.13.1. Overview

`HailoRoundRobin` is an element that provides muxing functionality. It receives input from one or more sink pads and forwards them into a single src pad in round-robin method.

It also adds metadata to each buffer with the input pad name it was received on, The metadata's pupose is to be able to de-mux it easily later on by *hailostreamrouter* . De-muxing by streamiddemux is not supported with this element.

It can work in 3 modes:

- Funnel mode - push every buffer when it is ready no matter which pad it came from.
- Blocking mode - push every buffer when it is its pad's turn, and if the buffer is not ready, block until ready. This is the default mode.
- Non Blocking mode - push every buffer when it is its pad's turn, and if the buffer is not ready, skip it. This mode is useful when the video sources are not stable and may stop sending buffers for a while. In this case, the pipeline should not be blocked and should continue to process the other streams.

When using non-blocking mode, the element maintains a queue for sink pad that holds pointers to buffers. When a buffer is pushed to a sink pad, it is added to the queue. When the src pad wants to push a buffer, the element tries to get a buffer from the queue of the pad that is next in line. If the queue is empty, the element retries to get a buffer from the queue for a number of times (retries-num property). If the queue is still empty, the element skips the pad and tries to get a buffer from the next pad in line. The size of the queue and the number of retries can be configured by the properties:

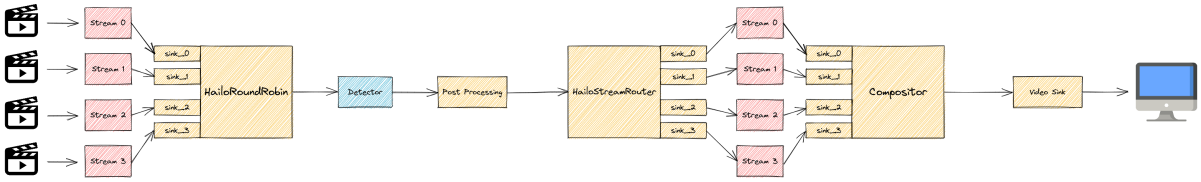
- queue-size - Size of the queue for each pad.
- retries-num - Number of retries to get a buffer from a pad queue.

 Confidential and Proprietary | Copyright © 2025 – Hailo Technologies Ltd.

When using non-blocking mode, Compositor element is not supported, since it requires all the streams to be synchronized.

5.13.2. Example

Here's an example of a pesudo pipeline muxing 4 streams into one detection pipeline, and then de-muxing them into 2 separate pipelines - one for person attributes and one for face attributes.




```
for ((n = 0; n < 4; n++)); do
  filesrc location=video_$n ! decodebin ! roundrobin.sink_$n
hailoroundrobin name=roundrobin mode=1 !
<Rest of the pipeline>
```

5.13.3. Hierarchy

GObject

+—-GInitiallyUnowned

+—-GstObject

+—-GstElement +—-GstHailoRoundRobin

Pad Templates:

SINK template: 'sink_%u' Availability: On request Capabilities:

ANY

SRC template: 'src' Availability: Always Capabilities:

ANY

Element has no clocking capabilities. Element has no URI handling capabilities.

Pads:

SRC: 'src' Pad Template: 'src'

Element Properties: mode : Select the mode of the element (0 - funnel mode (push every buffer when it is ready), 1 - blocking mode (push every buf

fer when it is its pad's turn, and if the buffer is not ready, block until ready), 2 - non blocking mode(push every buffer when it is i

turn, and if the buffer is not ready, skip it)) flags: readable, writable Enum "GstHailoRoundRobinMode" Default: 1, "blocking-mode"

(0): funnel-mode - Funnel Mode (push every buffer when it is ready) (1): blocking-mode - Blocking Mode (push every buffer when it is its pad's turn, and if the buffer is not ready,

block until ready) (2): non-blocking-mode - Non Blocking Mode (push every buffer when it is its pad's turn, and if the buffer is not re

ady, skip it)

 Confidential and Proprietary | Copyright © 2025 – Hailo Technologies Ltd.

name [The name of the object] flags: readable, writable String. Default: "hailoroundrobin0"

parent [The parent of the object] flags: readable, writable Object of type "GstObject"

queue-size [Size of the queue for each pad (only relevant when using non-blocking mode)] flags: readable, writable, controllable Unsigned Integer. Range: 1 - 10 Default: 3

retries-num [Number of retries to get a buffer from a pad queue (only relevant when using non-blocking mode)] flags: readable, writable, controllable Unsigned Integer. Range: 1 - 20 Default: 3

5.14. Hailo Stream Router

5.14.1. Overview

`HailoStreamRouter` is an element that provides de-muxing functionality. It is designed to be used after muxing any number of streams using a *hailoroundrobin* element.

Note: *hailoroundrobin* tags each incoming frame with the input pad name it was received on (adding it to the meta-data of the buffer).

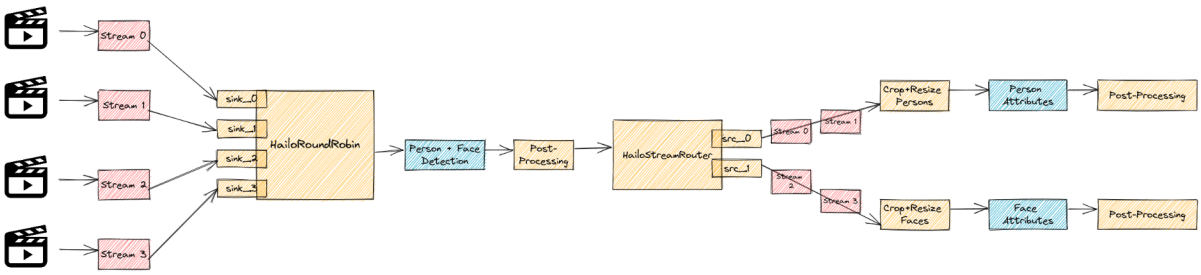
Each `HailoStreamRouter` has to be configured with a list of input names (In accordance with the `HailoRoundRobin` inputs). As an Example: .. code-block:

```
src_0::input-streams='<sink_0, sink_1>' src_1::input-streams='<sink_2, sink_3>'
```

`HailoStreamRouter` receives a frame on its sink pad, reads the input name from it's metadata, and then passes the frame to pre configured source pads.

5.14.2. Example

Here's an example of a pesudo pipeline muxing 4 streams into one detection pipeline, and then de-muxing them into 2 separate pipelines - one for person attributes and one for face attributes.

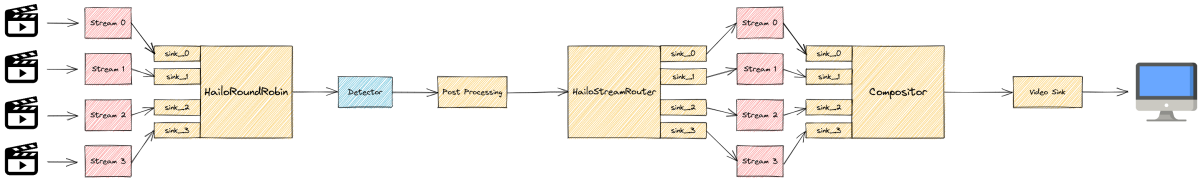



```
for ((n = 0; n < 4; n++)); do
 filesrc location=video_$n ! decodebin ! roundrobin.sink_$n
hailoroundrobin name=roundrobin funnel-mode=false !
 ... Logic ...
hailostreamrouter name=router src_0::input-streams='<sink_0, sink_1>' src_1::input-
↪streams='<sink_2, sink_3>'
router.src_0 ! ... Logic ...
router.src_1 ! ... Logic ...
```

In this example HailoStreamRouter is configured with 2 source pads that each have a list of 2 input streams.

Another Example is using `HailoStreamRouter` as a classic de-muxer, where each input stream is mapped to a single output, and into compositor afterwards.

 Confidential and Proprietary | Copyright © 2025 – Hailo Technologies Ltd.



5.14.3. Hierarchy

```
 GObject
 --GInitiallyUnowned
   --GstObject
      --GstElement
         --GstHailoStreamRouter

Implemented Interfaces:
 GstChildProxy

Pad Templates:
 SINK template: 'sink'
  Availability: Always
  Capabilities:
   ANY

 SRC template: 'src_%u'
  Availability: On request
  Capabilities:
   ANY
  Type: GstHailoStreamRouterPad
  Pad Properties:
   input-streams    : Input streams of a srcpad
              flags: readable, writable, controllable
              GstValueArray of GValues of type "gchararray"

Element has no clocking capabilities.
Element has no URI handling capabilities.

Pads:
 SINK: 'sink'
  Pad Template: 'sink'

Element Properties:
 name         : The name of the object
            flags: readable, writable, 0x2000
            String. Default: "hailostreamrouter0"
 parent       : The parent of the object
            flags: readable, writable, 0x2000
            Object of type "GstObject"
```

 Confidential and Proprietary | Copyright © 2025 – Hailo Technologies Ltd.

5.15. Hailo Export File

5.15.1. Overview

HailoExportFile is an element which provides an access point in the pipeline to export *HailoObjects meta* to a JSON file.
The meta itself is not changed or removed, and buffers continue onwards in the pipeline unchanged.

Parameters

The HailoExportFile element allows the user to change the output file name/path. The default is hailo_meta.json

5.15.2. Hierarchy

```
GObject
+----GInitiallyUnowned
   +----GstObject
      +----GstElement
         +----GstBaseTransform
            +----GstHailoExportFile

Pad Templates:
 SRC template: 'src'
  Availability: Always
  Capabilities:
   video/x-raw
        format: { (string)RGB, (string)YUY2 }
         width: [ 1, 2147483647 ]
        height: [ 1, 2147483647 ]
      framerate: [ 0/1, 2147483647/1 ]

 SINK template: 'sink'
  Availability: Always
  Capabilities:
   video/x-raw
        format: { (string)RGB, (string)YUY2 }
         width: [ 1, 2147483647 ]
        height: [ 1, 2147483647 ]
      framerate: [ 0/1, 2147483647/1 ]

Element has no clocking capabilities.
Element has no URI handling capabilities.

Pads:
 SINK: 'sink'
  Pad Template: 'sink'
 SRC: 'src'
  Pad Template: 'src'

Element Properties:
 name        : The name of the object
            flags: readable, writable
            String. Default: "hailoexportfile0"
 parent      : The parent of the object
            flags: readable, writable
            Object of type "GstObject"
 qos         : Handle Quality-of-Service events
            flags: readable, writable
```

(continues on next page)

 Confidential and Proprietary | Copyright © 2025 – Hailo Technologies Ltd.

(continued from previous page)

```
            Boolean. Default: false
 location       : Location of the JSON file to save
            flags: readable, writable, changeable only in NULL or READY state
            String. Default: "hailo_meta.json"
```

5.16. Hailo Export ZMQ

5.16.1. Overview

HailoExportZMQ is an element which provides an access point in the pipeline to export *HailoObjects meta* to a ZMQ socket.
The meta itself is not changed or removed, and buffers continue onwards in the pipeline unchanged.

Parameters

The HailoExportZMQ element allows the user to change the output port/protocol. The default is *tcp://*:5555*. Currently only PUB behvaior (PUB/SUB) is supported.

5.16.2. Hierarchy

```
GObject
+----GInitiallyUnowned
   +----GstObject
      +----GstElement
         +----GstBaseTransform
            +----GstHailoExportZMQ

Pad Templates:
 SINK template: 'sink'
  Availability: Always
  Capabilities:
   video/x-raw
        format: { (string)RGB, (string)YUY2 }
         width: [ 1, 2147483647 ]
        height: [ 1, 2147483647 ]
      framerate: [ 0/1, 2147483647/1 ]

 SRC template: 'src'
  Availability: Always
  Capabilities:
   video/x-raw
        format: { (string)RGB, (string)YUY2 }
         width: [ 1, 2147483647 ]
        height: [ 1, 2147483647 ]
      framerate: [ 0/1, 2147483647/1 ]

Element has no clocking capabilities.
Element has no URI handling capabilities.

Pads:
 SINK: 'sink'
  Pad Template: 'sink'
 SRC: 'src'
  Pad Template: 'src'
```

(continues on next page)

 Confidential and Proprietary | Copyright © 2025 – Hailo Technologies Ltd.

(continued from previous page)

```
Element Properties:
 name        : The name of the object
              flags: readable, writable
              String. Default: "hailoexportzmq0"
 parent      : The parent of the object
              flags: readable, writable
              Object of type "GstObject"
 qos         : Handle Quality-of-Service events
              flags: readable, writable
              Boolean. Default: false
 address     : Address to bind the socket to.
              flags: readable, writable, changeable only in NULL or READY state
              String. Default: "tcp://*:5555"
```

5.17. Hailo Import ZMQ

5.17.1. Overview

HailoImportZMQ is an element which provides an access point in the pipeline to import *HailoObjects meta* from a ZMQ socket.
The meta is added to any pre-existing ROI in the buffer, and then the buffer continues onwards in the pipeline.

Parameters

The HailoImportZMQ element allows the user to change the input port/protocol. The default is *tcp://localhost:5555*. Currently only SUB behvaior (PUB/SUB) is supported.

5.17.2. Hierarchy

```
GObject
+----GInitiallyUnowned
   +----GstObject
      +----GstElement
         +----GstBaseTransform
            +----GstHailoImportZMQ

Pad Templates:
 SRC template: 'src'
  Availability: Always
  Capabilities:
   ANY

 SINK template: 'sink'
  Availability: Always
  Capabilities:
   ANY

Element has no clocking capabilities.
Element has no URI handling capabilities.

Pads:
 SINK: 'sink'
  Pad Template: 'sink'
```

(continues on next page)

 Confidential and Proprietary | Copyright © 2025 – Hailo Technologies Ltd.

(continued from previous page)

```
 SRC: 'src'
  Pad Template: 'src'

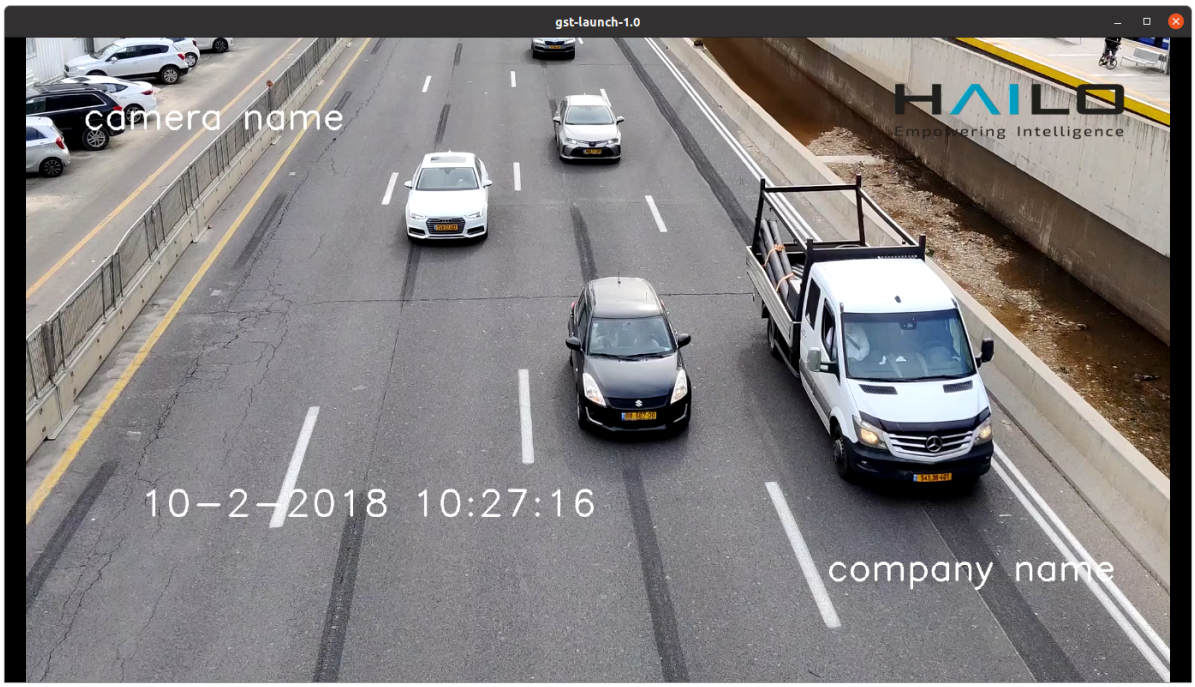
Element Properties:
 name        : The name of the object
             flags: readable, writable
             String. Default: "hailoimportzmq0"
 parent      : The parent of the object
             flags: readable, writable
             Object of type "GstObject"
 qos         : Handle Quality-of-Service events
             flags: readable, writable
             Boolean. Default: false
 address     : Address to bind the socket to.
             flags: readable, writable, changeable only in NULL or READY state
             String. Default: "tcp://localhost:5555"
```

5.18. Hailo On-Screen Display

5.18.1. Overview

HailoOSD is an element which enables the user to draw static text, images, and timestamps on GstBuffers **using the DSP provided in Hailo-15**. By offloading the image blending to the DSP, high performance overlays can be achieved. The DSP also supports transparent blending, allowing HailoOSD to draw image files with transparency.

Currently only NV12 pipelines are supported by HailoOSD.



 Confidential and Proprietary | Copyright © 2025 – Hailo Technologies Ltd.

Parameters

The hailoosd element provides default behavior on what telemetry to draw.
The user can customize the overlay contents via json through the **config-file-path** property. This property accepts the path to a json that follows the following schema:
Any number of entries can be added to the **“image”**, **“text”**, and **“dateTime”** arrays in the json.

```
{
  ”image”: [
    {
      ”id”: ”image_id”,
      ”image_path”: ”/path/to/image”,
      ”width”: 0.2,
      ”height”: 0.13,
      ”x”: 0.76,
      ”y”: 0.05,
      ”z-index”: 1
    }
  ],
  ”dateTime”: [
    {
      ”id”: ”datetime_id”,
      ”font_size”: 2,
      ”text_color”: [0, 0, 255],
      ”x”: 0.1,
      ”y”: 0.7,
      ”z-index”: 3
    }
  ],
  ”text”: [
    {
      ”id”: ”text1_id”,
      ”label”: ”example text 1”,
      ”font_size”: 2,
      ”text_color”: [255, 0, 0],
      ”x”: 0.7,
      ”y”: 0.8,
      ”z-index”: 1
    },
    {
      ”id”: ”text2_id”,
      ”label”: ”example text 2”,
      ”font_size”: 2,
      ”x”: 0.05,
      ”y”: 0.1,
      ”z-index”: 1
    }
  ]
}
```

 Confidential and Proprietary | Copyright © 2025 – Hailo Technologies Ltd.

5.18.2. Hierarchy

```
GObject
+----GInitiallyUnowned
   +----GstObject
      +----GstElement
         +----GstBaseTransform
            +----GstHailoOsd

Pad Templates:
 SINK template: 'sink'
  Availability: Always
  Capabilities:
   video/x-raw
        format: { (string)NV12 }
         width: [ 1, 2147483647 ]
        height: [ 1, 2147483647 ]
      framerate: [ 0/1, 2147483647/1 ]

 SRC template: 'src'
  Availability: Always
  Capabilities:
   video/x-raw
        format: { (string)NV12 }
         width: [ 1, 2147483647 ]
        height: [ 1, 2147483647 ]
      framerate: [ 0/1, 2147483647/1 ]

Element has no clocking capabilities.
Element has no URI handling capabilities.

Pads:
 SINK: 'sink'
  Pad Template: 'sink'
 SRC: 'src'
  Pad Template: 'src'

Element Properties:
 config-file-path    : json config file path
            flags: readable, writable, changeable only in NULL or READY state
            String. Default: "NULL"
 name          : The name of the object
            flags: readable, writable, 0x2000
            String. Default: "hailoosd0"
 parent        : The parent of the object
            flags: readable, writable, 0x2000
            Object of type "GstObject"
 qos           : Handle Quality-of-Service events
            flags: readable, writable
            Boolean. Default: false
```

 Confidential and Proprietary | Copyright © 2025 – Hailo Technologies Ltd.

5.19. Hailo Upload

5.19.1. Overview

`HailoUpload` is an element specifically designed for Hailo-15 system. It is responsible for transformation between memory spaces.

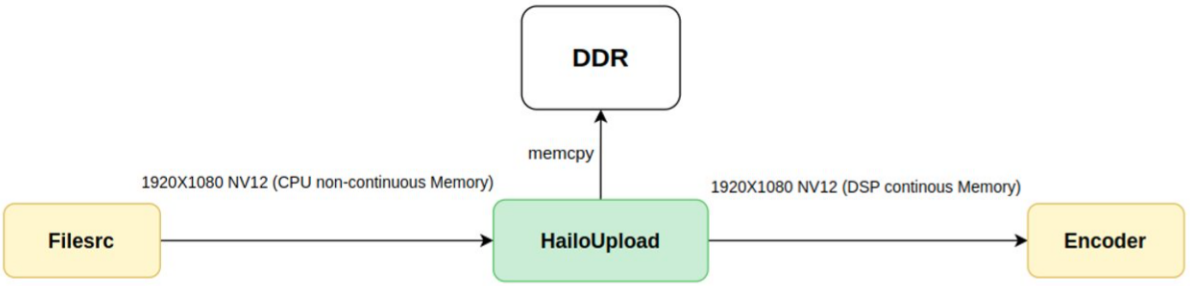
The element can be described by the following steps:

- Request an address (pointer) to a physically contiguous buffer from the kernel at a specifc size (determined by the format and resolution of the frame).
- Perform a memory copy to the target memory.
- Hold the address in a buffer pool - to free the buffer when its reference count is zero.
- Ensure that the output buffer is physically contiguous in memory and also virtually contiguous.

DSP and Encoder require physically contiguous buffers. It is recommended to make sure that a buffer is contiguous in memory, before being sent to an element that uses DSP- it is essential to avoid memcopies. (Like in *HailoCropper*). In Encoder - contiguous memory is mandatory. (see Media Library documentation for more information).

As an example, file source allocates non-contiguous buffers, meaning a hailoupload is required to use `hailoh265enc`:

```
gst-launch-1.0 filesrc location=video.raw name=src_0 ! rawvideoparse format=rgb
↪width=1920 height=1080 ! hailoupload ! hailoh265enc ! fakesink
```



Another example is to use hailoupload before hailonet, to make sure that the input buffer is virtually contiguous in memory:

```
gst-launch-1.0 v4l2src device=/dev/video0 io-mode=mmap ! video/x-raw,format=NV12,
↪width=1920,height=1080 ! hailoupload ! hailonet hef-path=yolov5m_wo_spp_60p_nv12.
↪hef ! fakesink
```

`HailoUpload` inherits from `HailoDspBaseTransform` which is responsible for managing the allocation queries and buffer pool.

5.19.2. Hierarchy

```
  GObject
↪
+----GInitiallyUnowned
↪
  +----GstObject
↪
      +----GstElement
↪
        +----GstBaseTransform
↪
```

(continues on next page)

 Confidential and Proprietary | Copyright © 2025 – Hailo Technologies Ltd.

(continued from previous page)

```
            +----GstHailoDspBaseTransform
  ↪
              +----GstHailoUpload
  ↪

  ↪
Pad Templates:
  ↪
SINK template: 'sink'
  ↪
  Availability: Always
  ↪
  Capabilities:
  ↪
  ANY
  ↪

  ↪
SRC template: 'src'
  ↪
  Availability: Always
  ↪
  Capabilities:
  ↪
  ANY
  ↪

  ↪
Element has no clocking capabilities.
  ↪
Element has no URI handling capabilities.
  ↪

  ↪
Pads:
  ↪
SINK: 'sink'
  ↪
  Pad Template: 'sink'
  ↪
SRC: 'src'
  ↪
  Pad Template: 'src'
  ↪

  ↪
Element Properties:
  ↪
name         : The name of the object
  ↪
             flags: readable, writable, 0x2000
  ↪
             String. Default: ”hailoupload0”
  ↪
parent       : The parent of the object
  ↪
             flags: readable, writable, 0x2000
  ↪
             Object of type ”GstObject”
  ↪
pool-size      : Size of the pool of buffers to use for cropping. Default 10
  ↪
```

(continues on next page)

 Confidential and Proprietary | Copyright © 2025 – Hailo Technologies Ltd.

(continued from previous page)

```
              flags: readable, writable, changeable only in NULL or READY
 ↪state
              Unsigned Integer. Range: 1 - 2147483647 Default: 10
 ↪
qos           : Handle Quality-of-Service events
 ↪
              flags: readable, writable
 ↪
              Boolean. Default: false
```

5.20. Hailo Gray to NV12

5.20.1. Overview

Hailograytonv12 is an element that replaces the GRAY8 buffer with an NV12 buffer that is stored in the metadata of the GRAY8 buffer. It is a mirror to the hailonv12togray element.It is worth noting that this process does not involve any buffer copies, enabling high performance.This element is particularly useful for NV12 pipelines that want to use a GRAY8 formatted HEF.In such cases, the hailonv12togray element should be placed before the hailonet element, and the hailograytonv12 element should be placed after the hailonet element, to retrieve the original NV12 buffer.

5.20.2. Hierarchy

```
GObject
+----GInitiallyUnowned
   +----GstObject
      +----GstElement
         +----GstBaseTransform
            +----GstHailograytonv12

Pad Templates:
 SRC template: 'src'
  Availability: Always
  Capabilities:
   video/x-raw
        format: { (string)NV12 }
         width: [ 1, 2147483647 ]
        height: [ 1, 2147483647 ]
     framerate: [ 0/1, 2147483647/1 ]

 SINK template: 'sink'
  Availability: Always
  Capabilities:
   video/x-raw
        format: { (string)GRAY8 }
         width: [ 1, 2147483647 ]
        height: [ 1, 2147483647 ]
     framerate: [ 0/1, 2147483647/1 ]

Element has no clocking capabilities.
Element has no URI handling capabilities.

Pads:
 SINK: 'sink'
  Pad Template: 'sink'
 SRC: 'src'
  Pad Template: 'src'
```

(continues on next page)

 Confidential and Proprietary | Copyright © 2025 – Hailo Technologies Ltd.

(continued from previous page)

```
Element Properties:
 name        : The name of the object
               flags: readable, writable
               String. Default: "hailograytonv12-0"
 parent      : The parent of the object
               flags: readable, writable
               Object of type "GstObject"
 qos         : Handle Quality-of-Service events
               flags: readable, writable
               Boolean. Default: false
```

5.21. Hailo NV12 to Gray

5.21.1. Overview

Hailonv12togray is an element that performs conversion of NV12 format to GRAY8 format, while preserving the initial NV12 buffer as the metadata of the resulting GRAY8 buffer. It is a mirror to the hailograytonv12 element.It is worth noting that this process does not involve any buffer copies, enabling high performance.This element is particularly useful for NV12 pipelines that want to use a GRAY8 formatted HEF.In such cases, the hailonv12togray element should be placed before the hailonet element, and the hailograytonv12 element should be placed after the hailonet element, to retrieve the original NV12 buffer.

5.21.2. Hierarchy

```
GObject
+----GInitiallyUnowned
   +----GstObject
      +----GstElement
         +----GstBaseTransform
            +----GstHailonv12togray

Pad Templates:
 SRC template: 'src'
  Availability: Always
  Capabilities:
   video/x-raw
        format: { (string)GRAY8 }
         width: [ 1, 2147483647 ]
        height: [ 1, 2147483647 ]
     framerate: [ 0/1, 2147483647/1 ]

 SINK template: 'sink'
  Availability: Always
  Capabilities:
   video/x-raw
        format: { (string)NV12 }
         width: [ 1, 2147483647 ]
        height: [ 1, 2147483647 ]
     framerate: [ 0/1, 2147483647/1 ]

Element has no clocking capabilities.
Element has no URI handling capabilities.

Pads:
 SINK: 'sink'
```

(continues on next page)

 Confidential and Proprietary | Copyright © 2025 – Hailo Technologies Ltd.

(continued from previous page)

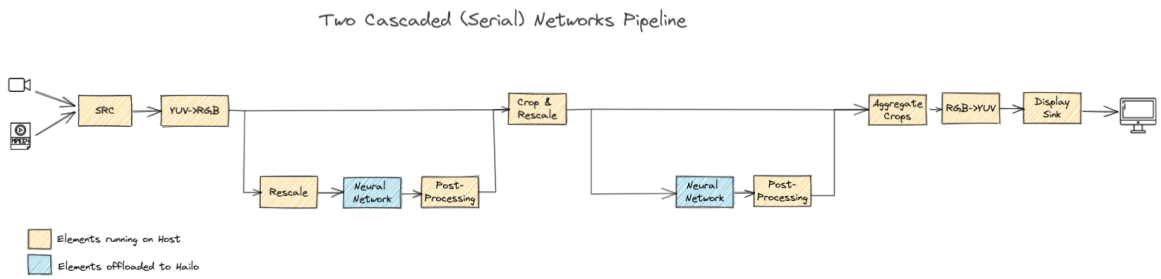
```
  Pad Template: 'sink'
 SRC: 'src'
  Pad Template: 'src'

Element Properties:
 name        : The name of the object
             flags: readable, writable
             String. Default: "hailonv12togray0"
 parent      : The parent of the object
             flags: readable, writable
             Object of type "GstObject"
 qos         : Handle Quality-of-Service events
             flags: readable, writable
             Boolean. Default: false
```

 Confidential and Proprietary | Copyright © 2025 – Hailo Technologies Ltd.

6. Pipelines

6.1. Cascaded Networks Structure



This section provides a drill-down into the template of our cascaded network pipelines with a focus on explaining the `GStreamer` pipeline.

6.1.1. Example Pipeline

First, it is necessary to declare two sub-pipelines, these pipelines are derived from our *Single network template*

```
NETWORK_ONE_PIPELINE="videoscale qos=false ! \
  queue leaky=no max-size-buffers=3 max-size-bytes=0 max-size-time=0 ! \
  hailonet net-name=$NETWORK_ONE_NAME \
  hef-path=$hef_path is-active=true qos=false ! \
  queue leaky=no max-size-buffers=3 max-size-bytes=0 max-size-time=0 ! \
  hailofilter so-path=$NETWORK_ONE_SO function-name=$NETWORK_ONE_FUNC_NAME
↪qos=false ! \
  queue leaky=no max-size-buffers=3 max-size-bytes=0 max-size-time=0"

NETWORK_TWO_PIPELINE="queue leaky=no max-size-buffers=3 max-size-bytes=0 max-size-
↪time=0 ! \
  hailonet net-name=$NETWORK_TWO_NAME \
  hef-path=$hef_path is-active=true qos=false ! \
  queue leaky=no max-size-buffers=3 max-size-bytes=0 max-size-time=0 ! \
  hailofilter so-path=$NETWORK_TWO_SO function-name=$NETWORK_TWO_FUNC_NAME
↪qos=false ! \
  queue leaky=no max-size-buffers=3 max-size-bytes=0 max-size-time=0"
```

Next, insert them into the full pipeline:

```
gst-launch-1.0 \
  $source_element ! \
  tee name=t \
  \
  hailomuxer name=hmux \
  \
  t. ! queue leaky=no max-size-buffers=3 max-size-bytes=0 max-size-time=0 ! hmux. \
  t. ! $NETWORK_ONE_PIPELINE ! hmux. \
  hmux. ! queue leaky=no max-size-buffers=3 max-size-bytes=0 max-size-time=0 ! \
  hailocropper internal-offset=$internal_offset name=cropper \
  \
  hailoaggregator name=agg \
  \
  cropper. ! queue leaky=no max-size-buffers=3 max-size-bytes=0 max-size-time=0 !
↪agg. \
  \
  cropper. ! $NETWORK_TWO_PIPELINE ! agg. \
```

(continues on next page)

 Confidential and Proprietary | Copyright © 2025 – Hailo Technologies Ltd.

(continued from previous page)

```
  agg. ! queue leaky=no max-size-buffers=3 max-size-bytes=0 max-size-time=0 ! \
  hailooverlay ! \
  queue leaky=no max-size-buffers=3 max-size-bytes=0 max-size-time=0 ! videoconvert
↪! \
  fpsdisplaysink video-sink=xvimagesink name=hailo_display sync=false text-
↪overlay=false ${additional_parameters}
```

To clarify the user's understanding pipeline is described section by section:

```
$source_element ! \
tee name=t \
\
hailomuxer name=hmux \
\
t. ! queue leaky=no max-size-buffers=3 max-size-bytes=0 max-size-time=0 ! hmux. \
t. ! $NETWORK_ONE_PIPELINE ! hmux. \
hmux. ! queue leaky=no max-size-buffers=3 max-size-bytes=0 max-size-time=0 ! \
```

This section of the pipeline is taken from our: *Single network template*. in brief: run an inference while keeping the original image resolution.

```
hailocropper internal-offset=$internal_offset name=cropper hailoaggregator name=agg
```

Hailocropper Hailocropper splits the pipeline into 2 branches: the first passes the original frame while the other passes crops from that original frame one at a time. The hailocropper chooses what crops to take from the original frame based on an so file that you can provide (typically a set of detections from a prior hailofilter postprocess). The hailocropper also scales all crops based on caps negotiation. This way if a hailonet is placed after the cropper (such as in this example), then the hailocropper will scale all crops to that hailonet's input network size. The hailoaggregator gets the original frame and then knows to wait for all related cropped buffers: aggregating all metadata to the original frame, then sending it forward.

```
cropper. ! queue leaky=no max-size-buffers=3 max-size-bytes=0 max-size-time=0 ! agg.
↪\
```

The first part of the cascading network pipeline, passes the original frame on the bypass pads to hailoaggregator.

```
 cropper. ! $NETWORK_TWO_PIPELINE ! agg.

The second part of the cascading network pipeline, performs a second network on all
↪objects, which are cropped and scaled to the needed resolution by the HEF in the
↪hailonet.
```

```
agg. ! queue leaky=no max-size-buffers=3 max-size-bytes=0 max-size-time=0 ! \
hailooverlay ! \
```

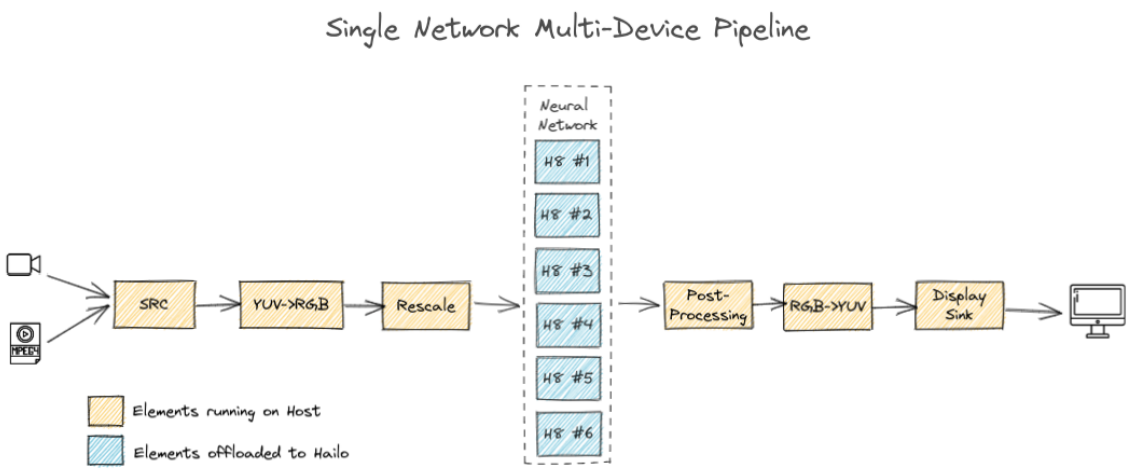
Aggregates all objects to the original frame, and draws them over the frame using the *hailooverlay* with specific drawing function.

```
queue leaky=no max-size-buffers=3 max-size-bytes=0 max-size-time=0 ! videoconvert ! \
fpsdisplaysink video-sink=xvimagesink name=hailo_display sync=false text-
↪overlay=false
```

Display the final image using `fpsdisplaysink`.

 Confidential and Proprietary | Copyright © 2025 – Hailo Technologies Ltd.

6.2. Single Network Multi-Device Pipeline Structure



This page provides a drill-down into the template of our multi-device pipelines with a focus on explaining the `GStreamer` pipeline.

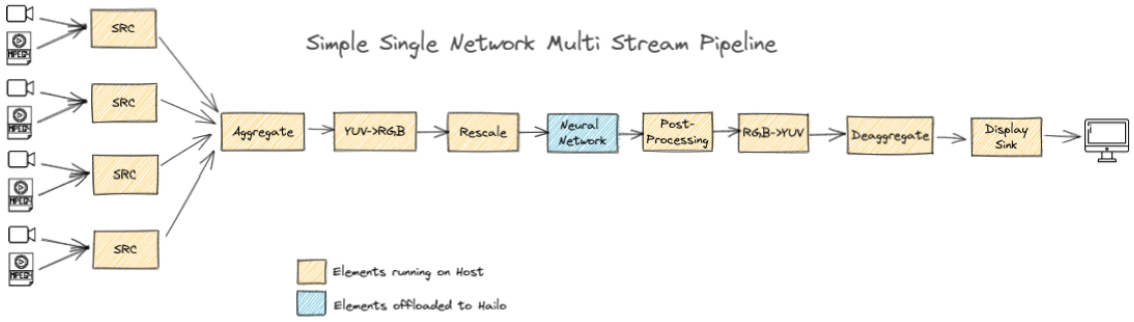
6.2.1. Example Pipeline

```
gst-launch-1.0 \
  $source_element ! videoconvert ! \
  videoscale ! \
  queue leaky=no max-size-buffers=30 max-size-bytes=0 max-size-time=0 ! \
  hailonet hef-path=$hef_path device-count=$device_count is-active=true ! \
  queue leaky=no max-size-buffers=30 max-size-bytes=0 max-size-time=0 ! \
  hailofilter function-name=$network_name so-path=$postprocess_so qos=false ! \
  queue leaky=no max-size-buffers=30 max-size-bytes=0 max-size-time=0 ! \
  hailooverlay qos=false ! \
  queue leaky=no max-size-buffers=30 max-size-bytes=0 max-size-time=0 ! \
  videoconvert ! \
  fpsdisplaysink video-sink=$video_sink_element name=hailo_display sync=$sync_
↪pipeline text-overlay=false
```

This pipeline is based on-top of our *single network pipeline*. The number of physical devices to utilize is set via the `hailonet` device-count property (The default is 4 in this app)

 Confidential and Proprietary | Copyright © 2025 – Hailo Technologies Ltd.

6.3. Multi Stream Pipeline Structure



This page provides a drill-down into the template of our multi stream pipelines with a focus on explaining the `GStreamer` pipeline.

6.3.1. Example Pipeline

The first stage is to create the pipeline sources.

```
n=4
sources=''

for ((n = $start_index; n < $num_of_src; n++)); do
  sources+="$source_element ! \
      queue name=hailo_preprocess_q_$n leaky=no max-size-buffers=5 max-size-
↪bytes=0 max-size-time=0 ! \
      videoconvert ! videoscale ! fun.sink_$n \
      sid.src_$n ! queue name=comp_q_$n leaky=downstream max-size-buffers=30 \
      max-size-bytes=0 max-size-time=0 ! comp.sink_$n "
done
```

Each source is a sub-pipeline

```
pipeline="gst-launch-1.0 \
    funnel name=fun ! \
    queue name=hailo_pre_infer_q_0 leaky=no max-size-buffers=30 max-size-bytes=0
↪max-size-time=0 ! \
    hailonet hef-path=$HEF_PATH is-active=true ! \
    queue name=hailo_postprocess0 leaky=no max-size-buffers=30 max-size-bytes=0
↪max-size-time=0 ! \
    hailofilter so-path=$POSTPROCESS_SO qos=false ! \
    queue name=hailo_draw0 leaky=no max-size-buffers=30 max-size-bytes=0 max-size-
↪time=0 ! \
    hailooverlay ! \
    streamiddemux name=sid \
    compositor name=comp start-time-selection=0 $compositor_locations ! \
    queue name=hailo_video_q_0 leaky=no max-size-buffers=30 max-size-bytes=0 max-
↪size-time=0 ! \
    videoconvert ! queue name=hailo_display_q_0 leaky=no max-size-buffers=30 max-
↪size-bytes=0 max-size-time=0 ! \
    $video_sink_element name=hailo_display sync=false \
    $sources
```

They can then be combined together

- `funnel` takes multiple input sinks and outputs one source. an N-to-1 funnel that attaches a streamid to each stream, can later be used to demux back into separate streams. this lets you queue frames from multiple

 Confidential and Proprietary | Copyright © 2025 – Hailo Technologies Ltd.

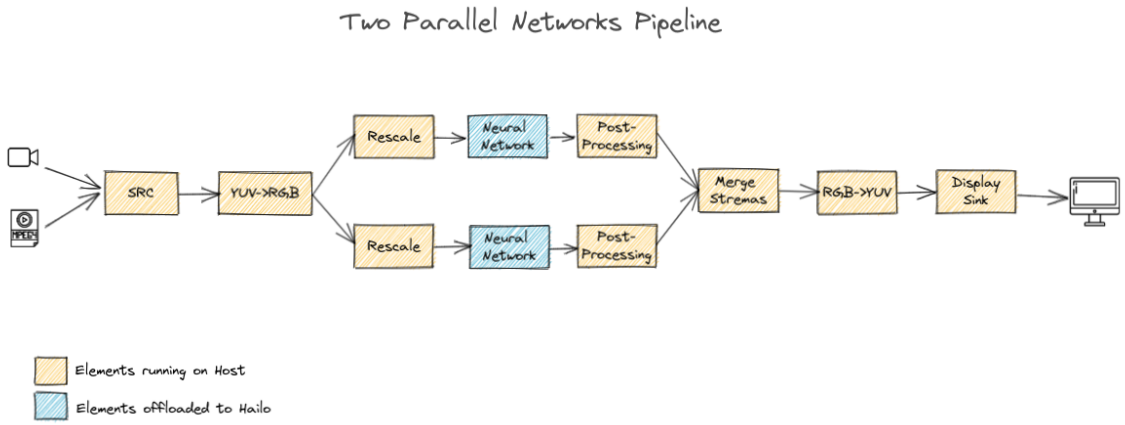
streams to send to the hailo device one at a time.

- `streamiddemux` a reverse to the funnel. It is a 1-to-N demuxer that splits a serialized stream based on stream id to multiple outputs.
- `compositor` composites pictures from multiple sources. handy for multi-stream/tiling like applications, as it lets you input many streams and draw them all together as a grid.

6.4. Parallel Networks Structure

This page provides a drill-down into the template of our parallel networks pipeline with a focus on explaining the `GStreamer` pipeline.

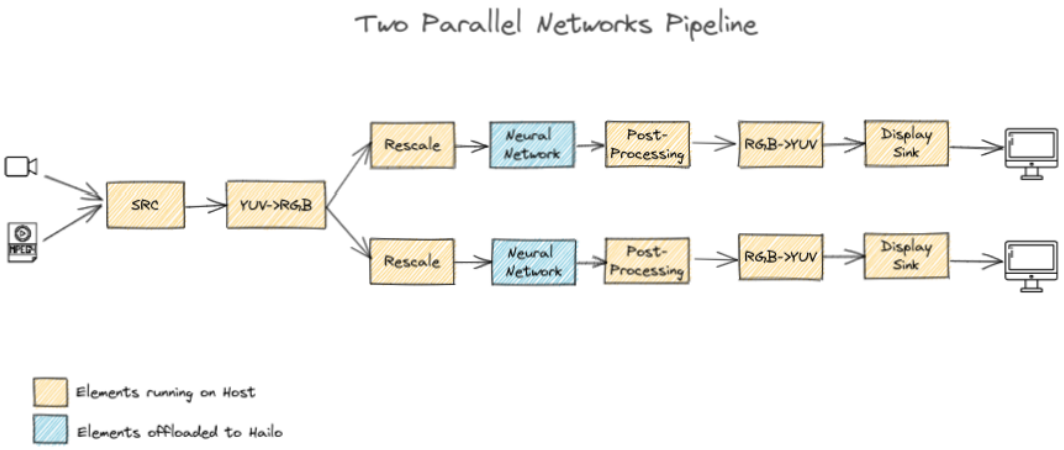
6.4.1. Example Pipeline of One Display Multi Source




```
gst-launch-1.0 \
  $source_element ! videoconvert ! \
  queue name=hailo_pre_split leaky=no max-size-buffers=30 max-size-bytes=0 max-
↪size-time=0 ! \
  tee name=splitter \
  hailomuxer name=hailomuxer ! \
  queue name=hailo_draw0 leaky=no max-size-buffers=30 max-size-bytes=0 max-size-
↪time=0 ! \
  hailooverlay qos=false ! \
  splitter. ! queue name=hailo_pre_infer_q_1 leaky=no max-size-buffers=5 max-size-
↪bytes=0 max-size-time=0 ! \
  hailonet hef-path=$NETWORK_ONE_HEF_PATH is-active=true ! \
  queue name=hailo_postprocess0 leaky=no max-size-buffers=30 max-size-bytes=0 max-
↪size-time=0 ! \
  hailofilter so-path=$NETWORK_ONE_POSTPROCESS_SO function-name=$NETWORK_ONE_
↪POSTPROCESS_FUNCTION_NAME qos=false ! hailomuxer. \
  splitter. ! queue name=hailo_pre_infer_q_0 leaky=no max-size-buffers=5 max-size-
↪bytes=0 max-size-time=0 ! \
  hailonet hef-path=$NETWORK_TWO_HEF_PATH is-active=true ! \
  queue name=hailo_postprocess1 leaky=no max-size-buffers=30 max-size-bytes=0 max-
↪size-time=0 ! \
  hailofilter so-path=$NETWORK_TWO_POSTPROCESS_SO function-name=$NETWORK_TWO_
↪POSTPROCESS_FUNCTION_NAME qos=false ! hailomuxer. \
  queue leaky=no max-size-buffers=30 max-size-bytes=0 max-size-time=0 ! \
  fpsdisplaysink video-sink=$video_sink_element name=hailo_display sync=false text-
↪overlay=false \
```

 Confidential and Proprietary | Copyright © 2025 – Hailo Technologies Ltd.

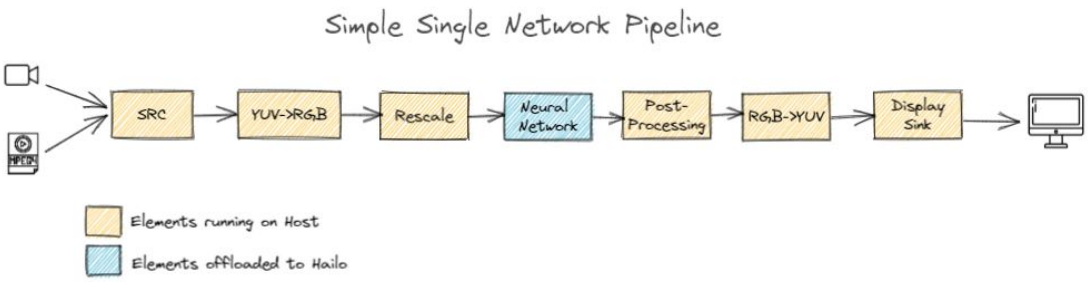
6.4.2. Example Pipeline of Two Displays




```
gst-launch-1.0 \
  $source_element ! videoconvert !
  queue leaky=no max-size-buffers=30 max-size-bytes=0 max-size-time=0 ! \
  tee name=t ! queue ! videoscale ! \
  queue leaky=no max-size-buffers=30 max-size-bytes=0 max-size-time=0 ! \
  hailonet hef-path=$hef_path is-active=true net-name=$network_one_name ! \
  queue leaky=no max-size-buffers=30 max-size-bytes=0 max-size-time=0 ! \
  hailofilter so-path=$network_one_so qos=false ! videoconvert ! \
  fpsdisplaysink video-sink=$video_sink_element name=hailo_display sync=false text-
↪overlay=false \
  t. ! \
  videoscale ! queue ! \
  hailonet hef-path=$hef_path is-active=true net-name=$network_two_name ! \
  queue leaky=no max-size-buffers=30 max-size-bytes=0 max-size-time=0 ! \
  hailofilter so-path=$network_two_so function-name=mobilenet_ssd_merged
↪qos=false ! \
  queue leaky=no max-size-buffers=30 max-size-bytes=0 max-size-time=0 ! \
  hailooverlay ! videoconvert ! \
  fpsdisplaysink video-sink=$video_sink_element name=hailo_display2 sync=false
↪text-overlay=false ${additional_parameters}
```

This pipeline is based on-top of the *single network pipeline*, the modification amounts to using the GStreamer built in the tee element.

6.5. Single Network Pipeline Structure



This page provides a drill-down into the template of our single network pipelines with a focus on explaining the

 Confidential and Proprietary | Copyright © 2025 – Hailo Technologies Ltd.

GStreamer pipeline.

6.5.1. Example Pipeline

```
gst-launch-1.0 \
  filesrc location=$video_device ! decodebin ! videoconvert ! \
  videoscale ! \
  queue leaky=no max-size-buffers=30 max-size-bytes=0 max-size-time=0 ! \
  hailonet hef-path=$hef_path is-active=true ! \
  queue leaky=no max-size-buffers=30 max-size-bytes=0 max-size-time=0 ! \
  hailofilter function-name=yolov5 so-path=$POSTPROCESS_SO qos=false ! \
  queue leaky=no max-size-buffers=30 max-size-bytes=0 max-size-time=0 ! \
  hailooverlay ! \
  videoconvert ! \
  fpsdisplaysink video-sink=xvimagesink name=hailo_display sync=true text-
↪overlay=false
```

The pipeline functionality will be explained section by section:

```
filesrc location=$video_device ! decodebin ! videoconvert !
```

Specifies the location of the video used, then decodes and converts to the required format.

```
 videoscale ! \

Re-scale the video dimensions to fit the input of the network. The video scale finds out
↪the requires width and height using caps negotiation with ``hailonet``.
```

```
queue leaky=no max-size-buffers=30 max-size-bytes=0 max-size-time=0 ! \
```

Before sending the frames into the `hailonet` element, set a queue so no frames are lost (Read more about queues here)

```
hailonet hef-path=$hef_path is-active=true ! \
queue leaky=no max-size-buffers=30 max-size-bytes=0 max-size-time=0 ! \
```

Performs the inference on the Hailo-8 device.

```
hailofilter function-name=$POSTPROCESS_NAME so-path=$POSTPROCESS_SO qos=false ! \
queue name=hailo_draw0 leaky=no max-size-buffers=30 max-size-bytes=0 max-size-
↪time=0 ! \
hailooverlay ! \
```

hailofilter performs a given post-process to extract the objects. The following *hailooverlay* element is able to draw standard `HailoObjects` to the buffer.

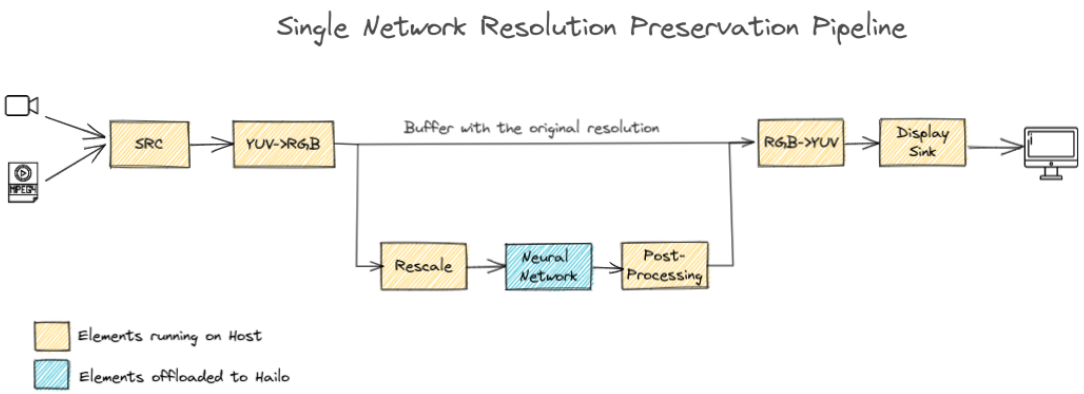
```
videoconvert ! \
fpsdisplaysink video-sink=xvimagesink name=hailo_display sync=true text-
↪overlay=false
```

Apply the final convert to let GStreamer utilize the format required by the `fpsdisplaysink` element

 Confidential and Proprietary | Copyright © 2025 – Hailo Technologies Ltd.

6.5.2. Example Pipeline with Resolution Preservation

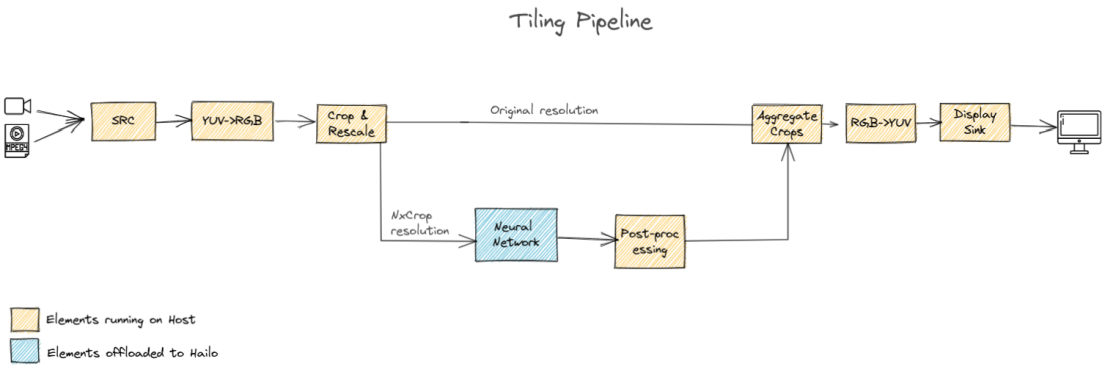
Using this template the source resolution would be preserved, this is an extension to our *Example pipeline*



An example for pipelines who preserve the original resolution:

```
gst-launch-1.0 \
  $source_element ! videoconvert ! \
  tee name=t hailomuxer name=mux \
  t. ! queue leaky=no max-size-buffers=30 max-size-bytes=0 max-size-time=0 ! mux. \
  t. ! videoscale ! \
  queue leaky=no max-size-buffers=30 max-size-bytes=0 max-size-time=0 ! \
  hailonet hef-path=$hef_path is-active=true ! \
  queue leaky=no max-size-buffers=30 max-size-bytes=0 max-size-time=0 ! \
  hailofilter function-name=$network_name so-path=$postprocess_so qos=false ! mux. \
  mux. ! queue leaky=no max-size-buffers=30 max-size-bytes=0 max-size-time=0 ! \
  hailooverlay ! queue leaky=no max-size-buffers=30 max-size-bytes=0 max-size-
↪time=0 ! \
  videoconvert ! \
  fpsdisplaysink video-sink=$video_sink_element name=hailo_display sync=$sync_
↪pipeline text-overlay=false
```

6.5.3. Example Pipeline Single Network with Tiling



Tiling introduces two new elements:

- `hailotilecropper` **which splits the frame into tiles by separating the frame into rows and columns** (given as parameters to the element).
- `hailotileaggregator` which aggregates the cropped tiles and stitches them back to the original resolution.

 Confidential and Proprietary | Copyright © 2025 – Hailo Technologies Ltd.

7. Tools

7.1. Dot Visulaizer

7.1.1. How to use the tool?

```
# Run `gst_set_debug`
$ gst_set_debug

# Optional, adjust the TRACERS
# Important: Keep the `graphic` tracer
$ export GST_TRACERS="proctime;interlatency;framerate;queuelevel;graphic"

# Run the application
$ gst-launch-1.0 ...

# Prepare the tracers dir
$ gst_prepare_tracers_dir
# output should be in the following format:
Splited tracers dir: /local/workspace/tappas/tappas_traces_<date>

# Run the tool
$ hailo_tappas_dot_visualizer /local/workspace/tappas/tappas_traces_<date>
```

7.2. Cross-compile Hailo's GStreamer plugins

7.2.1. Overview

Hailo recommended method at the moment for cross-compilation is using Yocto SDK (aka Toolchain). We provide wrapper scripts whose only requirement is a Yocto toolchain to make this as easy as possible.

7.2.2. Preparations

In order to cross-compile you need to run TAPPAS container on a X86 machine and copy the Yocto toolchain to the container.

Toolchain

What is Toolchain?

A standard Toolchain consists of the following:

- Cross-Development Toolchain: This toolchain contains a compiler, debugger, and various miscellaneous tools.
- Libraries, Headers, and Symbols: The libraries, headers, and symbols are specific to the image (i.e. they match the image).
- Environment Setup Script: This *.sh file, once run, sets up the cross-development environment by defining variables and preparing for Toolchain use.

You can use the standard Toolchain to independently develop and test code that is destined to run on some target machine.

 Confidential and Proprietary | Copyright © 2025 – Hailo Technologies Ltd.

What should I add to my image?

For this example we will add the recipes to an NXP-IMX based image

Must add

```
# GStreamer plugins
IMAGE_INSTALL_append += "          \
  imx-gst1.0-plugin            \
  gstreamer1.0-plugins-bad-videoparsersbad \
  gstreamer1.0-plugins-good-video4linux2  \
  gstreamer1.0-plugins-base        \
"

# Opencv requirements for the hailo gstreamer plugin's postprocess
CORE_IMAGE_EXTRA_INSTALL += "   \
   libopencv-core-dev     \
   libopencv-highgui-dev    \
"
```

Nice to add

```
# GStreamer plugins
IMAGE_INSTALL_append += "          \
  gstreamer1.0-python           \
  gst-instruments             \
"

# Enable trace hooks for GStreamer
PACKAGECONFIG_append_pn-gstreamer1.0 += " gst-tracer-hooks"
```

Prepare the Toolchain

In order to generate a Yocto toolchain, use this following command

```
# Generate the Toolchain
bitbake -c do_populate_sdk <image name>
```

7.2.3. Components

TAPPAS

This script cross-compiles TAPPAS components. This script first unpack and installs the toolchain (if not already installed), and then cross-compiles TAPPAS core/.

 Confidential and Proprietary | Copyright © 2025 – Hailo Technologies Ltd.

Flags

```
$ ./cross_compile_tappas.py --help
usage: cross_compile_tappas.py [-h] [--remote-machine-ip REMOTE_MACHINE_IP] [--
↪build-lib {all,apps,plugins,libs,tracers}] [--clean-build-dir]
              [--install-to-rootfs]
              {aarch64,armv7l,armv7lhf,armv8a} {imx8,hailo15} {debug,release}
↪toolchain_dir_path

Cross-compile TAPPAS

positional arguments:
{aarch64,armv7l,armv7lhf,armv8a}
            Arch to compile to
{imx8,hailo15}  Target platform to compile to
{debug,release}     Build and compilation type
toolchain_dir_path  Toolchain directory path

optional arguments:
-h, --help       show this help message and exit
--remote-machine-ip REMOTE_MACHINE_IP
            remote machine ip
--build-lib {all,apps,plugins,libs,tracers}
            Build a specific tappas lib target (default all)
--clean-build-dir   Delete previous build cache (default false)
--install-to-rootfs  Install to rootfs (default false)
```

Example

Run the compilation script

Note: In this example we assume that the toolchain is located under toolchain-raw/hailo-dartmx8m-dunfell-aarch64-toolchain

```
$ ./cross_compile_tappas.py aarch64 imx8 debug toolchain
INFO:./cross_compile_tappas.py:Building hailofilter plugin and post processes
INFO:./cross_compile_tappas.py:Running Meson build.
INFO:./cross_compile_tappas.py:Running Ninja command.
```

Check the output directory

```
$ ls aarch64-gsthailotools-build/
build.ninja compile_commands.json config.h libs meson-info meson-logs meson-
↪private plugins
```

`libgsthailotools.so` is stored under libs

```
$ ls aarch64-gsthailotools-build/plugins/*.so
libgsthailotools.so
```

And the post-processes are stored under plugins

```
$ ls aarch64-gsthailotools-build/libs/*.so
libcenterpose_post.so libmobilenet_ssd_post.so
libclassification.so  libsegmentation_draw.so
libdebug.so       libyolo_post.so
libdetection_draw.so
```

 Confidential and Proprietary | Copyright © 2025 – Hailo Technologies Ltd.

7.2.4. Copy the cross-compiled files

Find out where the GStreamer plugins are stored in your embedded device by running the following command:

```
gst-inspect-1.0 filesrc | grep Filename | awk '{print $2}' | xargs dirname
```

Copy libgsthailo.so + libgsthailotools.so to the path found out above. Copy the post-processes so files under libs to the embedded device under /usr/lib/hailo-post-processes (create the directory if it does not exist)

Run gst-inspect-1.0 hailo and gst-inspect-1.0 hailotools and make sure that no error raises

 Confidential and Proprietary | Copyright © 2025 – Hailo Technologies Ltd.

8. Scripts

8.1. Using record_perf script to get records from tappas

1) Enter core/hailo/meson.build
2) Add to the Compiler Arguments:

```
add_global_arguments('-ggdb', language : 'cpp')
```

3) Compile tappas in debug mode
4) Run:

```
./scripts/misc/internals/record_perf/record_perf.sh <time>
```

5) run specific pipeline you want to debug
6) upload the output file to https://flamegraph.com/

 Confidential and Proprietary | Copyright © 2025 – Hailo Technologies Ltd.