

DXNN Streamer (DX-STREAM) User Manual

2026/10/23 (v3.2.0)

DEEPX.ai

ⓒ Copyright 2026 DEEPX All Rights Reserved.

Table of contents

1. DXNN SDK Overview 5
1.1 DEEPX SDK Architecture 5
1.2 DX-STREAM Architecture 6
1.3 DX-STREAM Key Features 7
1.4 DX-STREAM Model Workflow 9
2. DX-STREAM Installation 10
2.1 System Requirements 10
2.2 Install DX-STREAM 11
2.3 Run DX-STREAM 18
3. Elements 21
3.1 Multi-Stream Domain (application/x-dxvideoraw) 21
3.2 DxPreprocess 26
3.3 DxInfer 32
3.4 DxPostprocess 36
3.5 DxTracker 39
3.6 DxOsd 42
3.7 DxGather 45
3.8 DxInputSelector 47
3.9 DxOutputSelector 50
3.10 DxRate 53
3.11 DxMsgConv 56
3.12 DxMsgBroker 60
3.13 DxScale 63
3.14 DxConvert 66
4. Writing Your Own Application 69
4.1 API Migration Notes (v3.1.0) 69
4.2 DX-STREAM Metadata System Overview 70
4.3 Custom Library for Model Inference 73
4.4 Custom Message Convert Library 81
5. Pipeline Example 87
5.1 Single Stream Pipeline 87
5.2 Multi Stream Pipeline 90
5.3 Secondary Inference Pipeline 92
5.4 MsgBroker Pipeline 94

ⓒ Copyright 2026 DEEPX All Rights Reserved.

6. Troubleshooting and FAQ 106
6.1 Debug Logging 106
6.2 Rendering Issues 107
6.3 Display Sink Issues on Raspberry Pi 5 108
6.4 Display Sink Issues on Orange Pi 5 Plus with Debian 12 109
6.5 Buffer Delays in Sink Element 112
6.6 Troubleshooting Message Broker Issues 113
6.7 Build Issues with Meson Installation 117
6.8 Multi-Stream Domain Issues 119
7. Debugging Guide 122
7.1 GST_DEBUG Environment Variable 122
7.2 Element-Specific Debugging 124
7.3 Metadata Debugging 128
7.4 Subsystem Debugging 129
7.5 Common Debugging Scenarios 130
7.6 Best Practices 131
7.7 Tips and Tricks 132
7.8 Additional Resources 133
8. DX-STREAM Agent-Driven Development Guide 134
8.1 Overview 134
8.2 Agents 134
8.3 Skills 135
8.4 Supported AI Tools 136
8.5 Generation Pipeline 142
8.6 User Scenarios 142
8.7 Quick Start 146
8.8 What Gets Created 146
8.9 Pipeline Categories 147
8.10 GStreamer Elements Reference 147
8.11 Pipeline-Specific Validation Rules 148
8.12 Validation Commands 149
8.13 Session Sentinels 149
8.14 Troubleshooting 150
9. Appendix 152
9.1 User Metadata Guide 152
9.2 Performance Evaluation with GstShark 161
9.3 Building DX-STREAM on Orange Pi 5 Plus 167
9.4 Building DX-STREAM on Windows 176

ⓒ Copyright 2026 DEEPX All Rights Reserved.

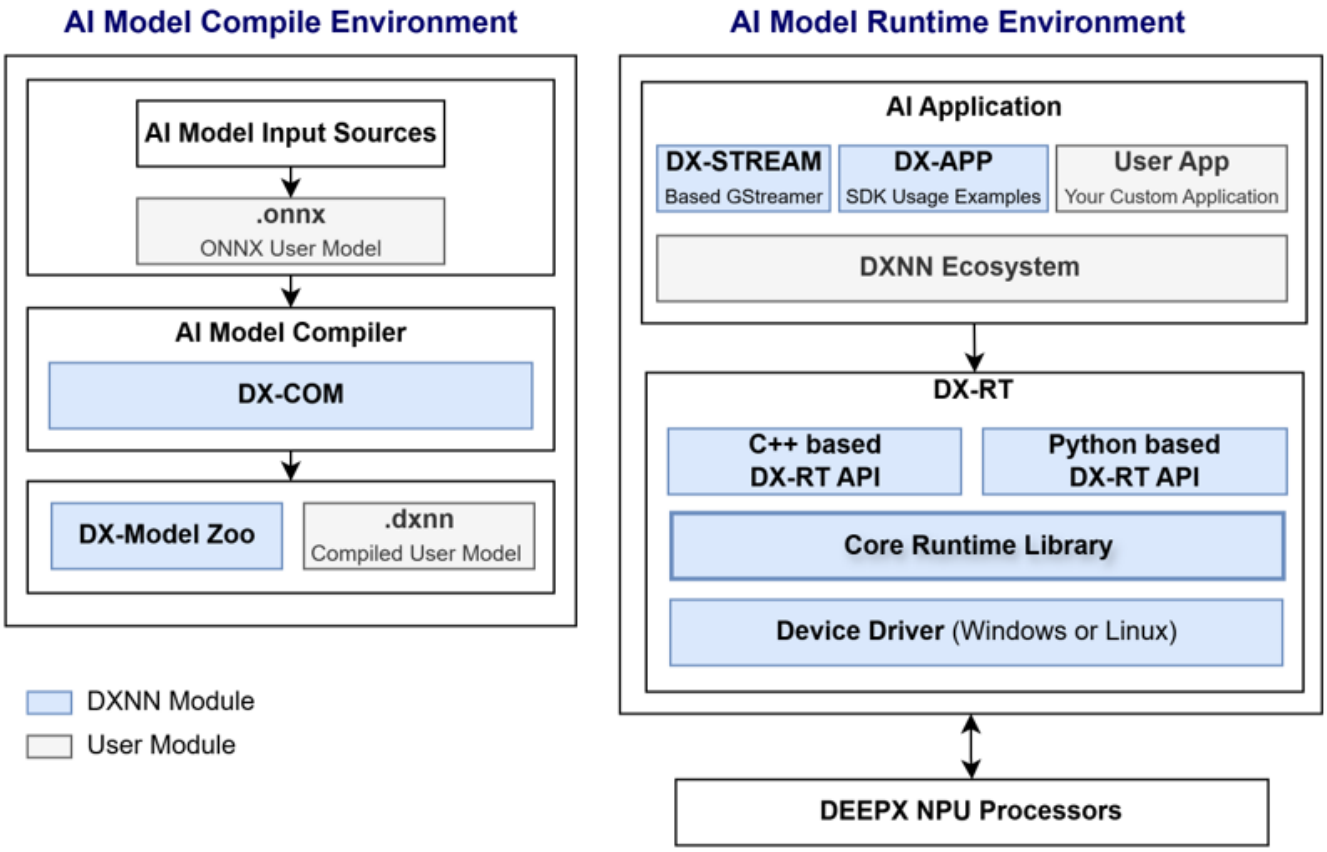
9.5 Third-Party Sample Model Licenses 188
10. RELEASE_NOTES 189
10.1 DX-STREAM v3.2.0 / 2026-10-23 189
10.2 DX-STREAM v3.1.0 / 2026-07-22 190
10.3 DX-STREAM v3.0.1 / 2026-05-06 191
10.4 DX-STREAM v3.0.0 / 2026-04-10 192
10.5 DX-STREAM v2.2.1 / 2026-02-26 193
10.6 DX-STREAM v2.2.0 / 2026-01-16 193
10.7 DX-STREAM v2.1.0 / 2025-11-28 194
10.8 DX-STREAM v2.0.0 / 2025-09-08 196
10.9 DX-STREAM v1.7.0 / 2025-07-23 197

ⓒ Copyright 2026 DEEPX All Rights Reserved.

1. DXNN SDK Overview

This chapter provides an overview of the DEEPX SDK architecture and explains each core component and its role in the AI development workflow.

1.1 DEEPX SDK Architecture



DEEPX SDK is an all-in-one software development platform that streamlines the process of compiling, optimizing, simulating, and deploying AI inference applications on DEEPX NPUs (Neural Processing Units). It provides a complete toolchain, from AI model creation to runtime deployment, optimized for edge and embedded systems, enabling developers to build high-performance AI applications with minimal effort.

DX-COM is the compiler in the DEEPX SDK that converts a pre-trained ONNX model and its associated configuration JSON file into a hardware-optimized `.dxnn` binary for DEEPX NPUs. The ONNX file contains the model structure and weights, while the JSON file defines pre/

© Copyright 2026 DEEPX All Rights Reserved.

post-processing settings and compilation parameters. DX-COM provides a fully compiled `.dxnn` file, optimized for low-latency and high-efficiency inference on DEEPX NPU.

DX-RT is the runtime software responsible for executing `.dxnn` models on DEEPX NPU hardware. DX-RT directly interacts with the DEEPX NPU through firmware and device drivers, using PCIe interface for high-speed data transfer between the host and the NPU, and provides C/C++ and Python APIs for application-level inference control. DX-RT offers a complete runtime environment, including model loading, I/O buffer management, inference execution, and real-time hardware monitoring.

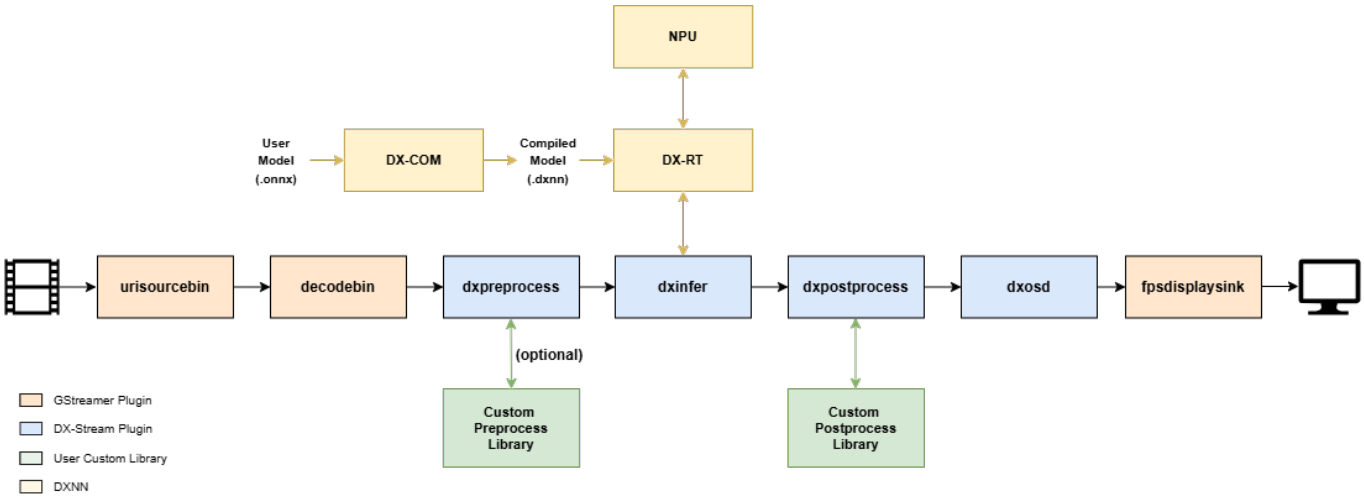
DX-ModelZoo is a curated collection of pre-trained neural network models optimized for DEEPX NPU, designed to simplify AI development for DEEPX users. It includes pre-trained ONNX models, configuration JSON files, and pre-compiled DXNN binaries, allowing developers to rapidly test and deploy applications. DX-ModelZoo also provides benchmark tools for comparing the performance of quantized INT8 models on DEEPX NPU with full-precision FP32 models on CPU or GPU.

DX-STREAM is a custom GStreamer plugin that enables real-time streaming data integration into AI inference applications on DEEPX NPU. It provides a modular pipeline framework with configurable elements for preprocessing, inference, and postprocessing, tailored to vision AI work. DX-STREAM allows developers to build flexible, high-performance applications for use cases such as video analytics, smart cameras, and edge AI systems.

DX-APP is a sample application that demonstrates how to run compiled models on actual DEEPX NPU using DX-RT. It includes ready-to-use code for common vision tasks such as object detection, face recognition, and image classification. DX-APP helps developers quickly set up the runtime environment and serves as a template for building and customizing their own AI applications.

1.2 DX-STREAM Architecture

ⓒ Copyright 2026 DEEPX All Rights Reserved.



DX-STREAM provides dedicated GStreamer elements for AI model inference on DEEPX NPU and includes auxiliary elements for developing AI applications.
For most AI model processing, you can quickly build an inference environment by setting the properties of the elements provided by DX-STREAM. In addition, we support processing through user-defined libraries in pre/post-processing and message conversion for the user's unique model inference and output requirements.
This plugin-based architecture accelerates development and deployment while providing the flexibility needed for a wide range of vision AI applications, including object detection, multi-stream analysis, and real-time tracking.

1.3 DX-STREAM Key Features

The following features highlight what makes DX-STREAM a powerful and flexible framework for Vision AI application development.

Pipeline Modularity

DX-STREAM divides the Vision AI data processing flow into discrete, functional units called Elements. Each element performs a specific task within a GStreamer pipeline, enabling modular, scalable, and reusable pipeline design.

ⓒ Copyright 2026 DEEPX All Rights Reserved.

Elements

- DxPreprocess – Performs input preprocessing for AI inference.
- DxInfer – Executes AI model inference using the DEEPX NPU.
- DxPostprocess – Processes and formats inference output.
- DxTracker – Tracks objects persistently across video frames.
- DxOsd – Overlays inference results onto video frames.
- DxGather – Merges multiple branched streams into a single unified stream.
- DxInputSelector – Merges frames from multiple input streams into a single synchronized output stream, ordered by PTS.
- DxOutputSelector – Distributes a single input stream to multiple outputs based on metadata such as `stream_id`.
- DxRate – Controls the output frame rate of the stream.
- DxMsgConv – Converts inference results into structured messages.
- DxMsgBroker – Sends messages to external systems or networks.
- DxScale – Scales video frames to a specified resolution.
- DxConvert – Converts video frame color formats.

Inference Workflow

The core inference process in DX-STREAM follows this three-stage structure.
- `DxPreprocess → DxInfer → DxPostprocess`

This three-stage pipeline enables seamless integration of AI model inference into multimedia streams while maintaining flexibility for diverse model architectures.

Custom Library Support

The **DxPreprocess**, **DxPostprocess**, and **DxMsgConv** elements support user-defined custom libraries, allowing developers to integrate application-specific preprocessing, postprocessing, and message conversion logic into the pipeline. This makes it easy to adapt DX-STREAM to diverse AI models and deployment requirements.

Efficient AI Inference

The **DxInfer** element executes AI models using the DEEPX NPU, ensuring high-throughput and low-latency performance optimized for edge AI applications.
Inference requires a model in `.dxnn` format, which is generated from an ONNX file using DX-COM.

ⓒ Copyright 2026 DEEPX All Rights Reserved.

1.4 DX-STREAM Model Workflow

The model workflow in **DX-STREAM** consists of three key stages, from model preparation to real-time inference on DEEPX NPU hardware.

Step 1. Model Compilation
DX-COM (Compiler) converts `.onnx` model files into `.dxnn` format, optimized for execution on DEEPX NPU hardware.

Step 2. Model Execution
The compiled `.dxnn` file is loaded onto the NPU.
The host system uses **DX-RT** (Runtime) to:

- Allocate computation tasks to the NPU
- Provide input tensors to the inference engine
- Receive output tensors (inference results) from the NPU

Step 3. Inference Process
All deep learning computations are offloaded to the NPU, ensuring high-performance and low-latency inference execution optimized for edge environments.

ⓒ Copyright 2026 DEEPX All Rights Reserved.

2. DX-STREAM Installation

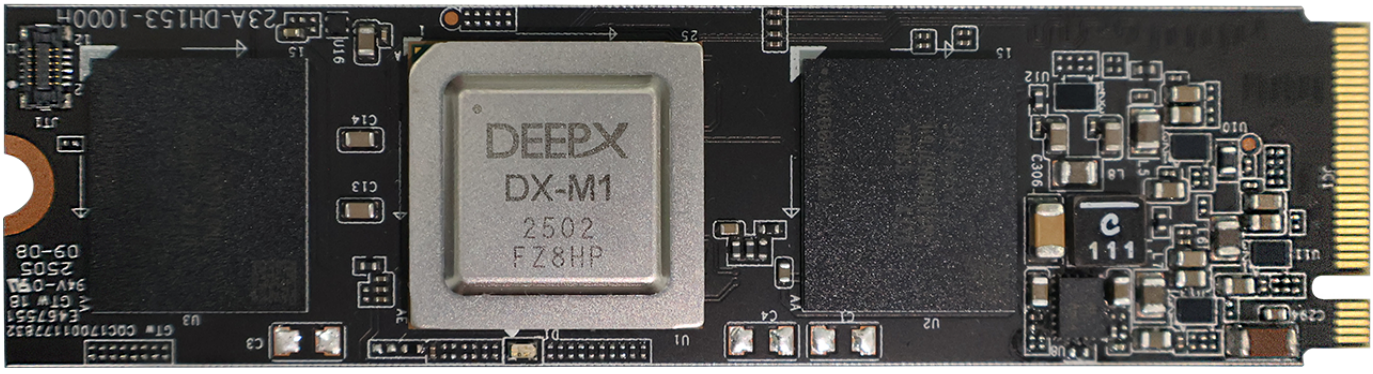
This chapter describes the installation of **DX-STREAM** through a standalone source build or the DX-AllSuite integrated build.

2.1 System Requirements

This section describes the hardware and software requirements for running **DX-STREAM**.

2.1.1 Hardware and Software Requirements

- **CPU:** amd64(x86_64), aarch64(arm64)
- **RAM:** 8GB RAM (16GB RAM or higher is recommended)
- **Storage:** 4GB or higher available disk space
- **OS:** Ubuntu 20.04 / 22.04 / 24.04 / 26.04 (x64, arm64)
  Debian 12 / Debian 13 (x64, arm64)
- **DX-RT version 3.5.0 or higher must be installed**
- The system **must** support connection to an **M1 M.2** module with the M.2 interface on the host PC.



NOTE

The **NPU Device Driver** and **DX-RT** must be installed. For detailed instructions on installing NPU Device Driver and DX-RT, refer to **DX-RT User Manual**.

© Copyright 2026 DEEPX All Rights Reserved.

> **NOTE**
>
> For building and running DX-STREAM on Windows, see Build on Windows.

2.2 Install DX-STREAM

DX-STREAM can be installed in two ways:

- **Standalone source build**: Build DX-STREAM independently from source code
- **DX-AllSuite integrated build**: Build DX-STREAM as part of the DX-AllSuite along with other DEEPX SDKs

DX-AllSuite provides comprehensive environment management with version compatibility for all required dependencies (DX-RT, DX-Driver, DX-FW, etc.) and supports both Docker-based and local installation methods. For detailed information about DX-AllSuite installation and integrated build options, please refer to the **DX-AllSuite User Manual**.

2.2.1 Standalone Source Build

This section focuses on the **standalone source build** approach for DX-STREAM only.

Step 1. Clone the DEEPX-AI GitHub repository

```
$ git clone https://github.com/DEEPX-AI/dx_stream.git
$ cd dx_stream
```

Step 2. Install dependencies

Run the provided script to automatically install all required packages.

```
$ ./install.sh
```

Optional Installation Options

You can selectively skip certain dependencies if they are already installed or not needed:

```
# Skip GStreamer installation
$ ./install.sh --wo-gst

# Skip OpenCV installation
```


© Copyright 2026 DEEPX All Rights Reserved.

```
$ ./install.sh --wo-opencv

# Skip both GStreamer and OpenCV
$ ./install.sh --wo-gst --wo-opencv

# Show help message
$ ./install.sh --help
```

NOTE

Use these options only if you have compatible versions already installed:

- GStreamer 1.16.3 or higher
- OpenCV 4.2.0 or higher

Step 3. Build DX-STREAM and Set Up the Environment

(1) Build & Installation

Compile **DX-STREAM** with the default installation prefix (`/usr/local`) or customize the build using available options.

```
$ ./build.sh
```

Available Build Options

The build script supports various options for customization and maintenance:

```
# Install to custom location
$ ./build.sh --prefix=/opt/dx-stream

# Install to relative path (automatically converted to absolute)
$ ./build.sh --prefix=./install

# Build with debug symbols
$ ./build.sh --type=debug

# Clean previous build files before building
$ ./build.sh --clean

# Enable the optional DEEPX V3 backend
$ ./build.sh --dxv3

# Enable the optional DEEPX VNPU backend
$ ./build.sh --dxvnpu
```


© Copyright 2026 DEEPX All Rights Reserved.

```
# Force-enable the RK RGA backend (built automatically when librga is installed)
$ ./build.sh --rkrga

# Combine flags
$ ./build.sh --dxv3 --type=debug

# Uninstall DX-STREAM
$ ./build.sh --uninstall

# Uninstall from custom location
$ ./build.sh --uninstall --prefix=/opt/dx-stream

# Show all available options
$ ./build.sh --help
```

Installation Structure

By default, DX-STREAM is installed to `/usr/local` with the following structure:

- Plugin: `/usr/local/lib/<arch>-linux-gnu/gstreamer-1.0/libgstdxstream.so`
- Headers: `/usr/local/include/gstdxstream/`
- Custom libraries: `/usr/local/share/gstdxstream/lib/`
- Applications: `/usr/local/share/gstdxstream/bin/`
- pkg-config: `/usr/local/lib/pkgconfig/gstdxstream.pc`

You can change the installation location using `--prefix` option. When using a custom prefix, all paths will be adjusted accordingly.

> **NOTE**
>
> `<arch>-linux-gnu` is `x86_64-linux-gnu` on amd64 and `aarch64-linux-gnu` on arm64. Replace it with the directory name for your system in the commands below.

(2) How to Apply Environment

The build script **automatically adds** required environment variables to your `~/.bashrc` during installation. After installation completes, apply the changes with Step A or Step B, then verify with Step C:

Step A. **Open a new terminal**: The changes will be applied automatically in any new session.

Step B. **Reload your current shell**: To use the changes in the current terminal, run.

© Copyright 2026 DEEPX All Rights Reserved.

```
$ source ~/.bashrc
```

Step C. **Check recognition**: Check if the plugin is correctly recognized by GStreamer.

```
$ gst-inspect-1.0 dxstream
```

Standard Installation (`/usr/local` or `/usr`)

Environment variables are added to `~/.bashrc`, but the system can often find the plugin automatically:

- pkg-config searches `/usr/local/lib/pkgconfig` by default
- GStreamer searches standard plugin directories automatically

If the plugin is not found, clear GStreamer cache:

```
$ rm -rf ~/.cache/gstreamer-1.0/
$ gst-inspect-1.0 dxstream
```


ⓒ Copyright 2026 DEEPX All Rights Reserved.

Custom Prefix Installation

Using custom prefix (e.g., `./install`, `/opt/dx-stream`):

The build script automatically configures all required environment variables in your `~/.bashrc`:

- `PKG_CONFIG_PATH` - For building projects that depend on gstdxstream
- `GST_PLUGIN_PATH` - For GStreamer plugin discovery
- `LD_LIBRARY_PATH` - For runtime library loading
- `PATH` - For DX-STREAM executables

For immediate use in the current terminal (before opening a new one):

```
$ source ~/.bashrc
```

Or set them manually for the current session:

```
$ export PKG_CONFIG_PATH="/path/to/install/lib/pkgconfig:${PKG_CONFIG_PATH}"
$ export GST_PLUGIN_PATH="/path/to/install/lib/<arch>-linux-gnu/gstreamer-1.0:${GST_PLUGIN_PATH}"
$ export LD_LIBRARY_PATH="/path/to/install/lib/<arch>-linux-gnu/gstreamer-1.0:/path/to/install/share/gstdxstream/lib:${LD_LIBRARY_PATH}"
$ export PATH="/path/to/install/share/gstdxstream/bin:${PATH}"
```

(3) Environment Variable Reference

Each variable serves a different purpose during build-time and runtime:

Variable	Purpose	When Needed
`PKG_CONFIG_PATH`	Lets pkg-config find `.pc` files	**Build-time** (when compiling against gstdxstream)
`GST_PLUGIN_PATH`	Tells GStreamer where plugins are	**Runtime** (when running pipelines)
`LD_LIBRARY_PATH`	Dynamic linker finds `.so` files	**Runtime** (when loading shared libraries)
`PATH`	Executable lookup	**Runtime** (when running DX-STREAM apps)

© Copyright 2026 DEEPX All Rights Reserved.

Best Practice

- For **production deployments**: Use standard prefix (`/usr/local`)
- For **development**: Use custom prefix (e.g., `./install`) to avoid requiring sudo
- For **CI/CD**: Set environment variables explicitly in your scripts

Step 4. Final Verification

Check that the plugin is correctly installed:

```
$ gst-inspect-1.0 dxstream
```

You should see the plugin details including available elements (dxpreprocess, dxinfer, dxpostprocess, dxscale, dxconvert, etc.).

© Copyright 2026 DEEPX All Rights Reserved.

Troubleshooting

If `gst-inspect-1.0 dxstream` fails with "No such element or plugin":

- (1) **Clear GStreamer cache** (most common fix):

```
$ rm -rf ~/.cache/gstreamer-1.0/
$ gst-inspect-1.0 dxstream
```

- (2) **Verify plugin file exists:**

```
# For standard installation
$ ls -la /usr/local/lib/<arch>-linux-gnu/gstreamer-1.0/libgstdxstream.so

# For custom installation
$ ls -la /path/to/install/lib/<arch>-linux-gnu/gstreamer-1.0/libgstdxstream.so
```

- (3) **For custom prefix installations**, check environment variables:

```
$ echo $GST_PLUGIN_PATH
# Should include: /path/to/install/lib/.../gstreamer-1.0
```

- (4) **Set GST_PLUGIN_PATH manually** (temporary):

```
$ export GST_PLUGIN_PATH="/usr/local/lib/<arch>-linux-gnu/gstreamer-1.0:${GST_PLUGIN_
PATH}"
$ gst-inspect-1.0 dxstream
```

Uninstalling DX-STREAM

To remove DX-STREAM from your system:

```
# Uninstall from default location (/usr/local)
$ ./uninstall.sh

# Or remove only the installed files with build.sh (does not clean ~/.bashrc)
$ ./build.sh --uninstall

# Uninstall from custom location
$ ./build.sh --uninstall --prefix=/opt/dx-stream
```


© Copyright 2026 DEEPX All Rights Reserved.

`./build.sh --uninstall` automatically removes:

- GStreamer plugin (`libgstdxstream.so`)
- Header files (`include/gstdxstream/`)
- Custom libraries and applications (`share/gstdxstream/`)
- pkg-config file (`gstdxstream.pc`)
- Build directory and pydxs build cache

`./uninstall.sh` runs `./build.sh --uninstall` and additionally:

- **Removes the environment variables from `~/.bashrc`** (automatic cleanup)
- Uninstalls the `pydxs` Python package from the current Python environment
- Deletes the sample models and videos downloaded by `./setup.sh` (run `./setup.sh` again to download them)

Automatic Cleanup

Only `./uninstall.sh` automatically removes DX-STREAM environment variables from your `~/.bashrc`. After uninstalling:

- (1) Open a new terminal, or
- (2) Reload your shell:

```
$ source ~/.bashrc
```

No manual editing of configuration files is required!

2.3 Run DX-STREAM

This section provides a step-by-step guide to quickly running **DX-STREAM**'s sample pipelines

2.3.1 Requirements

© Copyright 2026 DEEPX All Rights Reserved.

Before you start, ensure the following prerequisites are met.

- Properly install **DX-RT**, **the NPU Device Driver**, and **DX-STREAM** in the correct order.
- Download the sample video and model needed to run the demo.

```
$ cd dx_stream
$ ./setup.sh
```

By running the above command, you can download the resources needed for the demo.

2.3.2 Run Demo Pipelines

Execute the demo script.

```
$ cd dx_stream
$ ./run_demo.sh
```

When the script is executed, you'll be prompted to select a demo from the following options.

```
0: Object Detection (YOLO26n)
1: Object Detection (YOLOv5s with PPU)
2: Face Detection (YOLOv5s_Face)
3: Pose Estimation (YOLO26n_Pose)
4: Instance Segmentation (YOLO26n_Seg)
5: Depth Estimation (YOLO26n_Depth)
6: Multi-Object Tracking
7: Multi-Channel Object Detection
8: Multi-Channel Object Detection (RTSP)
9: Secondary Mode (Multi-Model Cascade)
which AI demo do you want to run:(timeout:10s, default:0)
```

Enter the number corresponding to the desired demo to run it.

If **no** input is provided within 10 seconds, the default option (`0: Object Detection (YOLO26n)`) will be executed automatically.

ⓒ Copyright 2026 DEEPX All Rights Reserved.

NOTE

Options with "PPU" suffix utilize **Post-Processing Unit (PPU)** acceleration, which performs model post-processing operations (such as bounding box decoding and score thresholding) directly on the NPU hardware instead of the CPU. This approach significantly improves inference performance by reducing CPU overhead.

Each demo corresponds to a specific pipeline described in **Chapter. Pipeline Example**.

On Raspberry Pi, if `gstreamer1.0-libcamera` and a CSI camera are available, additional camera demos are listed: `c` (Object Detection), `p` (Pose Estimation), `s` (Instance Segmentation), and `d` (Depth Estimation).

Third-Party Model License Notice

Sample models included in DX-STREAM are provided for **evaluation and development purposes only** and are **not licensed for commercial deployment**. Most models are subject to AGPL-3.0, GPL-3.0, or non-commercial research licenses. For commercial use, users must obtain appropriate licenses from the original model providers or use their own commercially licensed models. See Third-Party Sample Model Licenses for details.

© Copyright 2026 DEEPX All Rights Reserved.

3. Elements

3.1 Multi-Stream Domain (application/x-dxvideoraw)

DX-STREAM uses a dedicated caps domain — **`application/x-dxvideoraw`** — to merge multiple `video/x-raw` streams into a single processing chain while preserving each stream's identity, lifecycle, and per-frame metadata. This chapter is a single reference for how that domain works; element-specific pages link back here.

When the multi-stream domain is not used (single-stream pipelines), every DX-STREAM element behaves like a normal GStreamer element negotiating `video/x-raw`. This chapter is only relevant when you place elements between **DxInputSelector** and **DxOutputSelector**.

3.1.1 Domain Boundary

```
[upstream] N × video/x-raw
      ↓ (N sink pads)
    dxinputselector                  ← domain entry
      ↓ 1 src pad, caps = application/x-dxvideoraw
    middle elements …                  ← all negotiate dxvideoraw
      ↓
    dxoutputselector                 ← domain exit
      ↓ N src pads, caps = video/x-raw
[downstream] N × video/x-raw
```

- Inside the domain, all elements negotiate `application/x-dxvideoraw` on both sink and src.
- Standard GStreamer elements (e.g., `videoconvert`, `videoscale`, `compositor`) cannot accept `application/x-dxvideoraw` and will fail caps negotiation. The caps system itself prevents incorrect placement.

3.1.2 Caps Definition

```
application/x-dxvideoraw,
  width=(int)[1, MAX],
  height=(int)[1, MAX],
  framerate=(fraction)[0/1, MAX]
```


© Copyright 2026 DEEPX All Rights Reserved.

Caps fields are intentionally *ranges* and `format` is omitted to accept any pixel format. Actual per-buffer values (`stream_id`, `width`, `height`, `format`) are carried inside **`DXFrameMeta`** attached to each `GstBuffer`. This is what lets a single chain process multiple streams that may differ in resolution and color format.

3.1.3 Element Placement Matrix

Element	Inside domain	Outside domain	Notes
`dxpreprocess`, `dxinfer`, `dxpostprocess`, `dxosd`, `dxrate`, `dxtracker`	OK	OK	Dual mode — behavior is chosen from caps
`dxmsgconv`, `dxmsgbroker`	OK	OK	Dual, caps-agnostic (operate on metadata)
`dxscale`, `dxconvert`	NOT allowed	OK	Need stable single caps; multi-stream multi-resolution unsupported
`dxgather`	NOT allowed	OK	Merges branches that share a single source — does not support multiple stream IDs
`dxinputselector`, `dxoutputselector`	boundary only	NOT used	Domain entry / exit

`dxscale` and `dxconvert` must be placed **per stream upstream of `dxinputselector`** or **per output downstream of `dxoutputselector`**:

```
src1 ! decode1 ! dxscale ! dxconvert ! \
src2 ! decode2 ! dxscale ! dxconvert !  dxinputselector ! ... ! dxoutputselector ! ...
src3 ! decode3 ! dxscale ! dxconvert ! /
```

3.1.4 Per-Buffer Identity

ⓒ Copyright 2026 DEEPX All Rights Reserved.

Every buffer inside the domain carries a **DXFrameMeta**. Dual-mode elements use DXFrameMeta (not caps) to read the buffer's actual dimensions, format, and stream identity:

Field	Purpose
_stream_id	Logical stream index assigned by dxinputselector from the sink pad number
_width, _height, _format	Per-buffer dimensions/format (may differ between buffers)

If a buffer arriving at dxinputselector has no DXFrameMeta, the selector creates one and sets _stream_id from the sink pad number.

The maximum number of streams is 64 (_stream_id must be in the range 0–63). Buffers with an out-of-range _stream_id are dropped with a warning.

3.1.5 Event Model

Inside the domain, two layers of events coexist:

- **L1A — domain-wide standard events.** A single STREAM_START, CAPS (application/x-dxvideoraw), SEGMENT, and (eventually) EOS for the entire merged chain. dxinputselector emits these once; intermediate elements forward them through standard GStreamer paths.
- **L1B — pipeline-wide standard events.** FLUSH_START, FLUSH_STOP (and non-wrapped TAG / GAP). These are not stream-specific and are propagated normally. RECONFIGURE travels upstream and is handled as an L2 event (see below).
- **L2 — per-stream lifecycle events.** Each input stream's original STREAM_START, CAPS (video/x-raw), SEGMENT, TAG, EOS, GAP is **wrapped** as a CUSTOM_DOWNSTREAM event of type application/x-dx-wrapped-event that carries the stream-id. Upstream-direction QoS and RECONFIGURE are wrapped similarly as CUSTOM_UPSTREAM events of type application/x-dx-wrapped-event-upstream.

© Copyright 2026 DEEPX All Rights Reserved.

Downstream behavior:

- `dxinputselector` emits L1A sticky events once, and wraps every per-stream sticky event as an L2 event.
- Intermediate (dual-mode) elements either ignore L2 events (they are not their own caps) or peek at them to update per-stream state (e.g., `dxosd` learns each stream's `video_info` from wrapped CAPS).
- `dxoutputselector` drops L1A events (the global merged ones), broadcasts L1B (`FLUSH_START`, `FLUSH_STOP`, and non-wrapped `TAG` / `GAP`) to every output, ignores a raw `RECONFIGURE` arriving on its sink pad, and unwraps each L2 event to the matching srcpad for that stream-id.

Upstream behavior:

- A `QoS` event arriving on one of `dxoutputselector`'s src pads is wrapped with the stream-id and sent upstream. `dxinputselector` unwraps it and routes it to the corresponding sink (so a slow stream throttles its own source instead of throttling every input).

You do not need to construct or inspect L2 events yourself — the boundary elements handle all wrap/unwrap.

3.1.6 Query Policy

`dxinputselector` and `dxoutputselector` answer standard GStreamer queries so that LATENCY, ALLOCATION, POSITION, and DURATION work transparently across the domain. In particular:

- **LATENCY**: each intermediate element adds its own time to the upstream latency; `dxoutputselector` forwards LATENCY queries from each src pad to the sink peer.
- **ALLOCATION**: dual-mode elements in domain mode request `DXFrameMeta` as a buffer meta during allocation.
- **CAPS / ACCEPT_CAPS**: standard responses against the pad template (`application/x-dxvideoraw` inside the domain).

LATENCY inside the domain is reported as a single, stream-agnostic value — the same NPU, kernel and queues are shared by all streams.

ⓒ Copyright 2026 DEEPX All Rights Reserved.

3.1.7 Troubleshooting

- **"Could not link element ..."** between `dxinputselector` and a downstream element usually means the downstream element does not accept `application/x-dxvideoraw`. See `Troubleshooting and FAQ → Domain caps mismatch`.
- **Missing `DXFrameMeta`** inside the domain means a buffer entered the chain without going through `dxinputselector`. See `Troubleshooting and FAQ → Missing DXFrameMeta inside the multi-stream domain`.

ⓒ Copyright 2026 DEEPX All Rights Reserved.

3.2 DxPreprocess

DxPreprocess is an element that performs input preprocessing on raw video frames, converting them into a format suitable for AI models used by **DxInfer**. Each preprocessed input tensor is assigned an input ID specified by the `preprocess-id` property.

DxInfer must reference this ID in its own configuration to determine which tensor to use for inference.

Multiple **DxPreprocess** elements can co-exist in the same pipeline, enabling support for multi-input models or customized preprocessing steps for different video streams.

Key Features

Color Conversion

DxPreprocess converts the image color format to `RGB` or `BGR` based on the `color-format` property.

- Default format: `RGB`.

Modes of Operation

- **Primary Mode** applies preprocessing to the entire frame. If object detection is performed within the same pipeline, **DxPreprocess** will operate in this mode.
- **Secondary Mode** applies preprocessing to individual object regions detected in this frame. This mode requires object metadata (e.g., from an upstream object detection element).

Region of Interest (ROI)

The Region of Interest (ROI) is defined using the roi property.

- In **Primary Mode**, the specified ROI area is cropped and preprocessed as a whole.
- In **Secondary Mode**, only objects that are fully contained within the ROI are selected for preprocessing.

Processing Interval

The processing interval is controlled by the `interval` property. It skips a specified number of frames before preprocessing the next frame or object. This is useful for reducing processing frequency in resource-constrained environments.

ⓒ Copyright 2026 DEEPX All Rights Reserved.

Object Filtering in Secondary Mode

Objects can be filtered based on the following criteria.

- **Class ID**: Use the `target-class-id` property to process only objects that match the specified class. Non-matching objects are ignored.
- **Size** : Use the `min-object-width` and `min-object-height` properties to exclude objects smaller than the defined size.

Resizing

DxPreprocess resizes full frames or object regions using the `resize-width` and `resize-height` property.
If `keep-ratio` is set to `true`, the aspect ratio is preserved by applying padding.
Padding color is set using the `pad-value` property.

Custom Preprocessing

User-defined preprocessing logic can be implemented by providing:

- `library-file-path` : Path to the custom shared library (`.so`).
- `function-name` : Name of the preprocessing function within the library.

The custom function signature has been updated to include GstBuffer access:

```
extern "C" bool PreProcess(GstBuffer *buf,
                    DXFrameMeta *frame_meta,
                    DXObjectMeta *object_meta,
                    void *output);
```

This allows implementation of customized data handling tailored to specific AI models with direct access to the GStreamer buffer.

QoS Handling

If the downstream sink element has `sync=true`, input buffers may be dropped based on their timestamps. This helps maintain real-time performance and avoids frame backlog under system load.

ⓒ Copyright 2026 DEEPX All Rights Reserved.

H/W Acceleration

DxPreprocess uses the `VideoTransformFactory` to automatically select the best available hardware backend for color conversion and resizing. The selection priority is:

1. **V3 DSP** – DEEPX V3 device DSP (when the V3 sibling plugin is installed)
2. **VNPU** – DEEPX VNPU hardware backend (when the VNPU sibling plugin is installed)
3. **RGA** – Rockchip Raster Graphic Accelerator (when the RGA sibling plugin is installed)
4. **libyuv** – Software fallback (always available)

The backend is selected automatically at runtime — no configuration required.

Hierarchy

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

Pad Templates

Sink (input)

Property	Value
Format	`video/x-raw, format=(string){ RGB, I420, NV12 }; application/x-dxvideoraw`

Src (output)

Property	Value
Format	`video/x-raw, format=(string){ RGB, I420, NV12 }; application/x-dxvideoraw`

ⓒ Copyright 2026 DEEPX All Rights Reserved.

Caps `format` vs `color-format` property

The sink/src caps `format` specifies the **input video color space** (the raw frame layout that `DxPreprocess` accepts). The `color-format` property below specifies the **output tensor color space** (RGB or BGR) produced for `DxInfer`. They are independent — for example, an NV12 input video can produce a BGR output tensor.

Domain Mode Behavior

`dxpreprocess` is a dual-mode element (see Multi-Stream Domain).

- **Normal mode** (sink caps = `video/x-raw`): frame dimensions are read from caps. `stream_id` is treated as `0`.
- **Domain mode** (sink caps = `application/x-dxvideoraw`): frame dimensions are read from `DXFrameMeta._width/_height`, and `stream_id` from `DXFrameMeta._stream_id`. The element also requests `DXFrameMeta` as a buffer meta during `ALLOCATION` so that upstream pools allocate buffers with the meta attached.

Properties

This table provides a complete reference to the properties of the **DxPreprocess** element.

Name	Description	Type	Default Value
`name`	Sets the unique name of the DxPreprocess element.	String	`"dxpreprocess0"`
`config-file-path`	Path to the JSON config file containing the element's properties.	String	`null`
`preprocess-id`	Assigns an ID to the preprocessed input tensor.	Unsigned Integer	`0`
`resize-width`	Specifies the width for resizing.	Unsigned Integer	`0`
`resize-height`	Specifies the height for resizing.	Unsigned Integer	`0`

© Copyright 2026 DEEPX All Rights Reserved.

Name	Description	Type	Default Value
`keep-ratio`	Maintains the original aspect ratio during resizing.	Boolean	`true`
`pad-value`	Padding color value for R, G, B (range: 0-255)	Unsigned Integer	`0`
`color-format`	Specifies the color format for preprocessing. (`0` : RGB, `1` : BGR)	Unsigned Integer	`0` (RGB)
`secondary-mode`	Enables Secondary Mode for processing object regions.	Boolean	`false`
`target-class-id`	Filters objects in Secondary Mode by class ID. (`-1` processes all objects).	Integer	`-1`
`min-object-width`	Minimum object width for preprocessing in Secondary Mode.	Unsigned Integer	`0`
`min-object-height`	Minimum object height for preprocessing in Secondary Mode.	Unsigned Integer	`0`
`roi`	Defines the ROI (Region of Interest) for preprocessing as a comma-separated string `"x1,y1,x2,y2"` .	String	`"-1,-1,-1,-1"`
`interval`	Specifies the interval for preprocessing frames or objects.	Unsigned Integer	`0`
`transpose`	Enables transposing of the output tensor axes.	Boolean	`false`

ⓒ Copyright 2026 DEEPX All Rights Reserved.

Name	Description	Type	Default Value
library-file-path	Path to the custom preprocess library, if used.	String	null
function-name	Name of the custom preprocessing function to use.	String	null

Example JSON Configuration

```
{
  "preprocess_id": 1,
  "resize_width": 640,
  "resize_height": 640,
  "keep_ratio": true
}
```

NOTE

- For implementing custom preprocess logic, refer to **Chapter. Writing Your Own Application** **"Writing Custom Pre-Process Function"**. Custom libraries must follow the error reporting contract described there (no `g_error()` / `abort()`; use `g_warning()` for per-frame skips and `throw std::runtime_error` for permanent errors).
- All properties can also be configured using a JSON file for enhanced usability and flexibility.

© Copyright 2026 DEEPX All Rights Reserved.

3.3 DxInfer

DxInfer is an element that performs AI model inference using the **DEEPX** NPU. It processes input tensors received from **DxPreprocess** elements and produces output tensors for downstream processing.

- **Input tensors: DxInfer** receives preprocessed input tensors from **DxPreprocess** and performs inference using a specified AI model.
- **Output tensors:** Each output tensor is assigned an ID using the `inference-id` property, allowing downstream elements such as **DxPostprocess** to retrieve the correct output.
- **Model configuration:** The AI model used for inference must be specified using the `model-path` property, which points to a compiled `.dxnn` file.
- **Backend: DxInfer** performs inference using the `dxrt` or `dxvnpu` backend, selectable via the `backend` property.

Key Features

Input Tensor Management

Input tensors are linked to **DxInfer** using the `preprocess-id` property. This ensures that the correct tensor from **DXPreprocess** is used for inference.

Output Tensor Management

Each output tensor is uniquely defined in the `inference-id` property. This allows downstream elements like **DXPostprocess** to connect to the correct inference output.

Pipeline Configuration

The recommended pipeline chain is **[DxPreprocess] → [DxInfer] → [DxPostprocess]**. When using features like `secondary-mode`, the configuration must be consistently applied across all three elements (**DxPreprocess , DxInfer and DxPostprocess**).

QoS Handling

If the downstream sink element has `sync=true`, input buffers may be dropped based on their timestamps to maintain real-time processing performance.

Throttle QoS Events

When **DxRate** sends a Throttle QoS Event, **DxInfer** drops incoming frames until the accumulated time between frames exceeds the internal `throttling_delay` value. This value is derived from upstream Throttle QoS Events and cannot be configured as a **DxInfer** property. This avoids unnecessary NPU computation in low-framerate pipelines and promotes smooth and consistent streaming.

ⓒ Copyright 2026 DEEPX All Rights Reserved.

Backend Selection

DxInfer selects the inference backend via the `backend` property:

- **auto** (default): Automatically selects the available backend (`dxrt` or `dxvnpu`).
- **dxrt**: Uses the DEEPX Runtime (DX-RT) backend.
- **dxvnpu**: Uses the DEEPX VNPU backend (requires the VNPU sibling plugin, built with `--dxvnpu`).

JSON Configuration

All properties can be configured through a JSON file using the `config-file-path` property. This enables reusable, clean, and scalable configuration of inference behavior.

Hierarchy

```
GObject
 +----GInitiallyUnowned
      +----GstObject
           +----GstElement
                +----GstDxInfer
```

Pad Templates

Sink (input)

Property	Value
Format	`video/x-raw; application/x-dxvideoraw`

Src (output)

Property	Value
Format	`video/x-raw; application/x-dxvideoraw`

Domain Mode Behavior

ⓒ Copyright 2026 DEEPX All Rights Reserved.

`dxinfer` is a dual-mode element (see Multi-Stream Domain).

- **Normal mode** (sink caps = `video/x-raw`): operates on a single stream; the input tensor lookup uses `preprocess-id` as configured.
- **Domain mode** (sink caps = `application/x-dxvideoraw`): per-stream identity is read from `DXFrameMeta._stream_id`. Inference itself is a shared resource — a single NPU/runtime is used across all streams, and LATENCY is reported as a single domain-wide value (no per-stream LATENCY split).

Latency Reporting

`DxInfer` adds its measured inference time (the moving average `avg_latency` of recent inference calls) as the element's own latency contribution when responding to `LATENCY` queries. Downstream sinks therefore see a realistic pipeline latency that includes inference cost.

Properties

Name	Description	Type	Default Value
`name`	Sets the unique name of the DxInfer element.	String	`"dxinfer0"`
`config-file-path`	Path to the JSON config file containing the element's properties.	String	`null`
`model-path`	Path to the `.dxnn` model file used for inference.	String	`null`
`preprocess-id`	Key of the input tensor in `DXFrameMeta` / `DXObjectMeta` to feed into inference (must match the `preprocess-id` used by the upstream **DxPreprocess**).	Unsigned Integer	`0`

ⓒ Copyright 2026 DEEPX All Rights Reserved.

Name	Description	Type	Default Value
`inference-id`	Key under which inference output tensors are stored in `DXFrameMeta` / `DXObjectMeta` (the downstream **DxPostprocess** retrieves them by this ID).	Unsigned Integer	`0`
`secondary-mode`	Determines whether to operate in primary mode or secondary mode.	Boolean	`false`
`use-ort`	Determines whether to use ONNX Runtime (ORT) for inference.	Boolean	`true`
`backend`	Selects the inference backend: `auto`, `dxrt`, or `dxvnpu`.	Enum	`auto`

Example JSON Configuration

```
{
   "preprocess_id": 1,
   "inference_id": 1,
   "model_path" : "./dx_stream/samples/models/yolov5-s_640x640_ppu.dxnn",
   "backend": "auto"
}
```

NOTE

- The pipeline must follow **[DxPreprocess]** → **[DxInfer]** → **[DxPostprocess]** for correct and stable operation.
- All properties can also be configured using a JSON file for enhanced usability and flexibility.

© Copyright 2026 DEEPX All Rights Reserved.

3.4 DxPostprocess

DxPostprocess is a pipeline element that handles output tensors generated by the **DxInfer** element. It performs model-specific post-processing, such as classification, detection, or segmentation, and produces structured results by creating or modifying `DXObjectMeta`.

The component supports user-defined post-processing logic provided as a custom library, specified by the `library-file-path` and `function-name` properties. Processed results are attached to `DXFrameMeta`, enabling downstream tasks such as visualization, event handling, or application-specific decisions.

Key Features

Modes of Operation

- **Primary Mode:** Creates new `DXObjectMeta` objects and attaches them to the corresponding `DXFrameMeta`
- **Secondary Mode:** Modifies existing `DXObjectMeta` based on updated inference results from **DxInfer**

Custom Postprocessing

You can use a custom post-processing function by specifying:

- `library-file-path`: Path to the shared object `(.so)` containing your custom function
- `function-name`: Name of the function to be called

The custom function signature has been updated to include GstBuffer access:

```
extern "C" void PostProcess(GstBuffer *buf,
                    std::vector<dxs::DXTensor> network_output,
                    DXFrameMeta *frame_meta,
                    DXObjectMeta *object_meta);
```

This allows full flexibility to implement custom decoding, filtering, or data transformation logic with direct access to the GStreamer buffer.

Hierarchy

```
GObject
 +----GInitiallyUnowned
```


© Copyright 2026 DEEPX All Rights Reserved.

```
        +----GstObject
            +----GstElement
                +----GstBaseTransform
                    +----GstDxPostprocess
```

Pad Templates

Sink (input)

Property	Value
Format	`video/x-raw; application/x-dxvideoraw`

Src (output)

Property	Value
Format	`video/x-raw; application/x-dxvideoraw`

Domain Mode Behavior

`dxpostprocess` is a dual-mode, stateless element (see Multi-Stream Domain). Behavior is identical whether the sink caps are `video/x-raw` (single stream) or `application/x-dxvideoraw` (domain) — the element reads per-buffer information from `DXFrameMeta` / `DXObjectMeta` either way.

Properties

Name	Description	Type	Default Value
`name`	Sets the unique name of the DxPostprocess element.	String	`"dxpostprocess0"`
`config-file-path`	Path to the JSON config file containing the element's properties.	String	`null`
`inference-id`	Specifies the ID of the DxInfer output tensor	Unsigned Integer	`0`

© Copyright 2026 DEEPX All Rights Reserved.

Name	Description	Type	Default Value
	to use for postprocessing.		
secondary-mode	Enables Secondary Mode for processing object regions.	Boolean	false
library-file-path	Path to the custom postprocess library.	String	null
function-name	Name of the custom postprocessing function to use.	String	null

Example JSON Configuration

```
{
  "inference_id": 1,
  "library_file_path": "/usr/local/share/gstdxstream/lib/libpostprocess_yolo26od.so",
  "function_name": "PostProcess"
}
```

NOTE

- All properties can also be configured using a JSON file for enhanced usability and flexibility.
- For implementing custom postprocess logic, refer to **Chapter. Writing Your Own Application** **"Writing Custom Post-Process Function"**. Custom libraries must follow the error reporting contract described there (no `g_error()` / `abort()`; use `g_warning()` for per-frame skips and `throw std::runtime_error` for permanent errors).

© Copyright 2026 DEEPX All Rights Reserved.

3.5 DxTracker

DxTracker is a GStreamer element designed to perform Multi-Object Tracking (MOT) on object detection results in a video stream. It uses bounding box metadata from upstream detection elements (such as **DxPostprocess**) to continuously track and assign consistent IDs across frames.

Currently, **DxTracker** supports the `OC_SORT` algorithm as its default tracking method. For more information about `OC_SORT`, refer to the OC_SORT repository.

Key Features

Multi-Object Tracking (MOT)

DxTracker tracks the movement of detected objects over time. It assigns unique track IDs to each object. Objects that can **not** be reliably tracked are discarded and **not** forwarded downstream.

Configurable Tracking Algorithms

Tracking algorithms and their parameters are defined in a JSON configuration file specified via the `config-file-path` property.

Example JSON configuration

```
{
  "tracker_name": "OC_SORT",
  "params": {
    "det_thresh": 0.5,
    "max_age": 30,
    "min_hits": 3,
    "iou_threshold": 0.3,
    "delta_t": 3,
    "asso_func": "iou",
    "inertia": 0.2,
    "use_byte": false
  }
}
```

The configuration file must include

- `tracker_name` : Specifies the tracking algorithm (e.g., `OC_SORT`)
- `params` : Contains algorithm-specific parameters

© Copyright 2026 DEEPX All Rights Reserved.

Default Behavior
If no config file is provided, **DxTracker** uses the `OC_SORT` algorithm with built-in default values.
If only `tracker_name` is specified without params, the `default` parameter values for that algorithm are applied.

Limitations
The number of tracked objects is always less than or equal to the number of detected bounding boxes.
Algorithm parameters **must** be defined within the `params` block of the JSON file. These can **not** be set as individual GStreamer properties on the element.

Hierarchy

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

Pad Templates

Sink (input)

Property	Value
Format	`video/x-raw; application/x-dxvideoraw`

Src (output)

Property	Value
Format	`video/x-raw; application/x-dxvideoraw`

Properties

Name	Description	Type	Default Value
`name`		String	`"dxtracker0"`

ⓒ Copyright 2026 DEEPX All Rights Reserved.

Name	Description	Type	Default Value
	Sets the unique name of the DxTracker element.		
config-file-path	Path to the JSON config file containing the tracking algorithm and parameters.	String	null
tracker-name	Specifies the name of the tracking algorithm to use.	String	"OC_SORT"

Domain Mode Behavior

dxtracker is a dual-mode element (see Multi-Stream Domain).

- **Normal mode** (sink caps = video/x-raw): a single OC-SORT instance tracks objects across the single stream. All track IDs come from one shared ID space.
- **Domain mode** (sink caps = application/x-dxvideoraw): dxtracker keeps a **separate OC-SORT instance per _stream_id**. Track IDs are allocated independently inside each stream, so the same numeric track ID may appear in two different streams referring to two unrelated objects. Tracks never cross stream boundaries.

NOTE

- The JSON configuration file is required to customize tracking algorithm parameters.
- If **no** configuration file is provided, **DxTracker** uses OC_SORT with default settings.
- All parameter values **must** be defined within the params section of the JSON file.
- Track IDs are only assigned to successfully tracked objects. Objects that can **not** be tracked are removed and **not** passed downstream.

© Copyright 2026 DEEPX All Rights Reserved.

3.6 DxOsd

DxOsd is a GStreamer element that provides On-Screen Display (OSD) capabilities by overlaying object information onto the original video frame.
It uses object metadata (Object Meta) passed from upstream elements, such as DxPostprocess or DxTracker, to render visual elements directly on the frame.

Key Features

Draw Inference Results

- Draws bounding boxes on video frames
- Displays class labels and confidence scores
- Colors boxes by track ID or class
- Supports segmentation maps, depth maps, oriented bounding boxes (OBB), human pose, and facial landmarks

Hierarchy

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

Pad Templates

Sink (input)

Property	Value
Format	`video/x-raw, format=(string){ RGB, BGR, NV12, I420 }, width=(int)[1, 16384], height=(int)[1, 16384], framerate=(fraction)[0/1, 16384/1]; application/x-dxvideoraw`

ⓒ Copyright 2026 DEEPX All Rights Reserved.

Src (output)

Property	Value
Format	`video/x-raw, format=(string){ RGB, BGR, NV12, I420 }, width=(int)[1, 16384], height=(int)[1, 16384], framerate=(fraction)[0/1, 16384/1]; application/x-dxvideoraw`

Domain Mode Behavior

`dxosd` is a dual-mode element (see Multi-Stream Domain).

- **Normal mode** (sink caps = `video/x-raw`): a single `video_info` is taken from the caps and reused for every buffer.
- **Domain mode** (sink caps = `application/x-dxvideoraw`): `dxosd` reads the per-stream wrapped `CAPS` events emitted by `dxinputselector` (one per stream, carrying the original `video/x-raw` caps and the stream-id) and stores a separate `video_info` per `_stream_id`. For each incoming buffer it picks the matching `video_info` from `DXFrameMeta._stream_id`, so streams with different resolutions or formats are rendered correctly.

Properties

Name	Description	Type	Default Value
`name`	Sets the unique name of the DxOsd element.	String	`"dxosd0"`

NOTE

- DXOSD supports RGB, BGR, NV12, and I420 input formats. The output format matches the input format (passthrough — no format conversion is applied).
- In DXOSD's output buffers, the `DXFrameMeta` metadata is preserved and passed to downstream elements.
- Visualizations include bounding boxes, class names, confidence scores, and additional data like segmentation maps, depth maps, oriented bounding boxes, poses, or face landmarks, depending on available metadata.

© Copyright 2026 DEEPX All Rights Reserved.

Multi-Stream Support

DxOsd can be safely placed downstream of **DxInputSelector**. It automatically tracks per-stream video information via wrapped caps events, allowing correct OSD rendering for buffers from different streams with different resolutions and formats.

ⓒ Copyright 2026 DEEPX All Rights Reserved.

3.7 DxGather

DxGather is an element designed to merge multiple branches of a stream that originates from the same source, typically split using a GstTee. It recombines these branches back into a single synchronized output stream, ensuring efficient handling of shared content in multi-branch pipelines.

This makes **DxGather** ideal for scenarios where duplicated streams need to be reunited after parallel processing (e.g., detection, tracking, or visualization).

Key Features

PTS-Based Buffer Merging

- Merges
  buffers from multiple sink pads only when their Presentation Timestamps (PTS) match.
- Ensures that only synchronized frames are merged together.
- The latest PTS among the sink pads is used as the reference. Buffers with a different (older) PTS are dropped with a warning.

Buffer and Metadata Handling

- Assumes that buffer content is identical across all input branches.
- Retains one buffer while others are unreferenced.
- Merges object metadata (`DXObjectMeta` , matched by `_meta_id`) from all branches into the resulting output buffer. Empty fields are filled from the other branches, and objects that exist only in other branches are added.
- Frame-level metadata (segmentation map, depth map, user metadata) and object user metadata are not merged; the values of the retained buffer are kept.

Source Stream Assumption

- Only supports streams split from the same source (e.g., using GstTee).
- **Not** suitable for merging streams from different sources. Buffers from a different source (`_stream_id`) are dropped with a warning.

Hierarchy

```
GObject
 +----GInitiallyUnowned
```


ⓒ Copyright 2026 DEEPX All Rights Reserved.

```
        +----GstObject
            +----GstElement
                +----GstAggregator
                    +----GstDxGather
```

Pad Templates

Sink (input, request pad `sink_%u`)

Property	Value
Format	`video/x-raw`

Src (output, always pad `src`)

Property	Value
Format	`video/x-raw`

Properties

Name	Description	Type	Default Value
`name`	Sets the unique name of the DxGather element.	String	`"dxgather0"`

> ⚠ **DxInputSelector Placement**
>
> `dxgather` must **not** be placed downstream of **DxInputSelector** (i.e., it cannot be used inside the multi-stream domain). `dxgather` is designed to merge multiple branches that originate from the *same* source (e.g., split by `tee`), not to demultiplex multiple distinct streams. Inside the domain, buffers carry per-stream `DXFrameMeta` and `dxgather` has no notion of stream-id. See Multi-Stream Domain.

© Copyright 2026 DEEPX All Rights Reserved.

3.8 DxInputSelector

DxInputSelector is a GStreamer element designed for multi-channel video streaming. It merges frames from multiple input streams into a single synchronized output stream, selecting frames based on Presentation Timestamp (PTS) ordering.

It is the entry boundary of the multi-stream domain — its src caps are `application/x-dxvideoraw`. See Multi-Stream Domain for the overall model.

Key Features

Stream Selection

- Among N input streams, DxInputSelector selects the buffer with the smallest PTS and forwards it downstream.
- This approach ensures that the output stream maintains temporal consistency across input channels.
- Buffers without a valid PTS are dropped.

Event Handling

- After all input sink pads have received their first sticky events, `DxInputSelector` emits a single domain `STREAM_START`, a `CAPS` event with `application/x-dxvideoraw`, and a `SEGMENT` event downstream — these are the L1A events that drive the merged chain.
- The original per-stream sticky events (`STREAM_START`, `CAPS=video/x-raw`, `SEGMENT`, `TAG`, `GAP`) and per-stream `EOS` are wrapped as `application/x-dx-wrapped-event` (a `CUSTOM_DOWNSTREAM` event) tagged with the stream-id, and emitted alongside the L1A events. Downstream elements that need per-stream lifecycle information (e.g., `DxOsd`, `DxOutputSelector`) read them from the wrap.
- `EOS` received on a single sink does **not** propagate downstream immediately. It is wrapped as a per-stream `EOS`. Only after every sink has received `EOS` is a single global `EOS` emitted downstream.
- Upstream `QoS` / `RECONFIGURE` events arriving as `CUSTOM_UPSTREAM` wraps from `DxOutputSelector` are unwrapped and routed to the corresponding sink pad's upstream peer.

Hierarchy

```
GObject
 +----GInitiallyUnowned
      +----GstObject
```


© Copyright 2026 DEEPX All Rights Reserved.

```
            +----GstElement
                +----GstAggregator
                    +----GstDxInputSelector
```

Pad Templates

Sink (input, request pad `sink_%u`)

Property	Value
Format	`video/x-raw`

Src (output, always pad `src`)

Property	Value
Format	`application/x-dxvideoraw`

Properties

Name	Description	Type	Default Value
`name`	Sets the unique name of the DxInputSelector element.	String	`"dxinputselector0"`
`max-queue-size`	Number of buffers per stream assumed when reporting LATENCY (reported latency = `max-queue-size` × frame duration). It does not limit the queue size.	Unsigned Integer	`2`

ⓒ Copyright 2026 DEEPX All Rights Reserved.

NOTE

- If an incoming buffer does **not** contain `DXFrameMeta`, the element creates a new `DXFrameMeta` using `dx_create_frame_meta()` and assigns the sink pad index as the `stream_id`.
- This metadata tagging is essential for downstream elements that rely on stream identification, such as `DxOutputSelector`.
- When two input streams share the same framerate, their buffers can interleave by up to **50%** in the merged output (i.e., a buffer from one stream may arrive several PTS positions ahead or behind its strict interleave position before the other stream catches up). This is expected behavior of PTS-based selection.

Downstream Element Restrictions

`GstBaseTransform`-based elements that depend on stable caps negotiation (e.g., **DxScale**, **DxConvert**) must **not** be placed directly downstream of DxInputSelector. The reason is that the src caps of DxInputSelector are `application/x-dxvideoraw`, not `video/x-raw` — standard GStreamer elements (`videoconvert`, `videoscale`, `compositor`, …) will fail caps negotiation. Elements designed for the domain (`DxPreprocess`, `DxInfer`, `DxPostprocess`, `DxTracker`, `DxOsd`, `DxRate`, `DxMsgConv`, `DxMsgBroker`) accept `application/x-dxvideoraw` and read per-buffer dimensions/stream-id from `DXFrameMeta`.
See Multi-Stream Domain for the placement matrix.

© Copyright 2026 DEEPX All Rights Reserved.

3.9 DxOutputSelector

DxOutputSelector is a GStreamer element used in multi-channel streaming pipelines. It receives a unified stream from `DxInputSelector` and redistributes the buffers to N output pads based on metadata, such as `stream_id`.

It is the exit boundary of the multi-stream domain — see Multi-Stream Domain for the overall model.

Key Features

Buffer Routing

- Buffers arriving at the single sink pad are routed to the correct `src` pad using the `stream_id` field found in the associated `DXFrameMeta`.
- This enables seamless demultiplexing of multiple logical video streams within a unified pipeline.
- If `stream_id` is out of range for the configured number of src pads, the buffer is dropped with a warning and the pipeline continues.

Event Handling

Sink-side events arriving from upstream are classified and dispatched in three ways:

- **Domain-wide standard events** (`STREAM_START`, `CAPS=application/x-dxvideoraw`, `SEGMENT`, the final global `EOS`) are consumed at the boundary — the downstream pipeline outside the domain does not receive them. Each output stream's lifecycle is reconstructed from the per-stream wrapped events below.
- **Pipeline-wide events** (`FLUSH_START`, `FLUSH_STOP`) and non-wrapped `TAG` / `GAP` events (as well as any unknown event) are **broadcast to every src pad** so that every downstream branch flushes in sync. A raw `RECONFIGURE` arriving on the sink pad is ignored.
- **Per-stream wrapped events** (`application/x-dx-wrapped-event` from `dxinputselector`, carrying the original per-stream `STREAM_START`, `CAPS=video/x-raw`, `SEGMENT`, `TAG`, `EOS`, `GAP`) are unwrapped and forwarded to the src pad whose index matches the wrapped stream-id. This is how each downstream output sees its own original `video/x-raw` lifecycle.

Upstream QoS

`QoS` and `RECONFIGURE` events arriving on a src pad are wrapped with the corresponding stream-id and sent upstream as a `CUSTOM_UPSTREAM` event. `dxinputselector` unwraps them and forwards each to the matching sink, so a slow consumer on stream `N` only throttles its

© Copyright 2026 DEEPX All Rights Reserved.

own upstream source — not the other streams. Other upstream events (e.g., `SEEK`, `NAVIGATION`) are forwarded to the sink pad as-is.

Hierarchy

```
GObject
 +----GInitiallyUnowned
       +----GstObject
             +----GstElement
                   +----GstDxOutputSelector
```

Pad Templates

Sink (input, always pad `sink`)

Property	Value
Format	`application/x-dxvideoraw`

Src (output, request pad `src_%u`)

Property	Value
Format	`video/x-raw`

Properties

Name	Description	Type	Default Value
`name`	Sets the unique name of the DxOutputSelector element.	String	`"dxoutputselector0"`

ⓒ Copyright 2026 DEEPX All Rights Reserved.

NOTE

- `DxOutputSelector` requires asynchronous operation between its upstream and downstream elements. To achieve this, a `queue` element must be added to each of its src pads. Failure to do so may result in abnormal pipeline hangs.
- The `stream_id` from the `DXFrameMeta` of the buffer received on the sink pad is parsed and used as the index of the src pad. If `stream_id` does not match any configured src pad index, the buffer is dropped with a warning and the pipeline continues. A buffer without `DXFrameMeta` is dropped silently (no warning).

No global EOS downstream

By design, `DxOutputSelector` does **not** forward the domain-wide global `EOS` to its src pads. Applications that wait for `EOS` on the bus to terminate the pipeline must not rely on receiving it from the downstream side of `DxOutputSelector`. Use the per-stream wrapped `EOS` (unwrapped to the matching src pad) or trigger termination based on application logic (e.g., a timeout, or `EOS` from another sink).

© Copyright 2026 DEEPX All Rights Reserved.

3.10 DxRate

DxRate is an element that adjusts the framerate of a video stream to match a defined target framerate.
DxRate achieves this by dropping or duplicating frames based on the timestamps of incoming buffers.
DxRate always recomputes output buffer timestamps based on the target framerate, producing evenly spaced output PTS regardless of the input PTS pattern. There is no conditional passthrough — even when the input framerate already matches the target, the output PTS is regenerated from the target rate.

Key Features

Framerate Adjustment

- Ensures the output stream matches the target `framerate` by dropping or duplicating frames.
- The output buffer's timestamps are adjusted relative to the first input buffer's timestamp.
- A buffer with an invalid timestamp reuses the last known timestamp; if none exists, the buffer is dropped with a warning.
- Reverse playback (negative rate) is not supported and results in an error.

Throttle QoS Event

- If `throttle` is set to `true`, a Throttle QoS Event is sent upstream when frames are dropped.
- **DxInfer** can respond to this event by applying throttling, using the `throttling_delay` value.
- To enable this function properly, **DxRate must** be placed downstream of **DxInfer** in the pipeline.

Latency Reporting

- Adds one frame duration (`1/framerate`) to the latency reported in response to `LATENCY` queries.

Framerate and Video Speed

- **Framerate** refers to the number of frames per second (FPS) for visual smooth playback.
- **Video speed** refers to playback speed (e.g., fast-forward), which is a separate concept.

Hierarchy

ⓒ Copyright 2026 DEEPX All Rights Reserved.

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

Pad Templates

Sink (input)

Property	Value
Format	`video/x-raw; application/x-dxvideoraw`

Src (output)

Property	Value
Format	`video/x-raw; application/x-dxvideoraw`

Properties

Name	Description	Type	Default Value
`name`	Sets the unique name of the DxRate element.	String	`"dxrate0"`
`framerate`	Sets the target framerate (FPS, range: 0-10000). This property must be configured.	Unsigned Integer	`0`
`throttle`	Determines whether to send Throttle QoS Events upstream on frame drops.	Boolean	`false`

Domain Mode Behavior

ⓒ Copyright 2026 DEEPX All Rights Reserved.

`dxrate` is a dual-mode element (see Multi-Stream Domain).

- **Normal mode** (sink caps = `video/x-raw`): a single rate state controls drop/duplicate decisions for the single stream.
- **Domain mode** (sink caps = `application/x-dxvideoraw`): per-stream rate state. Each `_stream_id` from `DXFrameMeta` has its own first-PTS reference, drop counter and duplicate counter, so framerate adjustment is applied independently per stream.

> **NOTE**
>
> - The `framerate` property is mandatory and **must** be explicitly set for the element to function. Setting `framerate=0` results in a `GST_ELEMENT_ERROR` when the element starts.

ⓒ Copyright 2026 DEEPX All Rights Reserved.

3.11 DxMsgConv

DxMsgConv is an element that processes inference metadata from upstream **DxPostprocess** elements and converts it into message payloads in various formats.

This element bridges internal AI pipeline processing and external communication systems. It receives the raw inference metadata—data about detected objects, classifications, and tracking—from the upstream **DxPostprocess** elements. **DxMsgConv** then transforms this complex metadata into structured message payloads using various standard formats, such as
JSON or Protocol Buffers. Once converted, these payloads are passed downstream to the **DxMsgBroker** element, which is responsible for transmitting them reliably to an external broker server (like an MQTT or Kafka broker).

To properly implement the specific structure and content required for the final messages—referred to as the "actual message format logic"—a user-defined custom library is required. This library contains the unique code that dictates how the inference metadata is mapped into the desired message payload structure. For guidance on developing and integrating this custom logic, users must consult the relevant documentation, specifically: **Chapter. Writing Your Own Application "Custom Message Convert Library".**

Key Features

Metadata Conversion

- Converts metadata generated by **DxPostprocess** into structured message formats through custom library implementation.
- The actual message format and structure are defined by the user-implemented custom library.
- Requires a custom shared library to define and implement the required message formats.

Integration with DxMsgBroker

- Passes formatted message payloads (as generated by the custom library) downstream to **DxMsgBroker**, which handles delivery to external systems.

Configuration

- Element properties (`library-file-path` , `message-interval` , `include-frame`) can also be set through a JSON configuration file (`config-file-path`).

Example Output Format

ⓒ Copyright 2026 DEEPX All Rights Reserved.

The element processes metadata through the custom library to generate message payloads. The default example custom library provided with DX-STREAM generates JSON messages with comprehensive object detection information:

```
{
  "streamId": 0,
  "seqId": 123,
  "width": 1920,
  "height": 1080,
  "objects": [
    {
      "object": {
        "label_id": 1,
        "track_id": 42,
        "confidence": 0.87,
        "name": "person",
        "box": { ... },
        "body_feature": [...],
        "segment": { ... },
        "pose": { ... },
        "face": { ... }
      }
    }
  ]
}
```

NOTE

The actual message format depends entirely on the implementation of your custom library. The above is just an example from the provided default library. For detailed implementation guidance and complete JSON structure examples, refer to **Chapter 4. Writing Your Own Application**.

In the default library, frame-level segmentation remains a full-frame semantic class map, while object-level `segment` payloads are ROI-local binary masks aligned to the corresponding `object.box`.

Hierarchy

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


© Copyright 2026 DEEPX All Rights Reserved.

Pad Templates

Sink (input)

Property	Value
Format	`video/x-raw, format=(string){ NV12, I420, RGB, BGR }; application/x-dxvideoraw`

Src (output)

Property	Value
Format	`video/x-raw, format=(string){ NV12, I420, RGB, BGR }; application/x-dxvideoraw`

`DxMsgConv` is essentially **caps-agnostic** — it does not inspect raw video pixels. It accepts either `video/x-raw` or `application/x-dxvideoraw` solely so it can be placed anywhere in the chain; processing operates on the buffer's `DXFrameMeta` / `DXObjectMeta` only.

Custom Library Error Reporting

Custom message-converter libraries must follow the error reporting contract described in **Chapter. Writing Your Own Application "Error Reporting from Custom Libraries"** — no `g_error()` / `abort()`; use `g_warning()` + `return nullptr` for per-frame skips, and `throw std::runtime_error` for permanent errors.

Properties

Name	Description	Type	Default Value
`config-file-path`	Path to the configuration file. (optional).	String	`null`
`library-file-path`	Path to the custom message converter library. **Required**.	String	`null`
`message-interval`	Frame interval at which message is	Integer	`1`

© Copyright 2026 DEEPX All Rights Reserved.

Name	Description	Type	Default Value
	converted (range: 1-10000).		
include-frame	Flag whether to include frame data as base64 JPEG in the message. (optional).	Boolean	false

Limitation

DxMsgConv must not be placed after GStreamer's built-in `input-selector` element (this does not apply to **DxInputSelector**). Caps are configured once via `set_caps` callback and are not re-negotiated during stream switching. Placing **DxMsgConv** downstream of `input-selector` may result in caps mismatch when streams are switched at runtime.

NOTE

- The element itself does not define the message format - this is entirely determined by your custom library implementation.
- The default example library provided with DX-STREAM generates JSON format, but you can implement any message format in your custom library.

© Copyright 2026 DEEPX All Rights Reserved.

3.12 DxMsgBroker

DxMsgBroker is a sink element that transmits payload messages to an external message broker (e.g., MQTT, Kafka)
It is typically placed after the **DxMsgConv** element in the pipeline and is responsible for delivering the formatted messages to the configured broker server.
Connection settings are managed using the `conn-info` property.

Key Features

Message Sending

- Receives payloads from upstream elements and publishes them to the selected broker.
- Compatible with broker systems such as MQTT and Kafka.
- If the broker is not connected or publishing fails, the message is dropped with a warning. After 10 consecutive failures, the element posts an error and the pipeline stops. A successful publish resets the counter.

Connection Information

- The `conn-info` property specifies the connection in the format `host:port`.
- Additional connection details (e.g., `SSL/TLS` settings) can be specified through an optional configuration file.

Pipeline Integration

- Used as a sink element, typically placed just after **DxMsgConv** in the pipeline.

Hierarchy

```
GObject
 +----GInitiallyUnowned
      +----GstObject
           +----GstElement
                +----GstBaseSink
                     +----GstDxMsgBroker
```

Pad Templates

ⓒ Copyright 2026 DEEPX All Rights Reserved.

Sink (input)

Property	Value
Format	`video/x-raw; application/x-dxvideoraw`

`DxMsgBroker` is caps-agnostic — it only reads message payloads from buffer metadata produced by `DxMsgConv`, so it accepts both single-stream `video/x-raw` and the multi-stream `application/x-dxvideoraw` domain caps.

Properties

Name	Description	Type	Default Value
`broker-name`	Name of message broker system (mqtt or kafka). **Required**.	String	`"mqtt"`
`conn-info`	Connection info in the format `host:port`. **Required**.	String	`null`
`topic`	The topic name for publishing messages. **Required**.	String	`null`
`config`	Path to the broker configuration file (optional).	String	`null`

© Copyright 2026 DEEPX All Rights Reserved.

NOTE

- The `broker-name` property determines which broker type to use. Supported values are `"mqtt"` and `"kafka"`. While "mqtt" is set as the initial value, this property is **required** to be explicitly set.
- The `conn-info` property is **required** and **must** be set to establish a connection to the broker in the `host:port` format.
- The `topic` property is **required** and defines the topic for message publication.
- An optional configuration file can provide advanced settings, such as `SSL/TLS` encryption and authentication details.
- Ensure the `libmosquitto-dev` (for MQTT) and `librdkafka-dev` (for Kafka) libraries are installed for proper broker support.
- The element uses a **Broker Abstraction Layer (BAL)** that provides a unified interface for different broker types, making it easy to switch between MQTT and Kafka without changing the pipeline structure.

Property validation

`broker-name`, `conn-info`, and `topic` are validated when the element starts. If `broker-name` is not `mqtt` or `kafka`, or if `conn-info` or `topic` is empty, the element posts a `GST_ELEMENT_ERROR` and fails to start. Always set `conn-info` and `topic` explicitly.

© Copyright 2026 DEEPX All Rights Reserved.

3.13 DxScale

DxScale is an element that scales video frames to a specified target resolution.
DxScale uses the `VideoTransformFactory` to automatically select the best available hardware-accelerated backend. The selection priority is: V3 DSP > VNPU > RGA > libyuv (software fallback).

Key Features

Resolution Scaling

- Scales input video frames to the resolution specified by `width` and `height` properties.
- Supports NV12, I420, RGB, and BGR pixel formats (same format on input and output — scale only, no conversion).

Passthrough Mode

- When `width` and `height` are both set to `0` (default), the element passes frames through without any modification.

Automatic Backend Selection

- The `VideoTransformFactory` selects the best available backend automatically:
- **V3 DSP** – DEEPX V3 device DSP (when the V3 sibling plugin is installed)
- **VNPU** – DEEPX VNPU hardware backend (when the VNPU sibling plugin is installed)
- **RGA** – Rockchip Raster Graphic Accelerator (when the RGA sibling plugin is installed)
- **libyuv** – Software fallback (always available)

Hierarchy

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

Pad Templates

ⓒ Copyright 2026 DEEPX All Rights Reserved.

Sink (input)

Property	Value
Format	`video/x-raw, format=(string){ NV12, I420, RGB, BGR }`

Src (output)

Property	Value
Format	`video/x-raw, format=(string){ NV12, I420, RGB, BGR }`

Properties

Name	Description	Type	Default Value	Range
`width`	Target output width. Set to `0` for passthrough.	Unsigned Integer	`0`	`0 – 8192`
`height`	Target output height. Set to `0` for passthrough.	Unsigned Integer	`0`	`0 – 8192`

Usage Example

Scale an I420 video stream from 1920×1080 to 640×480:

```
gst-launch-1.0 \
  filesrc location=input.mp4 ! decodebin ! \
  dxscale width=640 height=480 ! \
  videoconvert ! autovideosink
```

Scale NV12 decoder output before inference:

```
gst-launch-1.0 \
  filesrc location=input.mp4 ! decodebin ! \
  dxscale width=640 height=640 ! \
  dxpreprocess preprocess-id=1 ... ! \
  dxinfer ...
```


© Copyright 2026 DEEPX All Rights Reserved.

NOTE

- `dxscale` does **not** perform color conversion. Input and output must be the same format.
- For color conversion, use **DxConvert**.
- Setting both `width` and `height` together is strongly recommended. If only one is set (the other left at `0`), the element does **not** raise an error — it silently falls back to passthrough behavior, which may not match user intent. Set both explicitly when scaling is required.

DxInputSelector Placement

`dxscale` must **not** be placed downstream of **DxInputSelector**. DxInputSelector merges multiple streams with potentially different resolutions into a single output, causing caps to change on every buffer. Since `dxscale` is a `GstBaseTransform` element, it requires stable caps for buffer pool allocation and kernel initialization. Placing it after DxInputSelector will cause repeated caps renegotiation, kernel re-creation, and potential caps/buffer mismatch errors. Place `dxscale` **before** DxInputSelector (per-stream) or **after** DxOutputSelector (per-stream).

ⓒ Copyright 2026 DEEPX All Rights Reserved.

3.14 DxConvert

DxConvert is an element that converts video frames between different color formats. **DxConvert** uses the `VideoTransformFactory` to automatically select the best available hardware-accelerated backend. The selection priority is: V3 DSP > VNPU > RGA > libyuv (software fallback).

Key Features

Color Format Conversion

- Converts video frames between NV12, I420, RGB, and BGR pixel formats.
- Supports all 4×4 format combinations (e.g., NV12→RGB, I420→BGR, RGB→NV12, etc.).

Same-Format Passthrough

- When the input and output formats are identical, the element passes frames through without any modification (in-place).

Automatic Backend Selection

- The `VideoTransformFactory` selects the best available backend automatically:
- **V3 DSP** – DEEPX V3 device DSP (when the V3 sibling plugin is installed)
- **VNPU** – DEEPX VNPU hardware backend (when the VNPU sibling plugin is installed)
- **RGA** – Rockchip Raster Graphic Accelerator (when the RGA sibling plugin is installed)
- **libyuv** – Software fallback (always available)

Hierarchy

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

Pad Templates

ⓒ Copyright 2026 DEEPX All Rights Reserved.

Sink (input)

Property	Value
Format	`video/x-raw, format=(string){ NV12, I420, RGB, BGR }`

Src (output)

Property	Value
Format	`video/x-raw, format=(string){ NV12, I420, RGB, BGR }`

Properties

DxConvert has no configurable properties. The target format is negotiated automatically through GStreamer caps negotiation.

Usage Example

Convert an NV12 stream to RGB:

```
gst-launch-1.0 \
  filesrc location=input.mp4 ! decodebin ! \
  video/x-raw,format=NV12 ! \
  dxconvert ! \
  video/x-raw,format=RGB ! \
  autovideosink
```

Convert I420 to RGB before inference:

```
gst-launch-1.0 \
  filesrc location=input.mp4 ! decodebin ! \
  video/x-raw,format=I420 ! \
  dxconvert ! \
  video/x-raw,format=RGB ! \
  dxpreprocess preprocess-id=1 ... ! \
  dxinfer ...
```

Combine with DxScale — convert format and resize in a pipeline:

```
gst-launch-1.0 \
  filesrc location=input.mp4 ! decodebin ! \
```


© Copyright 2026 DEEPX All Rights Reserved.

```
video/x-raw,format=NV12 ! \
dxconvert ! \
video/x-raw,format=RGB ! \
dxscale width=640 height=480 ! \
autovideosink
```

NOTE

- `dxconvert` does **not** perform scaling. Input and output resolutions must match.
- For video scaling, use **DxScale**.
- Which conversions are hardware-accelerated depends on the platform and the installed backends. Combinations that no hardware backend supports use the libyuv software backend.

DxInputSelector Placement

`dxconvert` must **not** be placed downstream of **DxInputSelector**. DxInputSelector merges multiple streams with potentially different color formats into a single output, causing caps to change on every buffer. Since `dxconvert` is a `GstBaseTransform` element, it requires stable caps for buffer pool allocation and kernel initialization. Placing it after DxInputSelector will cause repeated caps renegotiation, kernel re-creation, and potential caps/buffer mismatch errors. Place `dxconvert` **before** DxInputSelector (per-stream) or **after** DxOutputSelector (per-stream).

© Copyright 2026 DEEPX All Rights Reserved.

4. Writing Your Own Application

This chapter describes how to integrate a custom AI model and implement user-defined logic within the **DX-STREAM** pipeline. It assumes that your model has already been compiled to `.dxnn` format using **DX-COM**. For details on model compilation, refer to **DX-COM User Manual**.

This guide focuses on how to configure and integrate custom logic into the **DX-STREAM** pipeline using modular elements such as **DxPreprocess**, **DxInfer**, and **DxPostprocess**.

4.1 API Migration Notes (v3.1.0)

The following changes were introduced in DX-STREAM v3.1.0 to improve metadata handling and eliminate circular references:

Metadata API Changes:

- The `frame_meta->_buf` member has been removed to eliminate circular references
- All custom library functions now receive `GstBuffer *buf` as the first parameter for direct buffer access
- Object creation has changed from `dx_create_object_meta(buf)` to `dx_acquire_obj_meta_from_pool()`
- Use `dx_add_obj_meta_to_frame()` to add objects to frame metadata
- Custom libraries must be updated to use the new function signatures

Inference Mode Differences:

Primary Mode

- Inference is performed on the entire frame
- Postprocessing is responsible for creating new objects (`DXObjectMeta`) based on the model's output
- Use `dx_acquire_obj_meta_from_pool()` to create new objects and `dx_add_obj_meta_to_frame()` to add them to the frame
- These new objects are then added to the associated `DXFrameMeta`

© Copyright 2026 DEEPX All Rights Reserved.

Secondary Mode

- Inference is performed per object, based on existing `DXObjectMeta` in buffer
- Postprocessing is applied to modify or enrich existing object metadata
- The `DXObjectMeta` structure contains the input object information, which is passed to the postprocess function for update or enhancement

4.2 DX-STREAM Metadata System Overview

DX-STREAM provides a comprehensive metadata framework for handling inference results and custom data throughout the pipeline. The system is designed with a hierarchical structure that enables efficient data organization and access.

4.2.1 Metadata Architecture

DX-STREAM uses a hierarchical metadata structure that follows this organization:

Buffer → Frame → Object → User Meta

- **GstBuffer**: Contains video frame data and top-level frame metadata
- **DXFrameMeta**: Frame-level metadata (dimensions, stream info, object list)
- **DXObjectMeta**: Object-level metadata (detection results, features)
- **DXUserMeta**: User-defined custom metadata attached to frames or objects

4.2.2 Core Metadata Types

DXFrameMeta Structure:

```
struct _DXFrameMeta {
    GstMeta _meta;

    int _stream_id;
    int _width;
    int _height;
    std::string _format;
    std::string _name;
    float _frame_rate;

    int _roi[4];

    // segmentation
    std::vector<unsigned char> _seg_data;
    int _seg_width = 0;
    int _seg_height = 0;
```


© Copyright 2026 DEEPX All Rights Reserved.

```
    // depth
    std::vector<unsigned char> _depth_data;
    int _depth_width = 0;
    int _depth_height = 0;

    // classification result (primary mode)
    int _label;
    std::string _label_name;
    float _label_confidence;

    std::vector<DXObjectMeta*> _object_meta_list;

    std::vector<DXUserMeta*> _frame_user_meta_list;

    // RAII-managed tensors (shallow copy through shared_ptr)
    std::map<int, dxs::DXTensors> _input_tensors;   // preproc_id -> input tensors
    std::map<int, dxs::DXTensors> _output_tensors;   // infer_id -> output tensors
};
```

DXObjectMeta Structure:

```
struct _DXObjectMeta {
    int _meta_id;

    // body
    int _track_id;
    int _label;
    std::string _label_name;
    float _confidence;
    std::array<float, 4> _box;
    std::vector<float> _keypoints;
    std::vector<float> _body_feature;

    // oriented bounding box [cx, cy, w, h, angle]
    std::vector<float> _obb;

    // face
    std::array<float, 4> _face_box;
    float _face_confidence;
    std::vector<float> _face_landmarks;
    std::vector<float> _face_feature;

    // segmentation
    std::vector<unsigned char> _seg_data;
    int _seg_width = 0;
    int _seg_height = 0;

    // user meta
    std::vector<DXUserMeta*> _obj_user_meta_list;
```


ⓒ Copyright 2026 DEEPX All Rights Reserved.

```
    // RAII-managed tensors (shallow copy through shared_ptr)
    std::map<int, dxs::DXTensors> _input_tensors;   // preproc_id -> input tensors
    std::map<int, dxs::DXTensors> _output_tensors;   // infer_id -> output tensors

};
```

Segmentation Note:

- `DXFrameMeta._seg_data`, `_seg_width`, and `_seg_height` store a frame-level semantic class map.
- `DXFrameMeta._depth_data`, `_depth_width`, and `_depth_height` store a row-major, single-channel, 8-bit relative depth map.
- The YOLO26 depth postprocessor normalizes each frame independently to `[0, 255]`; values cannot be compared as absolute distances across frames.
- `DXObjectMeta._seg_data`, `_seg_width`, and `_seg_height` store an ROI-local binary mask aligned to `_box`.
- Legacy `SegClsMap` is no longer used for object metadata.

4.2.3 Metadata API Functions

Frame Metadata Operations:

```
// Create and access frame metadata
GstBuffer *dx_create_frame_meta(GstBuffer *buffer);
DXFrameMeta *dx_get_frame_meta(GstBuffer *buffer);

// Object management in frame
gboolean dx_add_obj_meta_to_frame(DXFrameMeta *frame_meta, DXObjectMeta *obj_meta);
gboolean dx_remove_obj_meta_from_frame(DXFrameMeta *frame_meta, DXObjectMeta *obj_meta)
;
```

Object Metadata Operations:

```
// Object lifecycle management
DXObjectMeta* dx_acquire_obj_meta_from_pool(void);
void dx_release_obj_meta(DXObjectMeta *obj_meta);
void dx_copy_obj_meta(DXObjectMeta *src_meta, DXObjectMeta *dst_meta);
```

User Metadata Operations:

```
// User metadata lifecycle
DXUserMeta* dx_acquire_user_meta_from_pool(void);
void dx_release_user_meta(DXUserMeta *user_meta);

// Data management with required safety functions
```

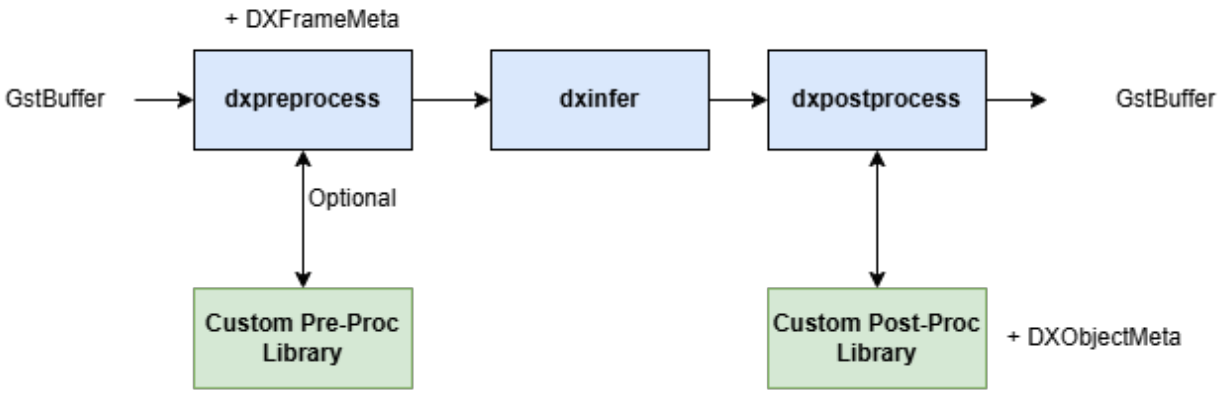

© Copyright 2026 DEEPX All Rights Reserved.

```
gboolean dx_user_meta_set_data(DXUserMeta *user_meta,
                        gpointer data,
                        gsize size,
                        DXUserMetaType meta_type,
                        GDestroyNotify release_func,    // Required: cleanup function
                        GBoxedCopyFunc copy_func);      // Required: copy function

// Attachment to frame/object
gboolean dx_add_user_meta_to_frame(DXFrameMeta *frame_meta, DXUserMeta *user_meta);
gboolean dx_add_user_meta_to_obj(DXObjectMeta *obj_meta, DXUserMeta *user_meta);

// Retrieval (returns std::vector pointer)
std::vector<DXUserMeta*>* dx_get_frame_user_metas(DXFrameMeta *frame_meta);
std::vector<DXUserMeta*>* dx_get_object_user_metas(DXObjectMeta *obj_meta);
```

4.3 Custom Library for Model Inference



The **DX-STREAM** inference pipeline is composed of the following elements:

DxPreprocess

- Allocates `DXFrameMeta` based on the `GstBuffer` received from upstream
- Performs the preprocessing algorithm as defined by element properties
- For custom preprocessing algorithms, a **Custom Pre-Process Library** can be built and integrated
- See the **dxpreprocess** section in the Elements documentation for details

DxInfer

- Performs inference using the selected backend (**DX-RT** or VNPU)
- Receives input tensors from `dxpreprocess` and generates output tensors
- Selects the inference backend via the `backend` property (`auto` (default), `dxrt` , `dxvnpu`)

© Copyright 2026 DEEPX All Rights Reserved.

DxPostprocess

- Receives the output tensor from `dxinfer`
- Executes the custom postprocessing algorithm defined in a custom library
- A custom postprocessing implementation is required for each model
- Example libraries for common vision tasks can be found in `dx_stream/custom_library/postprocess_library`

4.3.1 Writing Custom Pre-Process Function

For models requiring additional preprocessing beyond the default functionality, you can implement a **Custom Pre-Process Function** using a user-defined library.

Implementation Example

```
extern "C" bool CustomPreprocessFunc(GstBuffer *buf,
                                     DXFrameMeta *frame_meta,
                                     DXObjectMeta *object_meta,
                                     void* input_tensor)
{
   // Preprocessing logic
   return true;
}
```

Function Parameters:

GstBuffer

- Direct access to the GStreamer buffer containing the frame data
- Replaces the previous indirect access through `frame_meta->_buf`

DXFrameMeta

- Contains frame-level metadata such as dimensions, format, and stream information
- No longer contains the `_buf` member - buffer access is provided through the first parameter

DXObjectMeta

- In **Secondary Mode**, metadata for each object is passed to the function
- In **Primary Mode**, no object metadata is available (`nullptr`)

© Copyright 2026 DEEPX All Rights Reserved.

input_tensor

- The address of the input tensor generated through user-defined preprocessing
- It is pre-allocated based on the input tensor size specified by the `dxpreprocess` property and passed to the user
- Users **must not** free or reallocate this memory

Library Integration

To build the custom preprocess library, use a `meson.build` file and compile as follows:

```
gst_dep = dependency('gstreamer-1.0', version : '>=1.16.3',
    required : true, fallback : ['gstreamer', 'gst_dep'])

dx_stream_dep = dependency('gstdxstream')

libcustompreproc = shared_library('custompreproc',
    'preprocess.cpp',
    dependencies: [gst_dep, dx_stream_dep],
    install: true,
    install_dir: get_option('datadir') / 'gstdxstream' / 'lib'
)
```

Specify the library path and function name in the JSON configuration file for `dxpreprocess` as follows:

```
{
    "library_file_path": "/usr/local/share/gstdxstream/lib/libcustompreproc.so",
    "function_name": "CustomPreprocessFunc"
}
```

4.3.2 Writing Custom Post-Process Function

Postprocessing is essential for interpreting and converting the model's output tensor into meaningful results. To do this, a custom post-process library **must** be implemented to match your model's architecture and output format.

Output Tensor Parsing

To check the structure of the output tensor, use the following command. This prints the tensor shape for each output:

```
$ dxparse -m yolo26-n_640x640.dxnn
```


© Copyright 2026 DEEPX All Rights Reserved.

> **NOTE**
>
> `dxparse` is the standard CLI tool. Legacy name `parse_model` remains available as a backward-compatible alias.

Example output:

```
outputs:
  onnx::Reshape_491, FLOAT, [1, 80, 80, 256]
  onnx::Reshape_525, FLOAT, [1, 40, 40, 256]
  onnx::Reshape_559, FLOAT, [1, 20, 20, 256]
```

The example shows three blobs with NHWC dimensions. Use this information to implement the custom postprocessing logic.

Implementation Example

```
extern "C" void PostProcess(GstBuffer *buf,
                  std::vector<dxs::DXTensor> network_output,
                  DXFrameMeta *frame_meta,
                  DXObjectMeta *object_meta)
{
    // Access tensor data using struct members
    float *output_data = (float *)network_output[0]._data;
    auto shape = network_output[0]._shape;
    int batch = shape[0];
    int height = shape[1];
    int width = shape[2];
    int channels = shape[3];

    // Convert output tensor to bounding box information

    // Example of creating new object metadata:
    DXObjectMeta *obj_meta = dx_acquire_obj_meta_from_pool();
    // ... populate object metadata ...

    // Add object to frame
    dx_add_obj_meta_to_frame(frame_meta, obj_meta);
}
```


ⓒ Copyright 2026 DEEPX All Rights Reserved.

Function Parameters:

- **GstBuffer *buf**: Direct access to the GStreamer buffer containing frame data
- **std::vector\<dxs::DXTensor> network_output**: Output tensors from the inference engine (defined in `dxcommon.hpp`)
- **DXFrameMeta *frame_meta**: Frame-level metadata (dimensions, format, etc.)
- **DXObjectMeta *object_meta**: Object-level metadata (in Secondary Mode) or nullptr (in Primary Mode)

Tensor Access Members:

- `network_output[i]._data`: Get pointer to tensor data (void*)
- `network_output[i]._shape`: Get tensor shape as std::vector
- `network_output[i]._type`: Get tensor data type (dxs::DataType)
- `network_output[i]._elemSize`: Get size of each element
- `network_output[i]._name`: Get tensor name

Library Integration

Build the custom library using a `meson.build` script:

```
project('postprocess_yolo26od', 'cpp', version : '1.0.0', license : 'LGPL', default_options:
['cpp_std=c++14'])

gst_dep = dependency('gstreamer-1.0', version : '>=1.16.3',
    required : true, fallback : ['gstreamer', 'gst_dep'])

dx_stream_dep = dependency('gstdxstream')
yolo26_postprocess_lib = shared_library('postprocess_yolo26od',
    'postprocess.cpp',
    dependencies: [gst_dep, dx_stream_dep],
    install: true,
    install_dir: get_option('datadir') / 'gstdxstream' / 'lib'
)
```

Specify the library path and function name in the JSON configuration file for `dxpostprocess` as follows.

```
{
    "library_file_path": "/usr/local/share/gstdxstream/lib/libpostprocess_yolo26od.so",
    "function_name": "PostProcess"
}
```

Differences in Post-Processing Logic Based on Inference Mode

ⓒ Copyright 2026 DEEPX All Rights Reserved.

Primary Mode

- Inference is performed on the entire frame
- Postprocessing is responsible for creating new objects (`DXObjectMeta`) based on the model's output
- Use `dx_acquire_obj_meta_from_pool()` to create new objects and `dx_add_obj_meta_to_frame()` to add them to the frame
- These new objects are then added to the associated `DXFrameMeta`

Secondary Mode

- Inference is performed per object, based on existing `DXObjectMeta` in buffer
- Postprocessing is applied to modify or enrich existing object metadata
- The `DXObjectMeta` structure contains the input object information, which is passed to the postprocess function for update or enhancement

4.3.3 Using User Metadata in Custom Libraries

The DX-STREAM framework provides a simplified user metadata system for storing custom data. The system supports two main categories of user metadata:

User Meta Types:

```
enum class DXUserMetaType {
    DX_USER_META_FRAME = 0x1000,   // Frame-level user metadata
    DX_USER_META_OBJECT = 0x2000,  // Object-level user metadata
};
```

DXUserMeta Structure:

```
struct _DXUserMeta {
    gpointer user_meta_data;       // Pointer to user data
    gsize user_meta_size;          // Size of user data
    DXUserMetaType user_meta_type;  // Type (FRAME or OBJECT)

    GDestroyNotify release_func;    // Required: data cleanup function
    GBoxedCopyFunc copy_func;       // Required: data copy function
};
```

Adding Custom Metadata to Frame:

```
// Define custom data structure
typedef struct {
```


© Copyright 2026 DEEPX All Rights Reserved.

```
    gint custom_id;
    gchar *custom_name;
    gfloat custom_score;
} MyFrameData;

// Copy function for your data
static gpointer my_frame_data_copy(gconstpointer src) {
    const MyFrameData *src_data = (const MyFrameData *)src;
    MyFrameData *dst_data = g_new0(MyFrameData, 1);
    dst_data->custom_id = src_data->custom_id;
    dst_data->custom_name = g_strdup(src_data->custom_name);
    dst_data->custom_score = src_data->custom_score;
    return dst_data;
}

// Cleanup function for your data
static void my_frame_data_free(gpointer data) {
    MyFrameData *frame_data = (MyFrameData *)data;
    g_free(frame_data->custom_name);
    g_free(frame_data);
}

// Create and set user metadata
DXUserMeta *user_meta = dx_acquire_user_meta_from_pool();

MyFrameData *custom_data = g_new0(MyFrameData, 1);
custom_data->custom_id = 123;
custom_data->custom_name = g_strdup("example_frame");
custom_data->custom_score = 0.95f;

// Set data with required copy and release functions
dx_user_meta_set_data(user_meta,
                      custom_data,
                      sizeof(MyFrameData),
                      DXUserMetaType::DX_USER_META_FRAME,
                      my_frame_data_free,         // Required cleanup function
                      my_frame_data_copy);        // Required copy function

// Add to frame
dx_add_user_meta_to_frame(frame_meta, user_meta);
```

Retrieving User Metadata:

```
// Get all frame user metadata
auto frame_metas = dx_get_frame_user_metas(frame_meta);
for (auto user_meta : *frame_metas) {
    // Check if this is frame-type metadata
    if (user_meta->user_meta_type == DXUserMetaType::DX_USER_META_FRAME) {
        MyFrameData *data = (MyFrameData *)user_meta->user_meta_data;
        g_print("Frame data: ID=%d, Name=%s, Score=%.2f\n",
                data->custom_id, data->custom_name, data->custom_score);
```


© Copyright 2026 DEEPX All Rights Reserved.

```
    }
}
```

Important Safety Requirements:

- **Copy Function**: Always provide a proper copy function that performs deep copy of your data
- **Release Function**: Always provide a cleanup function that properly frees all allocated memory
- **Memory Management**: The UserMeta system will automatically handle lifecycle management using your provided functions
- **Type Checking**: Always verify the metadata type before casting to your custom structure

4.3.4 Error Reporting from Custom Libraries

Custom pre-process, post-process, and message-convert libraries are loaded via `dlopen` and called from inside the host element (`dxpreprocess` / `dxpostprocess` / `dxmsgconv`). The host wraps every call in `try { ... } catch (std::exception&) catch (...)` and converts any thrown exception into `GST_ELEMENT_ERROR(LIBRARY, FAILED, ...)`, which is delivered on the GStreamer bus so that the application can shut the pipeline down cleanly.

To preserve this contract, custom libraries **must** follow these rules:

- **Do not use `g_error()`, `g_assert()`, `abort()`, or `exit()`.** `g_error()` calls `G_BREAKPOINT()` → `abort()`, which terminates the process via `SIGABRT`. C++ stack unwinding is skipped, so the host `try/catch` cannot intercept it; bus error messages are never delivered; and `dlclose()`, push-thread join, and other resource cleanup never run. The result is a core dump instead of a clean EOS / NULL-state transition.
- **Per-frame recoverable errors** (missing input tensor, shape mismatch on a single frame, optional metadata absent, etc.) → use `g_warning()` and return early (`false` for pre-process, plain `return` for post-process, `nullptr` for message convert). The host element skips that frame and the pipeline continues.
- **Permanent errors** (config does not match the loaded model, required resource missing, invariant violated) → `throw std::runtime_error("descriptive message")`. The host element catches it, reports `GST_ELEMENT_ERROR`, and the application gets a normal bus error from which it can transition to `NULL`.

Example (post-process library):

```
extern "C" void PostProcess(GstBuffer *buf,
                            std::vector<dxs::DXTensor> network_output,
                            DXFrameMeta *frame_meta,
                            DXObjectMeta *object_meta) {
```


© Copyright 2026 DEEPX All Rights Reserved.

```
    // Recoverable: skip this frame, keep pipeline running.
    if (network_output.empty()) {
        g_warning("PostProcess: no output tensors for this frame, skipping");
        return;
    }

    // Permanent: model/config mismatch — let the host element report it.
    if (network_output[0]._shape.size() != 4) {
        throw std::runtime_error("PostProcess: unexpected tensor rank, "
                                 "check that model matches the configured library");
    }

    // ... normal processing ...
}
```

4.4 Custom Message Convert Library

Custom message conversion in **DX-STREAM** requires implementing a user-defined library that converts inference metadata into the desired message format (typically JSON).

The library converts comprehensive object detection metadata including:

- **Object Detection**: label_id, track_id, confidence, name, bounding box
- **Body Features**: extracted body feature vectors for re-identification
- **Segmentation**: object-level ROI binary masks with height, width, format, box, and data pointer
- **Pose Estimation**: 17 keypoints with coordinates (kx, ky) and confidence scores (ks)
- **Face Detection**: landmarks, face bounding box, confidence, and face feature vectors

4.4.1 Functions to Implement

Your custom library **must** define the following three functions.

- `dxmsg_create_context` : Initializes the message conversion context
- `dxmsg_delete_context` : Deletes and releases all resources associated with the context
- `dxmsg_convert_payload` : Converts the metadata into the target message format

4.4.2 Implementation Example

The custom library implementation consists of the main interface functions and helper functions for JSON conversion:

ⓒ Copyright 2026 DEEPX All Rights Reserved.

```
#include "dx_msgconvl_priv.hpp"

extern "C" DxMsgContext *dxmsg_create_context() {
    DxMsgContext *context = g_new0(DxMsgContext, 1);
    context->_priv_data = (void *)dxcontext_create_contextPriv();
    return context;
}

extern "C" void dxmsg_delete_context(DxMsgContext *context) {
    g_return_if_fail(context != nullptr);
    dxcontext_delete_contextPriv((DxMsgContextPriv *)context->_priv_data);
    g_free(context);
}

extern "C" DxMsgPayload *dxmsg_convert_payload(DxMsgContext *context,
                                               GstDxMsgMetaInfo *meta_info) {
    DxMsgPayload *payload = g_new0(DxMsgPayload, 1);
    if (!payload) {
        g_warning("Failed to allocate DxMsgPayload");
        return nullptr;
    }

    gchar *json_data = dxpayload_convert_to_json(context, meta_info);
    if (json_data == nullptr) {
        g_warning("dxpayload_convert_to_json returned null");
        g_free(payload);
        return nullptr;
    }

    payload->_size = strlen(json_data);
    payload->_data = json_data;
    return payload;
}
```

Helper Functions Implementation:

The core JSON conversion logic is implemented in helper functions:

```
// Private context management
DxMsgContextPriv *dxcontext_create_contextPriv(void);
void dxcontext_delete_contextPriv(DxMsgContextPriv *contextPriv);

// Main JSON conversion function
gchar *dxpayload_convert_to_json(DxMsgContext *context, GstDxMsgMetaInfo *meta_info);
```

The `dxpayload_convert_to_json` function processes the metadata and generates the final JSON string using json-glib library functions. The returned JSON data is automatically freed by the DxMsgConv element after transmission.

© Copyright 2026 DEEPX All Rights Reserved.

When `include-frame` is enabled on DxMsgConv, `meta_info->_frame_base64` contains the base64-encoded JPEG frame data. Custom libraries can include this in payloads:

```
if (meta_info->_frame_base64) {
    json_object_set_string_member(root, "frameData", meta_info->_frame_base64);
}
```

NOTE

Even when `include-frame` is set to `true`, `_frame_base64` may be `nullptr` if frame encoding fails (e.g., unsupported format, transform error). Always check for `nullptr` before using `_frame_base64`.

4.4.3 JSON Output Example

The example `dxpayload_convert_to_json` function implementation generates structured JSON messages by processing metadata from `DXFrameMeta` and `DXObjectMeta` structures. The function uses json-glib library to construct the JSON output:

JSON Structure Overview:

```
// Create root JSON object with frame metadata
json_object_set_int_member(jobj_root, "streamId", frame_meta->_stream_id);
json_object_set_int_member(jobj_root, "seqId", meta_info->_seq_id);
json_object_set_int_member(jobj_root, "width", frame_meta->_width);
json_object_set_int_member(jobj_root, "height", frame_meta->_height);

// Process each object in the frame
for (auto obj_meta : frame_meta->_object_meta_list) {
    add_object_meta_to_json(jarray_objects, obj_meta);
}
```

Complete JSON Output Format:

```
{
  "streamId": 0,
  "seqId": 123,
  "width": 1920,
  "height": 1080,
  "objects": [
    {
      "object": {
        "label_id": 1,
        "track_id": 42,
        "confidence": 0.87,
```


© Copyright 2026 DEEPX All Rights Reserved.

```
      "name": "person",
      "box": {
       "startX": 300.0,
       "startY": 400.0,
       "endX": 500.0,
       "endY": 600.0
      },
      "body_feature": [0.321, 0.654, 0.987],
      "segment": {
       "height": 200,
       "width": 200,
       "format": "roi-binary-mask",
       "background_value": 0,
       "foreground_value": 255,
       "box": {
        "startX": 300.0,
        "startY": 400.0,
        "endX": 500.0,
        "endY": 600.0
       },
       "data": 140712345678912
      },
      "pose": {
       "keypoints": [
        {"kx": 100.5, "ky": 200.3, "ks": 0.8},
        {"kx": 105.2, "ky": 205.7, "ks": 0.9}
       ]
      },
      "face": {
       "landmark": [
        {"x": 150.2, "y": 180.5},
        {"x": 155.8, "y": 185.3}
       ],
       "box": {
        "startX": 100.0,
        "startY": 150.0,
        "endX": 200.0,
        "endY": 250.0
       },
       "confidence": 0.95,
       "face_feature": [0.123, 0.456, 0.789]
      }
     }
    }
  ]
}
```


© Copyright 2026 DEEPX All Rights Reserved.

JSON Structure Explanation:

- **Frame-level metadata**: `streamId`, `seqId`, `width`, `height` are extracted from `DXFrameMeta`
- **Object-level metadata**: Each `DXObjectMeta` from the frame's object list is processed to create:
- **Object Detection**: `label_id` (_label), `track_id` (_track_id), `confidence` (_confidence), `name` (_label_name), `box` (_box[4])
- **Body Features**: `body_feature` array from _body_feature vector (if available)
- **Segmentation**: `segment` object with ROI-local binary mask dimensions, mask value semantics, ROI box, and data pointer from `seg_data` (if available)
- **Pose Estimation**: `pose` object with 17 keypoints from _keypoints vector, each with kx, ky, ks values (if available)
- **Face Detection**: `face` object with landmarks from _face_landmarks, face bounding box from _face_box, confidence from _face_confidence, and face features from _face_feature (if available)

Data Type Handling:

- Coordinates and confidence values are stored as double precision floating-point
- Feature vectors are converted to JSON arrays of double values
- Integer values (dimensions, IDs) remain as integers
- Memory addresses (like segmentation data pointer) are cast to integer representation
- Object-level `segment` payloads are ROI-local binary masks aligned to `object.box`, not full-frame class maps

4.4.4 Library Integration

Build the custom message convert library with proper dependencies:

```
gst_dep = dependency('gstreamer-1.0', version : '>=1.16.3',
    required : true, fallback : ['gstreamer', 'gst_dep'])

dx_stream_dep = dependency('gstdxstream')
json_glib_dep = dependency('json-glib-1.0', version : '>= 1.0')
opencv_dep = dependency('opencv4', required: true)
dxrt_dep = declare_dependency(link_args: ['-ldxrt'])

custom_msgconv_lib = shared_library('dx_msgconvl',
    ['dx_msgconvl.cpp', 'dx_msgconvl_priv.cpp'],
    dependencies: [gst_dep, opencv_dep, json_glib_dep, dx_stream_dep, dxrt_dep],
    install: true,
```


© Copyright 2026 DEEPX All Rights Reserved.

```
    install_dir: get_option('datadir') / 'gstdxstream' / 'lib'
)
```

Required Dependencies:

- **gstreamer-1.0**: Core GStreamer framework
- **dx_stream**: DX-STREAM metadata and type definitions
- **json-glib-1.0**: JSON processing library for structured output generation
- **opencv4**: Used by the sample library for frame and mask handling
- **dxrt**: DX-RT library linked by the sample library

Usage in Pipeline:

```
dxmsgconv library-file-path=/usr/local/share/gstdxstream/lib/libdx_msgconvl.so
```

> **NOTE**
>
> The custom library does not parse configuration files. The `config-file-path` property of DxMsgConv is optional and is parsed by the element itself.

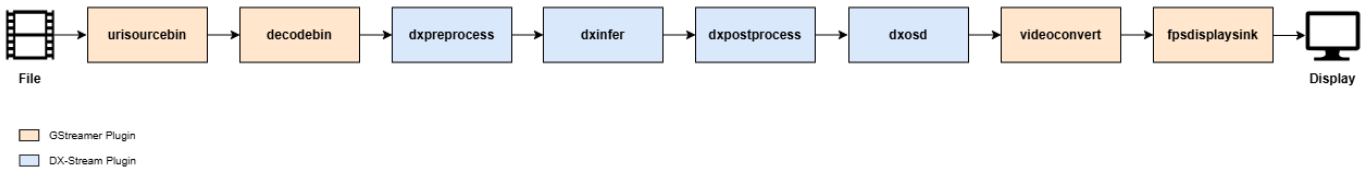
© Copyright 2026 DEEPX All Rights Reserved.

5. Pipeline Example

5.1 Single Stream Pipeline

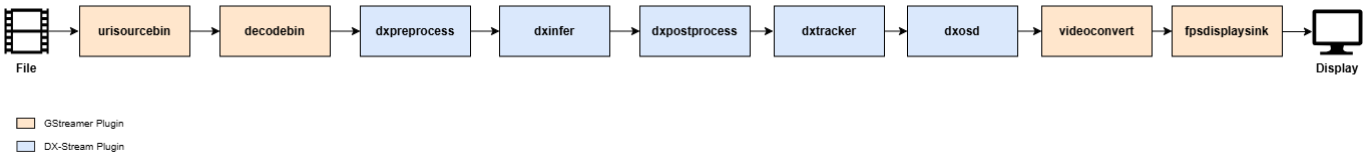
This section describes model inference for a single video stream.
Although the example pipeline performs inference using the YOLOv5s model, it can be easily adapted to various vision tasks with different models. The pipeline below reads video frames from a file and performs a pre-infer-post process tailored to YOLOv5s.

The inference results are then drawn using `dxosd`, and the final output is displayed via a display sink.



The pipeline in the figure is defined in `dx_stream/pipelines/single_network/object_detection/run_YoloV5S_PPU.sh` and can be used as a reference for execution.

When performing object detection on a single stream input, multi-object tracking can be enabled by adding dxtracker. As shown below, by inserting the tracker after the YOLO object detection, detected objects can be tracked over time.



The pipeline in the figure is defined in `dx_stream/pipelines/tracking/run_multi_object_tracker.sh` and can be used as a reference for execution.

Explanation

ⓒ Copyright 2026 DEEPX All Rights Reserved.

Element Descriptions

- **urisourcebin** : Specifies the input video file. The `uri` property **must** be set to the URI of the video you wish to process (e.g., `file:///path/to/video.mp4`).
- **decodebin** : Decodes the input video stream.
- **dxpreprocess** : Applies pre-processing according to the configuration file specified in the `config-file-path` .
- **dxinfer** : Runs inference using the YOLOv5s model. The model configuration file path is specified in `config-file-path` .
- **dxpostprocess** : Post-processes the model's output tensor to extract metadata. The configuration file path is specified in `config-file-path` .
- **dxtracker** : Tracks objects detected by the YOLO model using the OC-SORT algorithm.
- **dxosd** : Draws object detection results (e.g., bounding boxes, class labels, etc.) on the video frames.
- **fpsdisplaysink** : Displays the video frames along with the FPS (frames per second) information. The `sync=false` property ensures that all frames are displayed without being dropped.

Usage Notes

Custom Models

This pipeline is **not** limited to the YOLOv5s model. It can be easily adapted for other AI tasks by updating the corresponding model and configuration files.

- **Classification**: Image classification tasks
- **Segmentation**: Pixel-wise instance segmentation
- **Pose Estimation**: Detecting human keypoints

Update the `config-file-path` property in the `dxpreprocess` , `dxinfer` , and `dxpostprocess` elements to match your model’s configuration.

Pipeline Behavior

This pipeline runs asynchronously because a `queue max-size-buffers=1` element is placed between the processing elements. Each section runs in its own thread, so the elements can process different frames in parallel. The `queue` elements also help prevent pipeline deadlocks.

Sink Element Options

ⓒ Copyright 2026 DEEPX All Rights Reserved.

You can replace `fpsdisplaysink` with other display options.

- **`ximagesink`** : Displays video in an X11 window environment
- **`autovideosink`** : Automatically selects the most suitable video sink for the platform

Object Detection Requirement

- The dxtracker element requires bounding box information. Therefore, object detection **must** precede tracking in the pipeline.

Visualization

- The dxosd element overlays both detection and tracking results. Each object is assigned a unique track ID by the dxtracker element, which is visualized along with the bounding box.

Buffer Handling

- In this example, `fpsdisplaysink` is set with `sync=false`, which means no frames are dropped—all frames are displayed as-is.
- If you want to sync playback with the original video frame rate or reduce latency, set `sync=true`.

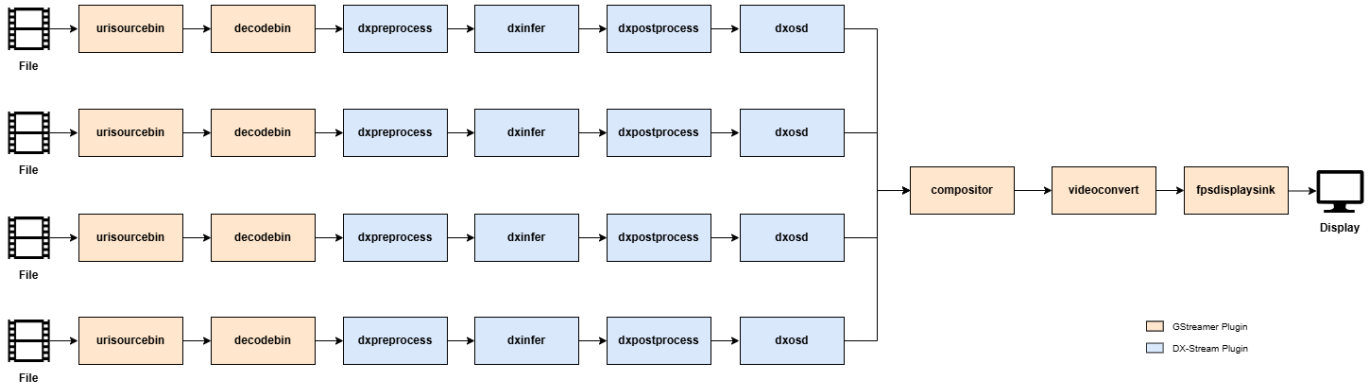
ⓒ Copyright 2026 DEEPX All Rights Reserved.

5.2 Multi Stream Pipeline

There are two possible approaches to building a pipeline structure for multi-stream inference.

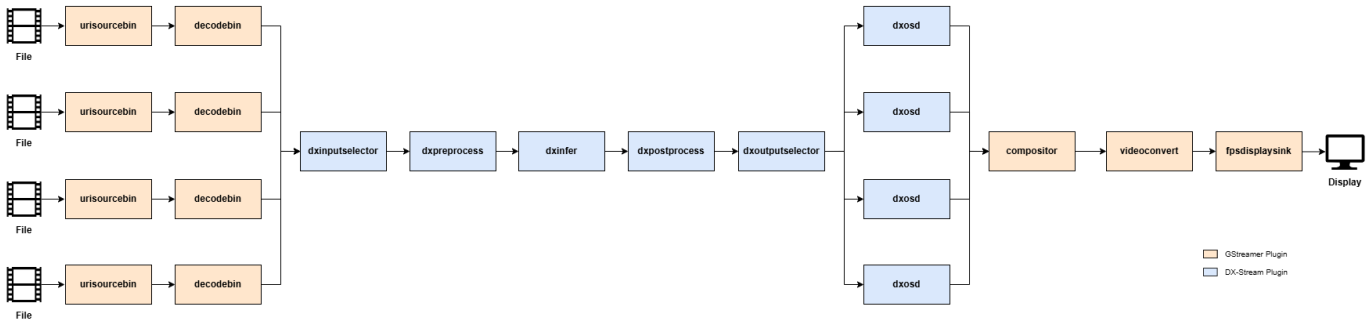
One approach is to configure an independent inference sub-pipeline for each stream, as shown in the figure below.

In this case, each sub-pipeline performs inference asynchronously, maximizing the utilization of hardware resources.



The pipeline in the figure is defined in `dx_stream/pipelines/multi_stream/run_multi_stream.sh` and can be used as a reference for execution.

The other approach uses **DxInputSelector** and **DxOutputSelector**.
DxInputSelector is an `N:1` element that receives buffers from multiple input streams and forwards the one with the smallest PTS downstream first.
The selected buffer passes through a single inference pipeline for processing, and **DxOutputSelector** then redistributes the results back to their corresponding streams.



The pipeline in the figure is defined in `dx_stream/pipelines/multi_stream/run_multi_stream_selector.sh` and can be used as a reference for execution.

ⓒ Copyright 2026 DEEPX All Rights Reserved.

Explanation

Element Descriptions

- **compositor** : Draws multiple stream buffers received through sink pads at specified positions. In the example above, it is used for tiled display of inference results from each stream.
- **dxinputselector** : Selects the buffer with the smallest PTS among multiple input streams received through sink pads and pushes it downstream.
- **dxoutputselector** : Routes buffers received from upstream back into multiple output streams.
- **dxscale** : Scales video frames to a specified resolution. Used to resize streams before display.

Usage Notes

- Depending on the model size, using a multi sub-pipeline structure may put a burden on memory resources
- While sub-pipeline structures can offer advantages in processing speed, the performance gain may be negligible compared to selector-based pipelines depending on the environment.
- **Multi-stream domain placement constraints.** The chain between `DxInputSelector` and `DxOutputSelector` uses `application/x-dxvideoraw` caps and carries per-stream identity in `DXFrameMeta`. Several elements cannot operate inside this domain and must be placed *per stream upstream of* *`DxInputSelector`* or *per output downstream of* *`DxOutputSelector`* :
  - **DxScale** and **DxConvert** — they require a single stable caps for buffer pool allocation and kernel setup. Inside the domain they would see interleaved buffers with different resolutions/formats and cannot reinitialize per buffer.
  - **DxGather** — designed to merge branches that share a single source (split by `tee`); it has no concept of `stream_id` and cannot demultiplex the merged domain stream.

  For the full element placement matrix, see Multi-Stream Domain.

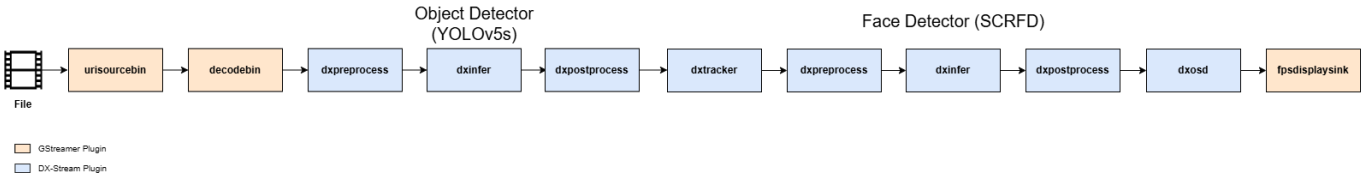
ⓒ Copyright 2026 DEEPX All Rights Reserved.

5.3 Secondary Inference Pipeline

This section describes the secondary mode inference pipeline, which performs object-level inference based on objects detected in the primary mode. A single input stream is processed through primary inference for object detection, followed by tracking.

The tracked objects are then passed to secondary inference, where additional tasks (object classification and face detection) run in two parallel branches split by `tee`. The results are stored in each object's metadata and merged by `dxgather`.

Finally, the results are rendered for visualization.



The pipeline in the figure is defined in `dx_stream/pipelines/secondary_mode/run_secondary_mode.sh` and can be used as a reference for execution.

Explanation

Pipeline Flow

- (1) **Object Detection**: Primary inference detects objects in the input stream
- (2) **Tracker**: Assigns unique IDs to detected objects and tracks them across frames
- (3) **Secondary Classification**: Classifies each tracked object with EfficientNet-Lite0 (runs in a `tee` branch, in parallel with (4))
- (4) **Secondary Face Detection**: Performs face detection with SCRFD-500M only on objects of class ID 0 (`target-class-id=0`) that are at least 50×50 pixels, every 5th frame (`interval=5`)
- (5) **Merge and Display**: `dxgather` merges the results of both branches, and `dxosd` draws them

Usage Notes

Configuration File Settings

- This pipeline runs multiple models: the primary model runs first, followed by two secondary models in parallel branches. Be careful to configure each model's preprocess_id and inference_id properly to avoid unexpected behavior.

ⓒ Copyright 2026 DEEPX All Rights Reserved.

Secondary post process

- When implementing a custom post-processing function for secondary inference mode, modify the provided DXObjectMeta directly. Removing or re-allocating the metadata may lead to unintended behavior.

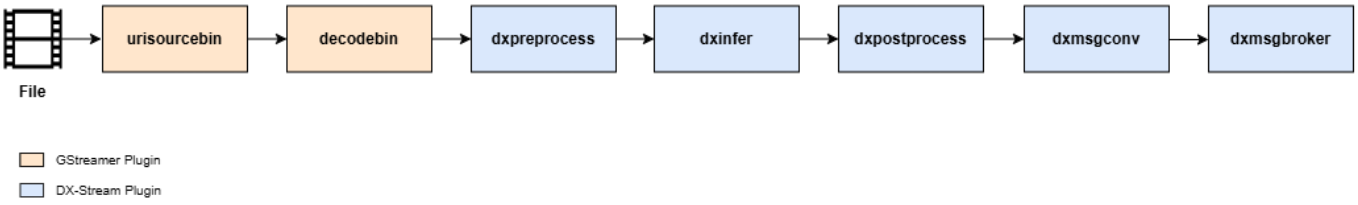
ⓒ Copyright 2026 DEEPX All Rights Reserved.

5.4 MsgBroker Pipeline

The pipeline demonstrates how to process a local video file with the YOLOv5s model for object detection, convert the results into a structured message format (e.g., JSON), and publish them to a message broker such as MQTT or Kafka for downstream applications.

5.4.1 Basic MsgBroker Pipeline Process

This section demonstrates how to run a basic message broker pipeline demo using MQTT or Kafka



The pipeline in the figure is defined in `dx_stream/pipelines/broker/*.sh` and can be used as a reference for execution.

Explanation

Element Descriptions

- **`dxmsgconv`**: Transform element that processes inference metadata from upstream **DxPostprocess** elements and converts it into structured message payloads (typically JSON) using a user-defined custom library. The element requires a `library-file-path` property pointing to the custom message conversion library.
- **`dxmsgbroker`**: Sink element that transmits payload messages to external message brokers (MQTT or Kafka) using a **Broker Abstraction Layer (BAL)** that provides unified interface for different broker types.

Usage Notes

Pipeline Execution

- **Server Side**: Run the DX-STREAM pipeline with broker output on the processing server
- **Client Side**: Run the consumer application to receive and process messages

ⓒ Copyright 2026 DEEPX All Rights Reserved.

Basic Properties

- **dxmsgconv** : Converts inference metadata to JSON format
- **dxmsgbroker** : Publishes messages to MQTT/Kafka broker

Quick Start Demo

This section demonstrates how to run a basic message broker pipeline demo using MQTT or Kafka.

MQTT DEMO

1. Server Setup (Processing Server with Message Broker)

```
# Install and start Mosquitto MQTT broker
sudo apt install mosquitto mosquitto-clients
sudo systemctl start mosquitto
sudo systemctl enable mosquitto
```

NOTE

By default, Mosquitto only accepts connections from `localhost`. To allow remote consumers, add the following to `/etc/mosquitto/mosquitto.conf`:

```
listener 1883 0.0.0.0
allow_anonymous true
```

Then restart the service: `sudo systemctl restart mosquitto`

2. Run DX-STREAM Pipeline (Server Side)

```
# Execute the broker pipeline on processing server
# The pipeline uses YOLOv5s model for object detection and converts to JSON messages
cd /path/to/dx_stream/dx_stream
./pipelines/broker/run_dxmsgbroker_mqtt.sh

# Or run manually with custom settings
gst-launch-1.0 \
  urisourcebin uri=file:///path/to/your/video.mp4 ! decodebin ! \
  dxpreprocess config-file-path=configs/YoloV5S_PPU/preprocess_config.json ! \
  dxinfer config-file-path=configs/YoloV5S_PPU/inference_config.json ! \
  dxpostprocess config-file-path=configs/YoloV5S_PPU/postprocess_config.json ! \
  dxmsgconv library-file-path=/usr/local/share/gstdxstream/lib/libdx_msgconvl.so config-file-path=co
```


© Copyright 2026 DEEPX All Rights Reserved.

```
nfigs/msgconv_config.json ! \
  dxmsgbroker broker-name=mqtt conn-info=localhost:1883 topic=test
```

3. Client Application (Any Machine - Consumer Only)

```
# Install Python MQTT client library (client machine only needs this)
pip install paho-mqtt

# Run the MQTT client to receive messages from server
python3 /usr/local/share/gstdxstream/bin/mqtt_sub_example.py -n <server_ip> -p 1883 -t test

# Or for C++ version
mqtt_sub_example -n <server_ip> -t test -p 1883
```

KAFKA DEMO

1. Server Setup (Processing Server with Message Broker)

This example uses Kafka **4.3.1** in **KRaft mode** for a single-node demo. Kafka **4.0+** no longer supports ZooKeeper and requires **Java 17 or later** to run the broker.

```
# Install Java 17 explicitly (default-jdk may install Java 11)
sudo apt update
sudo apt install openjdk-17-jdk
java -version

# Download and extract Kafka (skip if already installed)
wget https://downloads.apache.org/kafka/4.3.1/kafka_2.13-4.3.1.tgz
tar -xzf kafka_2.13-4.3.1.tgz
cd kafka_2.13-4.3.1

# Initialize KRaft storage (first time only)
KAFKA_CLUSTER_ID=$(bin/kafka-storage.sh random-uuid)
bin/kafka-storage.sh format -t "$KAFKA_CLUSTER_ID" -c config/server.properties --standalone

# Start the broker in the foreground; use another terminal for the pipeline
bin/kafka-server-start.sh config/server.properties
```

Java version

Confirm that `java -version` reports Java **17 or later** before running Kafka. If it still reports Java 11, select Java 17 with `sudo update-alternatives --config java`. If `JAVA_HOME` is set, it must also point to a Java 17+ installation, because Kafka uses it instead of the default `java`.

© Copyright 2026 DEEPX All Rights Reserved.

Initialize storage only once

Run all Kafka commands from the extracted Kafka root directory, not its `bin/` directory. If storage is already initialized, skip UUID generation and formatting; restart using only `bin/kafka-server-start.sh config/server.properties`. Do not delete existing data or reformat storage to resolve an initialization error. The default `log.dirs` is `/tmp/kraft-combined-logs`; for persistent use, change it to a durable directory before the first format.

After the broker starts, verify connectivity from another terminal in the Kafka root directory:

```
bin/kafka-topics.sh --bootstrap-server localhost:9092 --list
```

An empty list is normal if no topics have been created yet.

NOTE

If the consumer runs on a different machine from the broker, set `advertised.listeners` in `config/server.properties` to the broker's IP:

```
advertised.listeners=PLAINTEXT://<broker_ip>:9092
```

Then restart the Kafka server. Without this, remote consumers will fail to resolve the broker's internal hostname.

2. Run DX-STREAM Pipeline (Server Side)

```
# Execute the broker pipeline on processing server
cd /path/to/dx_stream/dx_stream
./pipelines/broker/run_dxmsgbroker_kafka.sh

# Or run manually with custom settings
gst-launch-1.0 \
  urisourcebin uri=file:///path/to/your/video.mp4 ! decodebin ! \
  dxpreprocess config-file-path=configs/YoloV5S_PPU/preprocess_config.json ! \
  dxinfer config-file-path=configs/YoloV5S_PPU/inference_config.json ! \
  dxpostprocess config-file-path=configs/YoloV5S_PPU/postprocess_config.json ! \
  dxmsgconv library-file-path=/usr/local/share/gstdxstream/lib/libdx_msgconvl.so config-file-path=co
nfigs/msgconv_config.json ! \
  dxmsgbroker broker-name=kafka conn-info=localhost:9092 topic=test
```


© Copyright 2026 DEEPX All Rights Reserved.

3. Client Application (Any Machine - Consumer Only)

```
# Install Python Kafka client library (client machine only needs this)
python3 -m pip install --upgrade pip
python3 -m pip install confluent-kafka

# Run the Kafka client to receive messages from server
python3 /usr/local/share/gstdxstream/bin/kafka_consume_example.py -n <server_ip> -p 9092 -t tes
t

# Or for C++ version
kafka_consume_example -n <server_ip> -p 9092 -t test
```

WINDOWS

On Windows, pre-built batch scripts are provided at `dx_stream/pipelines/windows/` :

```
broker_mqtt.bat    :: MQTT pipeline (localhost:1883)
broker_kafka.bat   :: Kafka pipeline (localhost:9092)
```

Scripts auto-detect `DXSTREAM_ROOT` from their location — no environment variable setup needed.

> **Broker server required**
>
> `install.bat` installs only the **client libraries** (libmosquitto, librdkafka) needed to build DX-Stream. A separate **broker server** must be running before executing the pipeline.

MQTT (Mosquitto)

Download the Windows installer from mosquitto.org/download and install. Then start the broker:

```
mosquitto -v
```

Or start as a Windows service via Services panel (`services.msc`).

Kafka

© Copyright 2026 DEEPX All Rights Reserved.

Kafka 4.0+ uses **KRaft mode** (ZooKeeper is no longer supported) and requires **Java 17 or later** to run the broker.

- (1) Download JDK **17 or later** from adoptium.net (MSI installer recommended) and install. Confirm with `java -version`; if `JAVA_HOME` is set, it must point to the same supported JDK.
- (2) Download Kafka binary from kafka.apache.org/downloads and extract to a **short path** (e.g., `C:\kafka`).

> ⚠ **Use a short path for Kafka**
>
> Extracting Kafka under a long path (e.g., inside a user's git repo folder) causes ("The input line is too long") errors in the batch scripts.
>
> Always extract to a short path like `C:\kafka`.

- (3) Initialize storage once, then start the broker from `C:\kafka`. On subsequent starts, run only the broker start command:

```
cd C:\kafka

:: Generate a cluster UUID
bin\windows\kafka-storage.bat random-uuid
:: → copy the printed UUID, e.g. KxWspAMDQNmidYkTX2KZcw

:: Format storage (first time only)
bin\windows\kafka-storage.bat format -t <UUID> -c config\server.properties --standalone

:: Start broker
bin\windows\kafka-server-start.bat config\server.properties
```

Running the pipelines

```
:: from dx_stream repo root
.\dx_stream\pipelines\windows\broker_mqtt.bat
.\dx_stream\pipelines\windows\broker_kafka.bat
```

Custom video (optional):

```
.\dx_stream\pipelines\windows\broker_mqtt.bat "C:\path\to\video.mp4"
```


© Copyright 2026 DEEPX All Rights Reserved.

Consumer apps are bundled at `install\share\gstdxstream\bin\` :

- `mqtt_sub_example.exe` / `mqtt_sub_example.py`
- `kafka_consume_example.exe` / `kafka_consume_example.py`

> ⚠ **Kafka: start consumer before pipeline**
>
> The C++ consumer (`kafka_consume_example.exe`) starts reading from the latest offset, so it does not receive messages published before it connected. The Python consumer reads from the beginning. Start the consumer first, then run the pipeline:
>
> ```
> :: Terminal 1 — consumer first
> install\share\gstdxstream\bin\kafka_consume_example.exe -n localhost -p 9092 -t test
>
> :: Terminal 2 — then pipeline
> .\dx_stream\pipelines\windows\broker_kafka.bat
> ```
>
> `Unknown topic or partition` on first launch is normal — the topic is created automatically once the pipeline runs.
>
> MQTT has no such constraint — consumer can connect before or after the pipeline.

MQTT consumer:

```
install\share\gstdxstream\bin\mqtt_sub_example.exe -n localhost -p 1883 -t test
:: or Python
python install\share\gstdxstream\bin\mqtt_sub_example.py -n localhost -p 1883 -t test
```

Kafka consumer:

```
install\share\gstdxstream\bin\kafka_consume_example.exe -n localhost -p 9092 -t test
:: or Python
python install\share\gstdxstream\bin\kafka_consume_example.py -n localhost -p 9092 -t test
```

NETWORK ARCHITECTURE

Processing Server:

- DX-STREAM Pipeline (dxmsgconv + dxmsgbroker)
- Message Broker Service (MQTT/Kafka)

© Copyright 2026 DEEPX All Rights Reserved.

Client Machine:

- Consumer Application (Python/C++)
- Only needs client library (paho-mqtt/confluent-kafka)
- No broker service needed

Connection:

- Processing Server ←→ Client Machine via network

Requirements:

- **Server**: DX-STREAM + Message Broker Service (MQTT/Kafka)
- **Client**: Only consumer application + client library (paho-mqtt/confluent-kafka)

> **NOTE**
>
> Replace `<server_ip>` with the actual IP address of your processing server. For local testing, use `localhost`.

PIPELINE PROPERTIES

The broker pipelines use the following key properties:

DxMsgConv Element:

- `config-file-path`: Path to the JSON configuration file for the element's properties (`library-file-path`, `message-interval`, `include-frame`) (optional)
- `library-file-path`: Path to custom message converter library (**required**)
- `message-interval`: Frame interval for message conversion (default: 1)

DxMsgBroker Element:

- `broker-name`: Message broker type - "mqtt" or "kafka" (**required**)
- `conn-info`: Connection string in format `host:port` (**required**)
- `topic`: Topic name for message publishing (**required**)
- `config`: Path to broker configuration file for advanced settings (optional)

© Copyright 2026 DEEPX All Rights Reserved.

5.4.2 Advanced MsgBroker Configuration

This section describes the detailed steps required to set up secure communication for the MQTT and Kafka message brokers, including SSL/TLS encryption and client authentication

MQTT Security Configuration

SSL/TLS ENCRYPTION SETUP

1. Generate CA Certificate and Server Keys

```
# Create Root CA
openssl genrsa -out ca.key 2048
openssl req -new -x509 -days 360 -key ca.key -out ca.crt -subj "/C=KR/ST=KK/L=SN/O=DXS/
OU=Test/CN=TestCA"

# Create Server Certificate (CN must match hostname)
openssl genrsa -out server.key 2048
openssl req -new -out server.csr -key server.key -subj "/C=KR/ST=KK/L=SN/O=DXS/OU=Server/
CN=DXS-BROKER"
openssl x509 -req -in server.csr -CA ca.crt -CAkey ca.key -CAcreateserial -out server.crt -days 360
openssl verify -CAfile ca.crt server.crt

# Create Client Certificate (for client authentication)
openssl genrsa -out client.key 2048
openssl req -new -out client.csr -key client.key -subj "/C=KR/ST=KK/L=SN/O=DXS/OU=Client/
CN=DXS-CLIENT"
openssl x509 -req -in client.csr -CA ca.crt -CAkey ca.key -CAcreateserial -out client.crt -days 360
openssl verify -CAfile ca.crt client.crt

# Move certificates to mosquitto directories
sudo mkdir -p /etc/mosquitto/ca_certificates /etc/mosquitto/certs
sudo mv ca.crt /etc/mosquitto/ca_certificates/
sudo mv ca.key /etc/mosquitto/ca_certificates/
sudo mv server.crt server.key /etc/mosquitto/certs/
sudo mv client.crt client.key /etc/mosquitto/certs/

# Set correct ownership and permissions
sudo chown mosquitto:mosquitto /etc/mosquitto/ca_certificates/ca.crt
sudo chown mosquitto:mosquitto /etc/mosquitto/certs/server.crt /etc/mosquitto/certs/server.key
sudo chmod 644 /etc/mosquitto/ca_certificates/ca.crt
sudo chmod 600 /etc/mosquitto/ca_certificates/ca.key
sudo chmod 644 /etc/mosquitto/certs/server.crt
sudo chmod 600 /etc/mosquitto/certs/server.key
sudo chmod 644 /etc/mosquitto/certs/client.crt
sudo chmod 600 /etc/mosquitto/certs/client.key
```

2. Configure Mosquitto Server

ⓒ Copyright 2026 DEEPX All Rights Reserved.

```
# Edit /etc/mosquitto/mosquitto.conf
sudo tee /etc/mosquitto/mosquitto.conf << 'EOF'
# Basic settings
persistence true
persistence_location /var/lib/mosquitto/
log_dest file /var/log/mosquitto/mosquitto.log
include_dir /etc/mosquitto/conf.d

# Global auth settings (apply to all listeners)
allow_anonymous false
password_file /etc/mosquitto/passwd

# Plain MQTT port
listener 1883

# TLS MQTT port
listener 8883
cafile /etc/mosquitto/ca_certificates/ca.crt
certfile /etc/mosquitto/certs/server.crt
keyfile /etc/mosquitto/certs/server.key
require_certificate false
use_identity_as_username false
EOF
```

3. Setup Authentication

```
# Create user credentials
sudo mosquitto_passwd -b /etc/mosquitto/passwd user 1234
sudo chmod 644 /etc/mosquitto/passwd
sudo chown root:mosquitto /etc/mosquitto/passwd

# Restart mosquitto
sudo systemctl restart mosquitto
sudo systemctl status mosquitto
```

4. Test Connection

```
# Test plain MQTT (port 1883)
mosquitto_sub -h localhost -p 1883 -t test -u user -P 1234

# Test TLS MQTT (port 8883) - in another terminal
# For self-signed certificates, use --insecure option
mosquitto_sub -h localhost -p 8883 -t test -u user -P 1234 --cafile /etc/mosquitto/ca_certificates/
ca.crt --insecure

# Publish test message
mosquitto_pub -h localhost -p 8883 -t test -u user -P 1234 --cafile /etc/mosquitto/ca_certificates/
ca.crt --insecure -m "TLS test message"
```


ⓒ Copyright 2026 DEEPX All Rights Reserved.

5. DxMsgBroker Configuration File

Create `broker_mqtt.cfg` :

```
# Authentication
username = user
password = 1234

# Client ID (optional)
client-id = dx_stream_client

# SSL/TLS Configuration
tls_enable = 1
tls_insecure = 1
tls_cafile = /etc/mosquitto/ca_certificates/ca.crt
tls_capath = /etc/mosquitto/ca_certificates
tls_certfile = /etc/mosquitto/certs/client.crt
tls_keyfile = /etc/mosquitto/certs/client.key
```

6. Modify DX-Stream Pipeline

```
   gst-launch-1.0 -e urisourcebin uri=file://$INPUT_VIDEO_PATH ! decodebin ! \
                dxpreprocess config-file-path=$SRC_DIR/configs/YoloV5S_PPU/
preprocess_config.json ! queue ! \
                dxinfer config-file-path=$SRC_DIR/configs/YoloV5S_PPU/inference_config.json !
queue ! \
                dxpostprocess config-file-path=$SRC_DIR/configs/YoloV5S_PPU/
postprocess_config.json ! queue ! \
                dxmsgconv config-file-path=$SRC_DIR/configs/msgconv_config.json ! queue ! \
                dxmsgbroker broker-name=mqtt conn-info=localhost:8883 topic=test
config=$SRC_DIR/configs/broker_mqtt.cfg
```

Kafka Security Configuration

SSL/TLS SETUP FOR KAFKA

1. Generate Certificates for Kafka

Follow the librdkafka SSL guide for detailed certificate generation.

2. Configure Kafka Server (server.properties)

```
# SSL Configuration
ssl.protocol=TLS
ssl.enabled.protocols=TLSv1.2,TLSv1.3
ssl.keystore.type=JKS
```


ⓒ Copyright 2026 DEEPX All Rights Reserved.

```
ssl.keystore.location=/path/to/kafka.server.keystore.jks
ssl.keystore.password=server_keystore_password
ssl.key.password=server_key_password
ssl.truststore.type=JKS
ssl.truststore.location=/path/to/kafka.server.truststore.jks
ssl.truststore.password=server_truststore_password
ssl.client.auth=required
```

3. DxMsgBroker Kafka Configuration

Create `broker_kafka.cfg` :

```
[kafka]
# Message size limits
message.max.bytes=10485760

# SSL Configuration
security.protocol=ssl
ssl.ca.location=/path/to/ca-cert.pem
ssl.certificate.location=/path/to/client-cert.pem
ssl.key.location=/path/to/client-key.pem
ssl.key.password=client_key_password
```


ⓒ Copyright 2026 DEEPX All Rights Reserved.

6. Troubleshooting and FAQ

6.1 Debug Logging

Overview

DX-STREAM provides comprehensive debug logging through GStreamer's `GST_DEBUG` system. This allows you to monitor element behavior, trace buffer flow, analyze performance, and diagnose issues at various levels of detail.

Basic Usage

Enable debug logging for all DX-STREAM elements:

```
$ GST_DEBUG=dx*:4 ./your_application
```

Enable debug logging for specific elements:

```
$ GST_DEBUG=dxinfer:4,dxtracker:4 ./your_application
```

Debug Levels

Level	Name	When to Use
0	NONE	No debug output
1	ERROR	Critical failures preventing operation
2	WARNING	Non-fatal issues that may affect behavior
3	FIXME	Fixme messages; rarely used in DX-STREAM
4	INFO	State changes and configuration confirmation
5	DEBUG	

© Copyright 2026 DEEPX All Rights Reserved.

Level	Name	When to Use
		Detailed operational information for troubleshooting
6	LOG	Per-buffer logging
7	TRACE	Internal algorithm details

Common Scenarios

Monitor inference performance:

```
$ GST_DEBUG=dxinfer:5 ./your_app 2>&1 | grep "Inference throughput"
```

Debug configuration loading:

```
$ GST_DEBUG=dx*:4 ./your_app 2>&1 | grep -i "loaded"
```

Track buffer flow:

```
$ GST_DEBUG=dx*:6 ./your_app 2>&1 | grep "Processing buffer"
```

Identify dropped frames:

```
$ GST_DEBUG=dxpreprocess:6,dxinfer:6 ./your_app 2>&1 | grep "Dropping"
```

Save logs to file:

```
$ GST_DEBUG=dx*:4 GST_DEBUG_FILE=/tmp/debug.log ./your_app
```

Detailed Guide

For comprehensive debugging strategies and element-specific logging information, see **Chapter 07 - Debugging Guide**.

6.2 Rendering Issues

Problem: Abnormal Behavior

© Copyright 2026 DEEPX All Rights Reserved.

The pipeline may exhibit abnormal behavior or fail to display video when attempting to render the video stream on the screen.

Cause: Unsupported Element

The root cause of abnormal behavior is:

- Incompatibility: The specified displaysink element is not supported by the current PC environment (e.g., due to unsupported graphics drivers or display server settings).

Solution: Element Selection & Compatibility

It's essential to address both the hardware fit and the data format compatibility.

Solution 1: Element Selection

- For CPU-based Environments: Use ximagesink or xvimagesink.
- For GPU-based Environments: Use glimagesink or similar GPU-accelerated elements.

Solution 2: Ensure Compatibility

Mandatory Step: Always add videoconvert directly before the displaysink element.

- Purpose: This element correctly converts the video format from upstream elements into a format the chosen sink can process, preventing format mismatch errors.

```
$ gst-launch-1.0 .... ! videoconvert ! autovideosink
```

6.3 Display Sink Issues on Raspberry Pi 5

```
(ERROR: from element /GstPipeline:pipeline0/GstFPSDisplaySink:fpsdisplaysink0/
GstAutoVideoSink:fps-display-video_sink/GstKMSSink:fps-display-video_sink-actual-sink-kms:
GStreamer encountered a general resource error.
Additional debug info:
../sys/kms/gstkmssink.c(2032): gst_kms_sink_show_frame (): /GstPipeline:pipeline0/
GstFPSDisplaySink:fpsdisplaysink0/GstAutoVideoSink:fps-display-video_sink/GstKMSSink:fps-display-
video_sink-actual-sink-kms:
drmModeSetPlane failed: Permission denied (13)
ERROR: pipeline doesn't want to preroll.
Setting pipeline to NULL ...)
```


ⓒ Copyright 2026 DEEPX All Rights Reserved.

Problem: Rendering Failure

The GStreamer pipeline fails to preroll and terminates prematurely, resulting in a general resource error. The specific error message indicates a system access issue: drmModeSetPlane failed: Permission denied (13).

Cause: High-Ranked KMSSink Failure

The issue stems from the automatic selection process of the sink element:

Automatic Selection: The fpsdisplaysink element automatically selects the rendering sink with the highest RANK.

The Culprit: In this environment, the high-ranked kmssink was automatically chosen.

Environmental Failure: Due to environmental issues, likely related to permissions or system configuration specific to the Raspberry Pi 5 setup, the selected kmssink failed to operate normally, leading to the "Permission denied" error.

Solution: Manual Sink Replacement

The solution involves overriding the automatic, failing selection by manually specifying a stable, compatible sink element.

Action: Modify the pipeline code to **replace the fpsdisplaysink element with ximagesink**.

Result: **ximagesink uses CPU-based rendering** within an X11 environment, circumventing the resource and permission issues associated with kmssink to ensure proper display output.

6.4 Display Sink Issues on Orange Pi 5 Plus with Debian 12

Problem: Corrupted or Distorted Video Rendering

When running DX-STREAM pipeline scripts on **Orange Pi 5 Plus with Debian 12**, you may experience corrupted, distorted, or improperly rendered video output. The video display may appear garbled, with incorrect colors, flickering, or visual artifacts that make the output unusable.

ⓒ Copyright 2026 DEEPX All Rights Reserved.

Affected Environment:

- **Hardware**: Orange Pi 5 Plus (RK3588 chipset)
- **Operating System**: Debian 12 (official image from Orange Pi download page)
- **Affected Component**: Video rendering pipeline (specifically the fpsdisplaysink element)

Cause: GStreamer Format Negotiation Bug

This is a **GStreamer bug specific to the Orange Pi 5 Plus + Debian 12 environment**. The issue occurs during format negotiation between the `videoconvert` element and the `fpsdisplaysink` element:

(1) **Format Mismatch**: The default video format negotiated by `videoconvert` is incompatible with the sink element selected by `fpsdisplaysink` in this environment.

(2) **Environment-Specific**: This bug was observed with the official Debian 12 image for Orange Pi 5 Plus. It may or may not occur depending on:

- The specific Debian 12 ISO image version used
- System library versions (GStreamer, graphics drivers, etc.)
- Display server configuration (X11/Wayland)

(3) **Not Universal**: While primarily seen on Orange Pi 5 Plus + Debian 12, **similar rendering issues could potentially occur on other ARM-based SBCs or platforms** with comparable GStreamer + display stack configurations.

Solution: Force I420 Format Conversion

The workaround is to explicitly force the video format to **I420** before passing it to `fpsdisplaysink`, which bypasses the faulty format negotiation.

Action: In the affected pipeline script, uncomment the Orange Pi 5 Plus workaround code block.

Before (default - may cause rendering issues):

```
# Default videoconvert pipeline
VIDEOCONVERT_PIPELINE="videoconvert"

# NOTE: If you experience rendering issues (corrupted/distorted video output) on Orange Pi 5 Plus
# with Debian 12, uncomment the lines below to force I420 format conversion.
# See troubleshooting documentation for more details.
# if grep -q "rk3588" /proc/device-tree/compatible 2>/dev/null; then
```


ⓒ Copyright 2026 DEEPX All Rights Reserved.

```
#     if [ "$(lsb_release -rs)" = "12" ]; then
#         echo "Detected Orange Pi 5 Plus with Debian 12 - using I420 format"
#         VIDEOCONVERT_PIPELINE="videoconvert ! video/x-raw,format=I420"
#     fi
# fi
```

After (uncommented - fixes rendering issues):

```
# Default videoconvert pipeline
VIDEOCONVERT_PIPELINE="videoconvert"

# NOTE: If you experience rendering issues (corrupted/distorted video output) on Orange Pi 5 Plus
# with Debian 12, uncomment the lines below to force I420 format conversion.
# See troubleshooting documentation for more details.
if grep -q "rk3588" /proc/device-tree/compatible 2>/dev/null; then
    if [ "$(lsb_release -rs)" = "12" ]; then
        echo "Detected Orange Pi 5 Plus with Debian 12 - using I420 format"
        VIDEOCONVERT_PIPELINE="videoconvert ! video/x-raw,format=I420"
    fi
fi
```

Location: This code block appears in all DX-STREAM pipeline scripts that use video display, including:

- `dx_stream/pipelines/single_network/*/run_*.sh`
- `dx_stream/pipelines/multi_stream/run_*.sh`
- `dx_stream/pipelines/rtsp/run_*.sh`
- `dx_stream/pipelines/tracking/run_*.sh`
- `dx_stream/pipelines/secondary_mode/run_*.sh`

Result: Forcing the `I420` format ensures proper format negotiation and eliminates the rendering corruption, allowing normal video display.

ⓒ Copyright 2026 DEEPX All Rights Reserved.

When to Apply This Workaround

- **Always apply** if you're using Orange Pi 5 Plus with official Debian 12 image and experiencing rendering issues
- **Try first without workaround** on other Debian versions or custom images - it may work fine
- **Consider applying** if you experience similar rendering issues on other ARM SBCs with similar software stacks
- **No harm in enabling** - forcing I420 format is a safe operation that only affects format negotiation

6.5 Buffer Delays in Sink Element

Problem & Symptoms

The core problem is a performance bottleneck in the system, leading to noticeable playback degradation and warning messages.

- Stuttering or lagging video playback.
- Pipeline performance degradation.
- Warning messages in the console, such as "buffering too slow" or "dropped frames."

Solutions: Performance Optimization

Solutions focus on optimizing both the PC environment and the GStreamer pipeline structure.

Solution 1: Optimize PC Performance

- **Terminate Background Processes**: Free up CPU/GPU resources by closing any unnecessary programs running in the background.
- **Use Lower-Resolution Videos**: Reduce the decoding and rendering workload by using lower input video resolutions or downscaling the video stream early in the pipeline.

© Copyright 2026 DEEPX All Rights Reserved.

Solution 2: Optimize the GStreamer Pipeline

- **1. Add queue Elements (Decoupling):**

  Insert queue elements at potential bottleneck points to decouple (separate) processing speeds between adjacent elements. This allows faster elements to process data ahead, mitigating delays caused by slower elements.

  Example:

```
gst-launch-1.0 filesrc location=video.mp4 ! decodebin ! queue ! autovideosink
```

- **2. Use Asynchronous Rendering (Disable Synchronization):**

  Disable real-time playback synchronization by setting sync=false on the sink element. This tells the sink to render frames as quickly as possible, ignoring the clock and often reducing perceived lag, though it may result in playback that is faster or slower than real-time.

  Example:

```
gst-launch-1.0 ... autovideosink sync=false
```

6.6 Troubleshooting Message Broker Issues

A. Common MQTT Problems

Connection Refused

- Check Broker Status

```
# Verify if Mosquitto is running
sudo systemctl status mosquitto
```

- Test Basic Connection

```
# Attempt a basic publish operation
mosquitto_pub -h localhost -p 1883 -t test -m "hello"
```

- Remote Connection Refused

By default, Mosquitto only accepts connections from `localhost`. To allow remote access, add the following to `/etc/mosquitto/mosquitto.conf`:

© Copyright 2026 DEEPX All Rights Reserved.

```
listener 1883 0.0.0.0
allow_anonymous true
```

Then restart the service: `sudo systemctl restart mosquitto`

SSL Certificate Issues

- Verify Chain

```
# Verify the certificate chain integrity
openssl verify -CAfile ca.crt server.crt
```

- Test SSL Connection

```
# Test secure publishing
mosquitto_pub -h localhost -p 8883 --cafile ca.crt -t test -m "ssl_test"
```

B. Common Kafka Problems

Consumer Lag

- Check Consumer Group Status

```
# monitor the consumer group status to identify lag
bin/kafka-consumer-groups.sh --bootstrap-server localhost:9092 --group my-group --describe
```

SSL Handshake Failures - Test SSL Connection

```
# Test SSL connection
bin/kafka-console-producer.sh --bootstrap-server localhost:9093 --topic test \
  --producer.config ssl.properties
```

C. DX-STREAM Broker Element Issues

Library Loading Errors

- Ensure message converter library path is correct
- Verify library dependencies (json-glib-1.0, etc.) are installed and linked.

ⓒ Copyright 2026 DEEPX All Rights Reserved.

Message Format Issues

- Check if custom library implements all required functions
- Verify that the final JSON output format matches expected structure

Performance Issues

- Adjust the `message-interval` property on **DxMsgConv** to reduce message frequency
- Monitor the broker server resources (CPU, memory, network)

D. Critical Kafka Connection Refusal Error

A common error when using Kafka is connection refusal, as indicated by the following log.

```
%3|1736310124.667|FAIL|rdkafka#producer-1| [thrd:localhost:9092/bootstrap]: localhost:9092/
bootstrap: Connect to ipv4#127.0.0.1:9092 failed: Connection refused (after 0ms in state CONNECT)
ERROR: Pipeline doesn't want to pause.
ERROR: from element /GstPipeline:pipeline0/GstDxMsgBroker:dxmsgbroker0: GStreamer error: state
change failed and some element failed to post a proper error message with the reason for the failure.
```

This error usually means the Kafka broker is not running. To resolve this, verify its status and restart it if necessary.

ⓒ Copyright 2026 DEEPX All Rights Reserved.

Solution Steps (If Broker is Not Running)

- Check Running Status

```
$ ps -ef | grep kafka
```

- Installation and Startup (If not running)

  Installation: Create a utilities directory and install **Java 17 or later** (required by Kafka 4.0+). Download and extract the Kafka distribution:

```
$ mkdir utils && cd utils
$ sudo apt update
$ sudo apt install openjdk-17-jdk
$ wget https://downloads.apache.org/kafka/4.3.1/kafka_2.13-4.3.1.tgz
$ tar -xzf kafka_2.13-4.3.1.tgz
$ cd kafka_2.13-4.3.1
```

  Initialize KRaft storage (first time only): Kafka 4.0+ no longer uses Zookeeper

```
$ KAFKA_CLUSTER_ID=$(bin/kafka-storage.sh random-uuid)
$ bin/kafka-storage.sh format -t "$KAFKA_CLUSTER_ID" -c config/server.properties --standalone
```

  Start the Kafka broker (in the foreground; keep this terminal open and use another terminal for the pipeline)

```
$ bin/kafka-server-start.sh config/server.properties
```

NOTE

Keep the broker terminal running while the DX-STREAM pipeline is active. If storage is already initialized, skip the format step and run only `bin/kafka-server-start.sh config/server.properties`.

ⓒ Copyright 2026 DEEPX All Rights Reserved.

Remote Access

If the consumer runs on a different machine from the broker, set `advertised.listeners` in `config/server.properties` to the broker's IP:

```
advertised.listeners=PLAINTEXT://<broker_ip>:9092
```

Then restart the Kafka server. Without this, remote consumers will fail to resolve the broker's internal hostname.

6.7 Build Issues with Meson Installation

Overview: How `build.sh` Handles `meson install`

DX-STREAM uses `sudo meson install` to install plugins into system directories (e.g., `/usr/local/`).
Since `sudo` runs as root and does not inherit the user's Python environment, `build.sh` explicitly passes the Python package paths via `PYTHONPATH`:

```
sudo env PYTHONPATH="$(python3 -c '...')" "$(which meson)" install -C "${BUILD_DIR}" --no-rebuild
```

This command resolves two things from the **current shell** before `sudo` execution:

- `$(which meson)` — locates the meson binary
- `$(python3 -c '...')` — collects all Python site-packages paths

Supported Meson Installation Methods

The following single-meson-installation environments are fully supported:

Installation Method	Binary Location	Module Location	Supported
`pip install --user meson`	`~/.local/bin/meson`	`~/.local/lib/python3.x/site-packages/`	☑
`pip install meson` (in venv)	`venv/bin/meson`	`venv/lib/python3.x/site-packages/`	☑

© Copyright 2026 DEEPX All Rights Reserved.

Installation Method	Binary Location	Module Location	Supported
`sudo pip install meson`	`/usr/local/bin/meson`	`/usr/local/lib/python3.x/dist-packages/`	☑
`apt install meson`	`/usr/bin/meson`	`/usr/lib/python3/dist-packages/`	☑

Known Limitation: Multiple Meson Installations

> **WARNING**
>
> If meson is installed in **more than one location** simultaneously, a version mismatch may occur between the meson binary and its Python module.

When multiple meson installations coexist, the `meson` binary resolved by `which meson` and the `mesonbuild` Python module loaded at runtime may come from **different installations**:

```
# Example: meson binary from venv (v1.4), but module from ~/.local/ (v0.61)
$ which meson
/home/user/venv/bin/meson          # meson 1.4

$ python3 -c "import mesonbuild; print(mesonbuild.__file__)"
/home/user/.local/.../mesonbuild/    # meson 0.61 (loaded first by import order)
```

This is a **fundamental limitation** of the `sudo + PYTHONPATH` pattern and cannot be fully resolved by any PYTHONPATH construction strategy.

Symptoms:

- `meson install` fails with unexpected errors or tracebacks
- Build succeeds but installed files are incorrect or incomplete
- `ModuleNotFoundError: No module named 'mesonbuild'`

Solution: Keep a Single Meson Installation

Step 1. Check for multiple installations:

© Copyright 2026 DEEPX All Rights Reserved.

```
# List all meson binaries in PATH
which -a meson

# Check pip-installed meson
pip show meson 2>/dev/null && echo "Found: pip (user/venv)"
pip3 show meson 2>/dev/null && echo "Found: pip3"

# Check system package
apt list --installed 2>/dev/null | grep meson

# Verify binary and module point to the same installation
which meson
python3 -c "import mesonbuild, os; print(os.path.dirname(os.path.dirname(mesonbuild.__file__)))"
```

Step 2. Remove duplicates, keeping only the intended one:

```
# Remove user-level installation
pip uninstall meson

# Or remove system-level installation
sudo apt remove meson

# Or remove from venv
pip uninstall meson  # with venv activated
```

Step 3. Verify single installation:

```
$ which meson
/home/user/.local/bin/meson

$ python3 -c "import mesonbuild, os; print(os.path.dirname(os.path.dirname(mesonbuild.__file__)))"
/home/user/.local/lib/python3.8/site-packages

# Both should share the same installation prefix (~/.local/ in this example)
```

6.8 Multi-Stream Domain Issues

Problem: `could not link element` between `DxInputSelector` and a downstream element

A pipeline like this fails to start:

```
gst-launch-1.0 ... ! dxinputselector ! videoconvert ! ...
WARNING: erroneous pipeline: could not link dxinputselector0 to videoconvert0
```


© Copyright 2026 DEEPX All Rights Reserved.

Cause: Domain caps mismatch

`DxInputSelector` outputs `application/x-dxvideoraw`, not `video/x-raw`. Standard GStreamer elements (`videoconvert`, `videoscale`, `compositor`, `videorate`, …) only accept `video/x-raw`, so caps negotiation fails immediately at pipeline construction.

Solution

Either place the standard element **upstream of `DxInputSelector`** (per stream, while caps are still `video/x-raw`), or use a DX-STREAM element that accepts `application/x-dxvideoraw` inside the domain (`DxPreprocess`, `DxInfer`, `DxPostprocess`, `DxTracker`, `DxOsd`, `DxRate`, `DxMsgConv`, `DxMsgBroker`). After `DxOutputSelector`, caps revert to `video/x-raw` and standard elements work again.

See the **Multi-Stream Domain** chapter for the full element placement matrix.

Problem: Missing `DXFrameMeta` inside the multi-stream domain

One of the following symptoms appears at runtime:

- `DxInfer` posts an error:

```
ERROR dxinfer ... No DXFrameMeta in push buffer
```

- `DxPostprocess`, `DxTracker`, or `DxOsd` pass the buffer through unchanged, so no results or overlays appear. The only trace is a log-level message, visible with `GST_DEBUG=dx*:6`:

```
LOG   dxpostprocess ... No DXFrameMeta, passing through
LOG   dxosd ... No frame metadata, passing through
```

- A dual-mode element such as `DxRate` or `DxMsgConv` falls back to `stream_id=0` for every buffer.

Cause: Buffer entered the domain without `DXFrameMeta`

`DXFrameMeta` is normally attached by `DxInputSelector` (or, in single-stream pipelines, by upstream DX-STREAM elements via the allocation query). If a buffer is injected into the domain through some other path — for example a custom `appsrc` placed after `DxInputSelector`, or a test element that does not propose the meta — dual-mode elements have no `_stream_id` to dispatch on.

© Copyright 2026 DEEPX All Rights Reserved.

Solution

- Make sure every buffer entering the domain originates from a sink pad of `DxInputSelector`, which guarantees `DXFrameMeta` is present.
- If you generate buffers yourself, attach `DXFrameMeta` with `dx_create_frame_meta()` and set `_stream_id` before pushing them.
- During `ALLOCATION`, dual-mode elements add a request for the `DXFrameMeta` API meta; respect that request in any custom upstream that implements a buffer pool.

© Copyright 2026 DEEPX All Rights Reserved.

7. Debugging Guide

This chapter provides comprehensive guidance on debugging DX-STREAM applications using GStreamer's built-in debug logging system. Understanding how to enable and interpret debug logs is essential for diagnosing issues, monitoring performance, and optimizing your AI inference pipelines.

7.1 GST_DEBUG Environment Variable

GStreamer provides a powerful debug logging system controlled by the `GST_DEBUG` environment variable. DX-STREAM elements fully integrate with this system, offering detailed insights into pipeline execution, buffer flow, and element-specific operations.

7.1.1 Debug Level Overview

The `GST_DEBUG` variable accepts numeric levels from 0 to 7:

Level	Name	Description
0	NONE	No debug output
1	ERROR	Fatal errors that prevent normal operation
2	WARNING	Non-fatal issues that may affect behavior
3	FIXME	Fixme messages (rarely used in DX-STREAM)
4	INFO	Informational messages about state changes and major operations
5	DEBUG	Detailed operational information useful for troubleshooting
6	LOG	

© Copyright 2026 DEEPX All Rights Reserved.

Level	Name	Description
		Per-buffer trace — every frame/buffer processed logs at this level
7	TRACE	Internal algorithm details (e.g., inference queue state, tensor contents)

7.1.2 Basic Usage

Enable debug logging for all DX-STREAM elements:

```
$ GST_DEBUG=dx*:4 ./your_application
```

Enable debug logging for specific elements:

```
$ GST_DEBUG=dxinfer:4,dxtracker:4 ./your_application
```

Combine global level with element-specific levels:

```
$ GST_DEBUG=2,dxinfer:4,dxpreprocess:4 ./your_application
```

This sets the global level to WARNING (2) while enabling INFO (4) for `dxinfer` and `dxpreprocess`.

7.1.3 Advanced Usage

Redirect debug output to a file:

```
$ GST_DEBUG=dx*:4 GST_DEBUG_FILE=/tmp/debug.log ./your_application
```

Disable colored output (useful for log files):

```
$ GST_DEBUG_NO_COLOR=1 GST_DEBUG=dx*:4 ./your_application
```

Filter output with grep:

```
$ GST_DEBUG=dxinfer:4 ./your_application 2>&1 | grep "Inference completed"
```


© Copyright 2026 DEEPX All Rights Reserved.

7.2 Element-Specific Debugging

Each DX-STREAM element has its own debug category that logs element-specific operations and state information.

7.2.1 DxPreprocess

Debug Category: `dxpreprocess`

Key Log Messages:

- `Transitioning to READY state` - Element state changes
- `Loaded configuration from: <path>` - Config file loading
- `Processing buffer: pts=<timestamp>` - Buffer flow tracking
- `Operating in secondary mode` - Mode confirmation
- `Dropping buffer due to QoS` - QoS-triggered frame drops

Common Use Cases:

```
# Debug preprocessing issues
$ GST_DEBUG=dxpreprocess:4 ./your_app

# Monitor QoS behavior
$ GST_DEBUG=dxpreprocess:4 ./your_app 2>&1 | grep "QoS"

# Trace buffer flow
$ GST_DEBUG=dxpreprocess:4 ./your_app 2>&1 | grep "Processing buffer"
```

7.2.2 DxInfer

Debug Category: `dxinfer`

Key Log Messages:

- `Model loaded: <path>` - Model initialization
- `Inference completed in X.XX ms` - Inference latency measurement
- `Queue status: X/Y buffers` - Input queue monitoring
- `Dropping buffer due to QoS` - Performance optimization
- `Received throttle QoS event` - Frame rate control

Common Use Cases:

© Copyright 2026 DEEPX All Rights Reserved.

```
# Monitor inference performance
$ GST_DEBUG=dxinfer:4 ./your_app 2>&1 | grep "completed in"

# Debug model loading
$ GST_DEBUG=dxinfer:3 ./your_app

# Analyze queue behavior
$ GST_DEBUG=dxinfer:4 ./your_app 2>&1 | grep "Queue status"
```

7.2.3 DxPostprocess

Debug Category: `dxpostprocess`

Key Log Messages:

- `Loaded postprocess library: <path>` - Custom library loading
- `Processing buffer: pts=<timestamp>` - Buffer processing
- `Operating in secondary mode` - Mode confirmation
- `Added X objects to frame metadata` - Detection results

Common Use Cases:

```
# Debug custom postprocess library
$ GST_DEBUG=dxpostprocess:4 ./your_app

# Monitor detection results
$ GST_DEBUG=dxpostprocess:4 ./your_app 2>&1 | grep "objects to frame"
```

7.2.4 DxTracker

Debug Category: `dxtracker`

Key Log Messages:

- `Tracker initialized: algorithm=<name>` - Tracker setup
- `Processing buffer: pts=<timestamp>` - Frame tracking
- `Tracking X objects` - Active track count
- `Updated track IDs` - ID assignment

Common Use Cases:

```
# Debug tracking behavior
$ GST_DEBUG=dxtracker:4 ./your_app
```


ⓒ Copyright 2026 DEEPX All Rights Reserved.

```
# Monitor track updates
$ GST_DEBUG=dxtracker:4 ./your_app 2>&1 | grep "Tracking"
```

7.2.5 DxOsd

Debug Category: `dxosd`

Key Log Messages:

- `Initializing OSD element (passthrough transform)` - Element initialization
- `Processing buffer: pts=<timestamp>` - Frame rendering
- `Drawing on NV12: N objects` - NV12 format visualization status
- `Drawing on I420: N objects` - I420 format visualization status

Common Use Cases:

```
# Debug rendering issues
$ GST_DEBUG=dxosd:4 ./your_app

# Monitor draw operations
$ GST_DEBUG=dxosd:4 ./your_app 2>&1 | grep "Drawing"
```

7.2.6 DxMsgBroker

Debug Category: `dxmsgbroker`

Key Log Messages:

- `Connected to broker: <host>:<port>` - Connection status
- `Publishing message to topic: <topic>` - Message transmission
- `Connection lost, reconnecting` - Connection recovery
- `Message sent successfully` - Delivery confirmation

Common Use Cases:

```
# Debug broker connection
$ GST_DEBUG=dxmsgbroker:4 ./your_app

# Monitor message publishing
$ GST_DEBUG=dxmsgbroker:4 ./your_app 2>&1 | grep "Publishing"
```


© Copyright 2026 DEEPX All Rights Reserved.

```
# Track connection issues
$ GST_DEBUG=dxmsgbroker:3 ./your_app 2>&1 | grep -E "Connected|lost"
```

7.2.7 DxMsgConv

Debug Category: `dxmsgconv`

Key Log Messages:

- `Processing buffer: pts=<timestamp> seq=<id>` - Message conversion
- `Loaded message conversion library: <path>` - Custom library loading
- `Generated message payload: size=<bytes>` - Payload creation

Common Use Cases:

```
# Debug message conversion
$ GST_DEBUG=dxmsgconv:4 ./your_app

# Monitor sequence numbers
$ GST_DEBUG=dxmsgconv:4 ./your_app 2>&1 | grep "seq="
```

7.2.8 DxRate

Debug Category: `dxrate`

Key Log Messages:

- `Processing buffer: pts=<timestamp>` - Rate control operation
- `Sending throttle QoS event: delay=<ms>` - Throttling notification
- `Frame rate control active` - Rate limiting status

Common Use Cases:

```
# Debug frame rate control
$ GST_DEBUG=dxrate:4 ./your_app

# Monitor throttling
$ GST_DEBUG=dxrate:4 ./your_app 2>&1 | grep "throttle"
```

7.2.9 DxGather, DxInputSelector, DxOutputSelector

Debug Categories: `dxgather`, `dxinputselector`, `dxoutputselector`

© Copyright 2026 DEEPX All Rights Reserved.

Key Log Messages:

- `Processing buffer: pts=<timestamp>` - Stream routing
- `Selecting input stream: <id>` - Input selection (DxInputSelector)
- `Routing to output pad: <id>` - Output routing (DxOutputSelector)
- `Merging X streams` - Stream merging (DxGather)

Common Use Cases:

```
# Debug multi-stream routing
$ GST_DEBUG=dxgather:4,dxinputselector:4,dxoutputselector:4 ./your_app
```

7.2.10 DxScale, DxConvert

Debug Categories: `dxscale` , `dxconvert`

Key Log Messages:

- `Setting target resolution: <width>x<height>` - Scale target configuration (DxScale)
- `Backend selected: <name>` - Transform backend selection
- `Passthrough mode enabled` - No-op when no transform is needed
- `Falling back to libyuv backend` - RGA does not support the source format

Common Use Cases:

```
# Debug video scaling and format conversion
$ GST_DEBUG=dxscale:4,dxconvert:4 ./your_app
```

7.3 Metadata Debugging

DX-STREAM uses custom GStreamer metadata structures to pass information between elements. All metadata logging is unified under a single debug category.

7.3.1 Metadata Category

Debug Category: `dxmeta`

Covers: DXFrameMeta, DXObjectMeta, DXMsgMeta, DXUserMeta

© Copyright 2026 DEEPX All Rights Reserved.

Key Log Messages:

- `Creating DXFrameMeta` - Frame metadata creation
- `Adding DXObjectMeta to DXFrameMeta` - Object attachment
- `Acquiring DXUserMeta from pool` - Custom metadata allocation
- `Releasing DXUserMeta` - Memory cleanup

Usage:

```
# Debug all metadata operations
$ GST_DEBUG=dxmeta:5 ./your_app

# Monitor object metadata
$ GST_DEBUG=dxmeta:5 ./your_app 2>&1 | grep "DXObjectMeta"
```

7.4 Subsystem Debugging

DX-STREAM internal subsystems (shared across multiple elements) use dedicated debug categories for focused diagnostics.

7.4.1 Transform Kernel

Debug Category: `transform_kernel`

Covers: libyuv, RGA, V3 DSP kernels, TransformKernelPool, PreprocessorFactory

Key Log Messages:

- `TransformKernelPool: using backend '<name>' (single/pool mode)` - Backend selection
- `RgaTransformKernel: imcheck rejected` - RGA validation failure with fallback
- `TransformKernelPool: '<name>' failed, falling back to libyuv` - Automatic fallback
- `PreprocessorFactory: kernel pool created` - Pool initialization

Usage:

```
# Debug video transform operations (resize, color convert, crop)
$ GST_DEBUG=transform_kernel:5 ./your_app

# Trace per-frame transform execution
$ GST_DEBUG=transform_kernel:6 ./your_app
```


© Copyright 2026 DEEPX All Rights Reserved.

7.4.2 Inference Backend

Debug Category: `inference_backend`

Covers: DXRT runtime, InferBackendFactory

Key Log Messages:

- `DxrtBackend: initialized (DXRT <version>, <N> devices)` - DXRT init
- `Put req_id=<id> (pending=<N>)` - Inference request submitted (TRACE level)
- `Got req_id=<id> (pending=<N>)` - Inference result received (TRACE level)
- `Get SKIP: seq=<N>, device[<idx>]` - Inference timeout

Usage:

```
# Debug inference backend initialization and errors
$ GST_DEBUG=inference_backend:5 ./your_app

# Trace per-request inference flow (very verbose)
$ GST_DEBUG=inference_backend:7 ./your_app
```

7.5 Common Debugging Scenarios

7.5.1 Performance Analysis

Monitor inference latency:

```
$ GST_DEBUG=dxinfer:4 ./your_app 2>&1 | grep "completed in" | awk '{print $NF}' | sed 's/ms//'
```

Identify bottlenecks:

```
$ GST_DEBUG=dx*:4 ./your_app 2>&1 | grep -E "completed in|took|duration"
```

7.5.2 Pipeline Flow Debugging

Trace buffer flow through entire pipeline:

```
$ GST_DEBUG=dx*:4 ./your_app 2>&1 | grep "Processing buffer"
```

Monitor state transitions:

© Copyright 2026 DEEPX All Rights Reserved.

```
$ GST_DEBUG=dx*:3 ./your_app 2>&1 | grep -E "Transitioning|state"
```

7.5.3 QoS and Frame Drop Analysis

Track dropped frames:

```
$ GST_DEBUG=dxpreprocess:4,dxinfer:4 ./your_app 2>&1 | grep "Dropping"
```

Monitor QoS events:

```
$ GST_DEBUG=dx*:4 ./your_app 2>&1 | grep "QoS"
```

7.5.4 Configuration Issues

Verify element initialization:

```
$ GST_DEBUG=dx*:3 ./your_app 2>&1 | grep -E "Loaded|initialized|configuration"
```

Check property values:

```
$ GST_DEBUG=dx*:3 ./your_app 2>&1 | grep -E "property|config"
```

7.5.5 Memory and Resource Tracking

Monitor metadata lifecycle:

```
$ GST_DEBUG=dxmeta:5 ./your_app 2>&1 | grep -E "Creating|Freeing|Releasing"
```

Track buffer allocation:

```
$ GST_DEBUG=dx*:4 ./your_app 2>&1 | grep -E "Acquiring|Releasing|pool"
```

7.6 Best Practices

7.6.1 Development Phase

1. **Start with INFO level** (`:4`) to monitor state changes and initialization
2. **Enable DEBUG level** (`:5`) for specific elements showing issues

© Copyright 2026 DEEPX All Rights Reserved.

3. **Use targeted filtering** with grep to focus on relevant messages
4. **Capture logs to file** for detailed analysis

```
$ GST_DEBUG=2,dxinfer:4,dxtracker:4 GST_DEBUG_FILE=/tmp/debug.log ./your_app
```

7.6.2 Production Deployment

(1) **Set global level to WARNING** (`:2`) or ERROR (`:1`)
(2) **Disable unnecessary element logging** to reduce overhead
(3) **Use log rotation** if file logging is enabled
(4) **Monitor ERROR and WARNING** messages for operational issues

```
$ GST_DEBUG=1 ./your_app 2>&1 | tee -a /var/log/dxstream.log
```

7.6.3 Performance Testing

(1) **Minimize debug output** during benchmarking
(2) **Use selective logging** for targeted profiling
(3) **Capture timing information** for latency analysis

```
$ GST_DEBUG=dxinfer:4 ./your_app 2>&1 | grep "completed in" > inference_times.txt
```

7.7 Tips and Tricks

7.7.1 Combining Multiple Tools

Use timestamps for timing analysis:

```
$ GST_DEBUG=dx*:4 ./your_app 2>&1 | ts '[%Y-%m-%d %H:%M:%.S]'
```

Pipe to text editor for analysis:

```
$ GST_DEBUG=dx*:4 ./your_app 2>&1 | tee debug.log | less
```

7.7.2 Quick Diagnostics

Check if elements are loading:

© Copyright 2026 DEEPX All Rights Reserved.

```
$ gst-inspect-1.0 dxinfer
```

Verify debug categories:

```
$ GST_DEBUG=dx*:3,dxmeta:3,transform_kernel:3,inference_backend:3 gst-launch-1.0 --gst-debug-help | grep -E "dx|transform_kernel|inference_backend|dxmeta"
```

7.7.3 Common Patterns

Find all ERROR messages:

```
$ GST_DEBUG=dx*:4 ./your_app 2>&1 | grep "ERROR"
```

Count specific events:

```
$ GST_DEBUG=dxinfer:4 ./your_app 2>&1 | grep "Inference completed" | wc -l
```

Extract PTS timestamps:

```
$ GST_DEBUG=dx*:4 ./your_app 2>&1 | grep -oP 'pts=\K[0-9:]+'
```

7.8 Additional Resources

- **GStreamer Debug Documentation:** https://gstreamer.freedesktop.org/documentation/gstreamer/running.html
- **DX-STREAM Troubleshooting:** See Chapter 06 - Troubleshooting and FAQ
- **Element-Specific Documentation:** See Chapter 03 - DX-STREAM Elements

> **Tip**
>
> When reporting issues, always include relevant debug logs with `GST_DEBUG=dx*:4` to help diagnose problems quickly.

© Copyright 2026 DEEPX All Rights Reserved.

8. DX-STREAM Agent-Driven Development Guide

8.1 Overview

This guide describes how to use DEEPX agent-driven development (dx-agent-dev) to build GStreamer pipeline applications with **dx_stream** and DEEPX NPU accelerators. Agents handle pipeline construction, model management, and validation so you can go from a natural-language request to a working pipeline script in minutes.

8.2 Agents

Four agents collaborate to build, configure, and validate dx_stream pipelines.

Agent	Description	Routes To
`dx-stream-builder`	Master router — classifies the pipeline type from the user request and dispatches to the appropriate specialist agent	`dx-pipeline-builder`, `dx-model-manager`, `dx-validator`
`dx-pipeline-builder`	Builds GStreamer pipeline apps across 5 categories (single-model, multi-model, cascaded, parallel, broker)	—
`dx-model-manager`	Downloads and configures `.dxnn` models for use in pipelines	—
`dx-validator`	Validates generated pipeline scripts and `.deepx/` framework integrity	—

8.2.1 Routing Flow

```
User Request
    └─▶ dx-stream-builder (classify & route)
```


© Copyright 2026 DEEPX All Rights Reserved.

```
        ├─► dx-pipeline-builder (generate pipeline)
        ├─► dx-model-manager   (resolve models)
        └─► dx-validator       (validate output)
```

8.3 Skills

General SWE Process

Skill	Description
`dx-agent-brainstorm`	Brainstorm and plan before any pipeline generation
`dx-agent-tdd`	Test-driven development — validate each file immediately after creation
`dx-agent-verify`	Verify before claiming completion — evidence before assertions
`dx-swe-writing-plans`	Write implementation plans from specs or requirements
`dx-swe-executing-plans`	Execute a written implementation plan with review checkpoints
`dx-swe-debugging`	Systematic debugging — diagnose before proposing fixes
`dx-swe-parallel-agents`	Dispatch 2+ independent tasks to parallel agents
`dx-swe-subagent-dev`	Execute implementation plans with independent sub-agent tasks
`dx-swe-receiving-review`	Receive and process code review feedback with technical rigor
`dx-swe-requesting-review`	Request code review to verify work meets requirements
`dx-skill-router`	Route to the appropriate skill based on task classification

ⓒ Copyright 2026 DEEPX All Rights Reserved.

DEEPX Build

Skill	Description
`dx-agent-stream-build-pipeline`	Build a GStreamer pipeline across 5 categories: single-model, multi-model, cascaded, parallel, broker
`dx-agent-stream-build-mqtt-kafka`	Build an MQTT or Kafka message broker pipeline for event publishing
`dx-agent-stream-model-management`	Download and configure `.dxnn` models for target NPU architecture
`dx-agent-stream-validate`	Run pipeline validation checks (syntax, properties, element order)

DEEPX Harness (Internal)

Skill	Description
`dx-harness-validate`	Validate the integrity of `.deepx/` (internal harness development)
`dx-harness-writing-skills`	Create, edit, and verify skills before deployment (internal harness development)

8.4 Supported AI Tools

dx_stream agent-driven development works with four AI coding tools. Each auto-loads the knowledge base through its own configuration.

Tool	Config Files	Agents Available
Claude Code	`CLAUDE.md`	All 4 agents via context routing
GitHub Copilot	`.github/copilot-instructions.md`, 4 agents in `.github/agents/`, 17 skills in `.github/skills/`, 2	`@dx-stream-builder`, `@dx-pipeline-builder`, `@dx-model-manager`, `@dx-validator`

© Copyright 2026 DEEPX All Rights Reserved.

Tool	Config Files	Agents Available
	instructions in `.github/instructions/`	
Cursor	`.cursor/rules/dx-stream.mdc` (always), `dx-model-manager.mdc`, `dx-pipeline-builder.mdc`, `dx-stream-builder.mdc`, `dx-validator.mdc`, `stream-pipelines.mdc`, `tests.mdc`, 17 `skill-*.mdc` files (24 total)	Free-form with auto-applied rules
OpenCode	`AGENTS.md`, `opencode.json`, 4 agents in `.opencode/agents/`, 17 skills in `.deepx/skills/`	`@dx-stream-builder` or `/dx-agent-stream-build-pipeline`

8.4.1 Copilot File-Specific Instructions

Glob Pattern	Injected Instruction	Content
`**/pipeline/**`, `**/pipelines/**`, `**/*pipeline*.py`	`stream-pipelines.instructions.md`	preprocess-id matching, queue placement, DxRate for RTSP, element ordering
`test/**`	`tests.instructions.md`	pytest patterns, pipeline fixtures, mock elements

8.4.2 OpenCode Skills (Slash Commands)

General SWE Process

Slash Command	Description
`/dx-agent-brainstorm`	Brainstorm and plan before pipeline generation
`/dx-agent-tdd`	Test-driven development with incremental validation
`/dx-agent-verify`	Verify completion with evidence before assertions

ⓒ Copyright 2026 DEEPX All Rights Reserved.

Slash Command	Description
`/dx-swe-writing-plans`	Write implementation plans from specs
`/dx-swe-executing-plans`	Execute a written implementation plan
`/dx-swe-debugging`	Systematic debugging before proposing fixes
`/dx-swe-parallel-agents`	Dispatch 2+ independent tasks to parallel agents
`/dx-swe-subagent-dev`	Execute plans with independent sub-agent tasks
`/dx-swe-receiving-review`	Process code review feedback
`/dx-swe-requesting-review`	Request code review
`/dx-skill-router`	Route to the appropriate skill

DEEPX Build

Slash Command	Description
`/dx-agent-stream-build-pipeline`	Build a GStreamer pipeline across 5 categories
`/dx-agent-stream-build-mqtt-kafka`	Build an MQTT/Kafka broker pipeline
`/dx-agent-stream-model-management`	Download and configure .dxnn models
`/dx-agent-stream-validate`	Run pipeline validation checks

DEEPX Harness (Internal)

Slash Command	Description
`/dx-harness-validate`	Validate the integrity of `.deepx/`
`/dx-harness-writing-skills`	Create, edit, and verify skills before deployment

8.4.3 Platform File Loading Reference

Each AI coding agent auto-loads different configuration files at the dx_stream level.

ⓒ Copyright 2026 DEEPX All Rights Reserved.

Auto-Loaded Files

File	Auto-loaded by	Loading
`.github/copilot-instructions.md`	Copilot Chat/CLI	Auto
`CLAUDE.md`	Claude Code	Auto
`AGENTS.md` + `opencode.json`	OpenCode	Auto
`AGENTS.md` + `.codex/skills/dx-codex-identity/SKILL.md`	Codex CLI	Auto
`.cursor/rules/dx-stream.mdc`	Cursor	Auto

Agent Files (Manual @mention)

Agent	Copilot (`@mention`)	OpenCode (`@mention`)	Claude Code (`@mention`)
`dx-stream-builder`	`.github/agents/dx-stream-builder.agent.md`	`.opencode/agents/dx-stream-builder.md`	`.claude/agents/dx-stream-builder.md`
`dx-pipeline-builder`	`.github/agents/dx-pipeline-builder.agent.md`	`.opencode/agents/dx-pipeline-builder.md`	`.claude/agents/dx-pipeline-builder.md`
`dx-model-manager`	`.github/agents/dx-model-manager.agent.md`	`.opencode/agents/dx-model-manager.md`	`.claude/agents/dx-model-manager.md`
`dx-validator`	`.github/agents/dx-validator.agent.md`	`.opencode/agents/dx-validator.md`	`.claude/agents/dx-validator.md`

Skill Files (All Platforms — `/slash-command`)

Skill	File
`/dx-agent-brainstorm`	`.deepx/skills/dx-agent-brainstorm/SKILL.md`
`/dx-agent-stream-build-mqtt-kafka`	`.deepx/skills/dx-agent-stream-build-mqtt-kafka/SKILL.md`
`/dx-agent-stream-build-pipeline`	

ⓒ Copyright 2026 DEEPX All Rights Reserved.

Skill	File
	`.deepx/skills/dx-agent-stream-build-pipeline/SKILL.md`
`/dx-swe-parallel-agents`	`.deepx/skills/dx-swe-parallel-agents/SKILL.md`
`/dx-swe-executing-plans`	`.deepx/skills/dx-swe-executing-plans/SKILL.md`
`/dx-agent-stream-model-management`	`.deepx/skills/dx-agent-stream-model-management/SKILL.md`
`/dx-swe-receiving-review`	`.deepx/skills/dx-swe-receiving-review/SKILL.md`
`/dx-swe-requesting-review`	`.deepx/skills/dx-swe-requesting-review/SKILL.md`
`/dx-skill-router`	`.deepx/skills/dx-skill-router/SKILL.md`
`/dx-swe-subagent-dev`	`.deepx/skills/dx-swe-subagent-dev/SKILL.md`
`/dx-swe-debugging`	`.deepx/skills/dx-swe-debugging/SKILL.md`
`/dx-agent-tdd`	`.deepx/skills/dx-agent-tdd/SKILL.md`
`/dx-agent-stream-validate`	`.deepx/skills/dx-agent-stream-validate/SKILL.md`
`/dx-agent-verify`	`.deepx/skills/dx-agent-verify/SKILL.md`
`/dx-swe-writing-plans`	`.deepx/skills/dx-swe-writing-plans/SKILL.md`
`/dx-harness-validate`	`.deepx/skills/dx-harness-validate/SKILL.md`
`/dx-harness-writing-skills`	`.deepx/skills/dx-harness-writing-skills/SKILL.md`

Shared Knowledge Base (`.deepx/`)

The `.deepx/` directory is the canonical source for all agent-driven development content. Platform-specific files (`.claude/` , `.github/` , `.cursor/` , `.opencode/`) are generated from `.deepx/` by `dx-agent-gen generate --repo dx-runtime/dx_stream` . Never edit generated files directly — update `.deepx/` and regenerate.

Directory	Files	Description
`.deepx/agents/`	4 files (`dx-stream-builder.md` , `dx-pipeline-builder.md` , `dx-`	Authoritative agent definitions

ⓒ Copyright 2026 DEEPX All Rights Reserved.

Directory	Files	Description
	`model-manager.md`, `dx-validator.md`)	
`.deepx/skills/`	17 directories (one per skill, each containing `SKILL.md`)	Detailed skill workflows
`.deepx/toolsets/`	4 files	GStreamer elements and API references
`.deepx/instructions/`	6 files	Coding standards and workflow rules
`.deepx/memory/`	4 files	Persistent knowledge — pitfalls and session memory
`.deepx/templates/`	Template files	Code generation templates
`.deepx/knowledge/`	Knowledge files	Domain-specific knowledge bases
`.deepx/contextual-rules/`	Rule files	Context-dependent rules for agents
`.deepx/prompts/`	Prompt files	Reusable prompt fragments
`.deepx/scripts/`	Script files	Validation and utility scripts

Claude Code Agents (`.claude/agents/`)

Agent	File
`dx-model-manager`	`.claude/agents/dx-model-manager.md`
`dx-pipeline-builder`	`.claude/agents/dx-pipeline-builder.md`
`dx-stream-builder`	`.claude/agents/dx-stream-builder.md`
`dx-validator`	`.claude/agents/dx-validator.md`

GitHub Copilot Skills (`.github/skills/`)

17 skill directories (inline copies generated by `dx-agent-gen`), one per skill matching `.deepx/skills/`.

© Copyright 2026 DEEPX All Rights Reserved.

8.5 Generation Pipeline

All platform-specific configuration files are generated from the canonical `.deepx/` source by the `dx-agent-gen` tool:

```
dx-agent-gen generate --repo dx-runtime/dx_stream
```

This generates: - `.claude/agents/` — Claude Code agent files - `.github/agents/`, `.github/skills/`, `.github/instructions/` — GitHub Copilot files - `.cursor/rules/` — Cursor rule files (24 total) - `.opencode/agents/` — OpenCode agent files - `CLAUDE.md`, `AGENTS.md`, `opencode.json` — Platform entry points

Never edit generated files directly. Update `.deepx/` and regenerate.

8.6 User Scenarios

8.6.1 Scenario 1: Build a Detection Pipeline with Tracking

Prompt:

```
"Build an object detection pipeline with yolo26n and tracking on RTSP camera"
```

Tool	How to Use
Claude Code	Type the prompt directly. `CLAUDE.md` routes to `dx-agent-stream-build-pipeline` skill. Asks about RTSP URL, display preferences, and tracker type, then generates the pipeline with DxRate → DxPreprocess → DxInfer → DxTracker → DxOsd chain.
GitHub Copilot	`@dx-stream-builder` followed by the prompt. Classifies as "single-model + tracking", hands off to `dx-pipeline-builder`, runs `dx-validator`.
Cursor	Type the prompt directly. `dx-stream.mdc` (always loaded) provides the 13-element catalog. `stream-pipelines.mdc` activates for pipeline files.

© Copyright 2026 DEEPX All Rights Reserved.

Tool	How to Use
OpenCode	`@dx-stream-builder` followed by the prompt, or `/dx-agent-stream-build-pipeline` skill directly.

8.6.2 Scenario 2: Build an MQTT Broker Pipeline

Prompt:

> "Build a pipeline that detects people and publishes events to MQTT"

Tool	How to Use
Claude Code	Type the prompt directly. Routes to `dx-agent-stream-build-mqtt-kafka` skill. Generates a pipeline ending with `DxPostprocess ! DxMsgConv ! DxMsgBroker`.
GitHub Copilot	`@dx-stream-builder` followed by the prompt.
Cursor	Type the prompt directly.
OpenCode	`@dx-stream-builder` followed by the prompt, or `/dx-agent-stream-build-mqtt-kafka` skill directly.

8.6.3 Scenario 3: Multi-Model Cascaded Pipeline

Prompt:

> "Build a cascaded pipeline: first detect people, then classify their actions"

Tool	How to Use
Claude Code	Type the prompt directly. Generates cascaded pattern: `DxInfer (primary) → DxPostprocess → DxTracker → tee → DxPreprocess(secondary-mode=true) → DxInfer(secondary-mode=true) → DxGather`.
GitHub Copilot	`@dx-pipeline-builder` followed by the prompt.

© Copyright 2026 DEEPX All Rights Reserved.

Tool	How to Use
Cursor	Type the prompt directly.
OpenCode	`@dx-stream-builder` followed by the prompt, or `/dx-agent-stream-build-pipeline` skill directly.

8.6.4 Scenario 4: Validate a Pipeline

Prompt:

> "Validate the pipeline I just created"

Tool	How to Use
Claude Code	`@dx-validator` followed by the prompt.
GitHub Copilot	`@dx-validator` followed by the prompt.
Cursor	Type the prompt directly.
OpenCode	`@dx-validator` followed by the prompt, or run manually: `python .deepx/scripts/validate_app.py <script.sh>`

8.6.5 Scenario 5: Build a Pose Estimation Pipeline

Prompt:

> "Build a pose estimation pipeline with yolo26n-pose on USB camera"

Tool	How to Use
Claude Code	Type the prompt directly. Routes to `dx-agent-stream-build-pipeline` with pose estimation model. Generates pipeline with keypoint overlay via `DxOsd`.
GitHub Copilot	`@dx-stream-builder` followed by the prompt. Classifies as "single-model + pose", hands off to `dx-pipeline-builder`.

© Copyright 2026 DEEPX All Rights Reserved.

Tool	How to Use
Cursor	Type the prompt directly. `stream-pipelines.mdc` activates for generated pipeline files.
OpenCode	`@dx-stream-builder` followed by the prompt, or `/dx-agent-stream-build-pipeline` skill directly.

8.6.6 Scenario 6: Build a Multi-Stream Parallel Pipeline

Prompt:

> "Build a parallel pipeline processing 4 RTSP cameras with yolo26n"

Tool	How to Use
Claude Code	Type the prompt directly. Routes to `dx-agent-stream-build-pipeline` with parallel category. Generates `DxInputSelector` + `DxOutputSelector` for shared inference, or independent sub-pipelines per stream.
GitHub Copilot	`@dx-pipeline-builder` followed by the prompt.
Cursor	Type the prompt directly.
OpenCode	`@dx-stream-builder` followed by the prompt, or `/dx-agent-stream-build-pipeline` skill directly.

8.6.7 Scenario 7: Build a Segmentation Pipeline

Prompt:

> "Build a segmentation pipeline with yolo26n-seg for road scene analysis"

Tool	How to Use
Claude Code	Type the prompt directly. Routes to `dx-agent-stream-build-pipeline` with segmentation model. Generates pipeline with per-pixel mask overlay via `DxOsd`.

© Copyright 2026 DEEPX All Rights Reserved.

Tool	How to Use
GitHub Copilot	`@dx-stream-builder` followed by the prompt. Classifies as "single-model + segmentation".
Cursor	Type the prompt directly. `stream-pipelines.mdc` activates for generated pipeline files.
OpenCode	`@dx-stream-builder` followed by the prompt, or `/dx-agent-stream-build-pipeline` skill directly.

8.7 Quick Start

Invoke the master router with a natural-language description:

```
@dx-stream-builder "object detection pipeline with yolo26n and tracking on RTSP camera"
```

The agent will:

1. **Ask clarifying questions** — pipeline category, source type, model variant, output sink
2. **Present a pipeline plan** — element chain, properties, model path
3. **Route to `dx-pipeline-builder`** — generate the pipeline script
4. **Resolve the model** — download or locate the `.dxnn` file via `dx-model-manager`
5. **Validate and report** — run `dx-validator` and return results

8.8 What Gets Created

By default, AI-generated pipeline code is placed in the `dx-agent-dev/` isolation directory to prevent conflicts with existing scripts.

8.8.1 Default Output (dx-agent-dev/)

```
dx-agent-dev/<session_id>/
├── README.md            # Session metadata and run instructions
├── session.json         # Machine-readable session config
└── {pipeline_name}.py   # Generated pipeline script
```

Session ID format: `YYYYMMDD-HHMMSS_model_task` (e.g., `20260403-150045_yolo26n_tracking`).

© Copyright 2026 DEEPX All Rights Reserved.

8.8.2 Production Output

When you explicitly request production placement, files are written to the standard pipeline directory.

8.9 Pipeline Categories

dx_stream supports five pipeline categories. Each follows a distinct GStreamer element pattern.

Category	Pattern	Key Elements
Single-model	`src ! DxPreprocess ! DxInfer ! DxPostprocess ! DxOsd ! sink`	Core inference trio + on-screen display
Multi-model	Chain multiple `DxInfer` stages, each with a distinct `preprocess-id`	Multiple inference passes in sequence
Cascaded	`DxPostprocess ! tee ! DxPreprocess(secondary) ! DxInfer ! DxGather`	Secondary inference branches merge after processing
Parallel	`DxInputSelector ! shared DxInfer ! DxOutputSelector`	Multiple streams share inference and return to their output pads
Broker	`DxPostprocess ! DxMsgConv ! DxMsgBroker`	Serialize detections and publish to MQTT/Kafka

8.10 GStreamer Elements Reference

dx_stream provides 13 custom GStreamer elements for NPU-accelerated pipelines.

Element	Purpose
`DxPreprocess`	Resize, normalize, and color-convert frames for model input
`DxInfer`	Run a `.dxnn` model on the DEEPX NPU

© Copyright 2026 DEEPX All Rights Reserved.

Element	Purpose
DxPostprocess	Decode raw tensors into structured detection/ classification results
DxTracker	Multi-object tracking (assign persistent IDs across frames)
DxOsd	Draw bounding boxes, labels, and overlays on frames
DxGather	N-to-1 merge — collect buffers from multiple branches
DxInputSelector	N-to-1 merge — combine frames from multiple streams in PTS order
DxOutputSelector	1-to-N demux — route buffers to one of N output pads
DxRate	Limit frame rate (recommended for variable-rate sources such as RTSP)
DxMsgConv	Serialize detection metadata to JSON
DxMsgBroker	Publish serialized messages to MQTT or Kafka
DxScale	Resize frames to a target resolution
DxConvert	Color space conversion between formats

8.11 Pipeline-Specific Validation Rules

ⓒ Copyright 2026 DEEPX All Rights Reserved.

Agents enforce these rules when generating and validating pipelines:

1. **`preprocess-id` matching** — `DxPreprocess` and `DxInfer` must share the same `preprocess-id` value. A mismatch causes silent inference failures.
2. **Queue placement** — Insert a `queue` element between every processing stage to prevent pipeline deadlocks.
3. **`DxRate` for RTSP** — Agents insert `DxRate` immediately after RTSP sources to prevent frame flooding and buffer overruns from variable-rate sources.
4. **`DxMsgConv` before `DxMsgBroker`** — Always serialize metadata with `DxMsgConv` before publishing with `DxMsgBroker`.
5. **Absolute model paths** — The `model-path` property on `DxInfer` must resolve to an absolute filesystem path (for example, derived from `$SRC_DIR`). A relative path depends on the working directory and can fail at runtime.

8.12 Validation Commands

```
# Validate a pipeline script (static checks)
python .deepx/scripts/validate_app.py <script.sh>

# Validate with smoke test (requires NPU hardware)
python .deepx/scripts/validate_app.py <script.sh> --smoke-test

# Check .deepx/ framework integrity
python .deepx/scripts/validate_framework.py
```

Validation checks include: - Element property completeness - `preprocess-id` consistency across `DxPreprocess` / `DxInfer` pairs - Queue placement between processing stages - Model file existence at the specified `model-path` - Broker element ordering (`DxMsgConv` before `DxMsgBroker`)

8.13 Session Sentinels

Agents output fixed markers at the start and end of each task for automated testing:

Marker	When
`[DX-AGENT-DEV: START]`	First line of the agent's response

© Copyright 2026 DEEPX All Rights Reserved.

Marker	When
`[DX-AGENT-DEV: DONE (output-dir: <relative_path>)]`	Last line after all work is complete. `<relative_path>` is the session output directory relative to the project root. If no files were generated, omit the `(output-dir: ...)` part.

Sub-agents invoked via handoff do not output sentinels — only the top-level agent does.

Rules: 1. **CRITICAL** — Output `[DX-AGENT-DEV: START]` as the absolute first line of the first response, before any other text. This is non-negotiable even if the user says to proceed autonomously. 2. Output `[DX-AGENT-DEV: DONE (output-dir: <path>)]` as the very last line after all work is complete. 3. Sub-agents invoked via handoff do not output sentinels. 4. If the user sends multiple prompts in a session, output START/DONE for each prompt. 5. The `output-dir` in DONE must be the relative path from the project root to the session output directory. 6. **Never output DONE after only producing planning artifacts** (specs, plans, design documents). DONE means all deliverables are produced — implementation code, scripts, configs, and validation results.

8.14 Troubleshooting

Symptom	Cause	Fix
Inference produces no detections	`preprocess-id` mismatch between `DxPreprocess` and `DxInfer`	Ensure both elements share the same `preprocess-id` value
Pipeline deadlocks or stalls	Missing `queue` elements between processing stages	Add a `queue` element between every pair of processing elements
RTSP stream drops frames	No frame rate limiting after source	Insert `DxRate` immediately after the RTSP source element
`model-path` not found at runtime	Relative path resolved against a different working directory	Use an absolute path (e.g., `/opt/deepx/models/yolo26-n_640x640.dxnn`)
`DxInfer` plugin not registered	Plugin not installed or `GST_PLUGIN_PATH` not set	Run `gst-inspect-1.0 dxinfer` to verify; check `GST_PLUGIN_PATH`

© Copyright 2026 DEEPX All Rights Reserved.

Symptom	Cause	Fix
Broker pipeline sends empty messages	`DxMsgConv` missing before `DxMsgBroker`	Add `DxMsgConv` to serialize metadata before the broker element

ⓒ Copyright 2026 DEEPX All Rights Reserved.

9. Appendix

9.1 User Metadata Guide

This guide provides comprehensive information on using the DX-STREAM User Metadata system for storing custom data alongside inference results.

9.1.1 Core Concepts & Architecture

The DX-STREAM User Metadata system allows developers to attach custom data to frames and objects throughout the pipeline. This enables storing additional analytics results, custom features, tracking information, or any application-specific data.

Key Features

- **Simplified Type System**: Two main categories - Frame and Object metadata
- **Memory Safety**: Required copy and release functions ensure proper memory management
- **Lifecycle Management**: Automatic cleanup when metadata is no longer needed
- **Pipeline Integration**: Seamless integration with existing DX-STREAM elements

User Metadata Types

```
enum class DXUserMetaType {
    DX_USER_META_FRAME = 0x1000,   // Frame-level user metadata
    DX_USER_META_OBJECT = 0x2000,  // Object-level user metadata
};
```

When to use Frame-level metadata:

- Scene-level analytics (crowd count, scene classification)
- Frame-wide processing results
- Pipeline statistics or timing information
- Global configuration or state data

© Copyright 2026 DEEPX All Rights Reserved.

When to use Object-level metadata:

- Per-object features or analytics
- Object-specific tracking data
- Custom object attributes or classifications
- Object relationship information

9.1.2 Workflow Implementation

Critical Development Sequence

Custom metadata integration must follow these three sequential steps precisely. Skipping or reordering any stage will break compilation or cause critical memory runtime issues.

Step 1. Define Your Data Structure

First, create your custom structures. If your data structure contains pointers, ensure you handle deep copying in the next step.

```
// Example: Custom analytics data for frames
typedef struct {
    gint total_objects;
    gfloat scene_confidence;
    gchar *scene_type;
    guint64 processing_time_us;
} SceneAnalytics;

// Example: Custom feature data for objects
typedef struct {
    gfloat custom_score;
    gint feature_dim;
    gfloat *feature_vector;
    gchar *algorithm_name;
} CustomObjectFeature;
```

Step 2. Implement Required Functions

Copy Function (Deep copy your data):

```
static gpointer scene_analytics_copy(gconstpointer src) {
    const SceneAnalytics *src_data = (const SceneAnalytics *)src;
    SceneAnalytics *dst_data = g_new0(SceneAnalytics, 1);

    dst_data->total_objects = src_data->total_objects;
```


© Copyright 2026 DEEPX All Rights Reserved.

```
    dst_data->scene_confidence = src_data->scene_confidence;
    dst_data->scene_type = g_strdup(src_data->scene_type);  // Deep copy string
    dst_data->processing_time_us = src_data->processing_time_us;

    return dst_data;
}

static gpointer custom_feature_copy(gconstpointer src) {
    const CustomObjectFeature *src_data = (const CustomObjectFeature *)src;
    CustomObjectFeature *dst_data = g_new0(CustomObjectFeature, 1);

    dst_data->custom_score = src_data->custom_score;
    dst_data->feature_dim = src_data->feature_dim;

    // Deep copy feature vector
    dst_data->feature_vector = g_new(gfloat, src_data->feature_dim);
    memcpy(dst_data->feature_vector, src_data->feature_vector,
           src_data->feature_dim * sizeof(gfloat));

    dst_data->algorithm_name = g_strdup(src_data->algorithm_name);

    return dst_data;
}
```

Release Function (Clean up your data):

```
static void scene_analytics_free(gpointer data) {
    SceneAnalytics *analytics = (SceneAnalytics *)data;
    g_free(analytics->scene_type);
    g_free(analytics);
}

static void custom_feature_free(gpointer data) {
    CustomObjectFeature *feature = (CustomObjectFeature *)data;
    g_free(feature->feature_vector);
    g_free(feature->algorithm_name);
    g_free(feature);
}
```

Step 3. Create and Attach Metadata

Frame Metadata Example

```
void add_scene_analytics_to_frame(DXFrameMeta *frame_meta,
                                  gint object_count,
                                  const gchar *scene_type,
                                  guint64 processing_time_us) {
    // Allocate user metadata
    DXUserMeta *user_meta = dx_acquire_user_meta_from_pool();
```


© Copyright 2026 DEEPX All Rights Reserved.

```
    // Create and populate your data
    SceneAnalytics *analytics = g_new0(SceneAnalytics, 1);
    analytics->total_objects = object_count;
    analytics->scene_confidence = 0.89f;
    analytics->scene_type = g_strdup(scene_type);
    analytics->processing_time_us = processing_time_us;  // Elapsed time in microseconds

    // Set data with required functions
    gboolean success = dx_user_meta_set_data(user_meta,
                                    analytics,
                                    sizeof(SceneAnalytics),
                                    DXUserMetaType::DX_USER_META_FRAME,
                                    scene_analytics_free,    // Required cleanup
                                    scene_analytics_copy);   // Required copy

    if (success) {
        // Attach to frame; on failure, release_func frees the data
        if (!dx_add_user_meta_to_frame(frame_meta, user_meta)) {
            g_warning("Failed to attach user metadata to frame");
            dx_release_user_meta(user_meta);
        }
    } else {
        g_warning("Failed to set user metadata");
        scene_analytics_free(analytics);  // set_data failed, so the data is still owned by the caller
        dx_release_user_meta(user_meta);
    }
}
```

Object Metadata Example

```
void add_custom_feature_to_object(DXObjectMeta *obj_meta,
                            const gchar *algorithm_name,
                            gfloat *features,
                            gint feature_count) {
    // Allocate user metadata
    DXUserMeta *user_meta = dx_acquire_user_meta_from_pool();

    // Create and populate feature data
    CustomObjectFeature *feature_data = g_new0(CustomObjectFeature, 1);
    feature_data->custom_score = 0.92f;
    feature_data->feature_dim = feature_count;
    feature_data->feature_vector = g_new(gfloat, feature_count);
    memcpy(feature_data->feature_vector, features, feature_count * sizeof(gfloat));
    feature_data->algorithm_name = g_strdup(algorithm_name);

    // Set data with required functions
    gboolean success = dx_user_meta_set_data(user_meta,
                                    feature_data,
                                    sizeof(CustomObjectFeature),
                                    DXUserMetaType::DX_USER_META_OBJECT,
                                    custom_feature_free,     // Required cleanup
```


© Copyright 2026 DEEPX All Rights Reserved.

```
                                custom_feature_copy);    // Required copy

    if (success) {
        // Attach to object; on failure, release_func frees the data
        if (!dx_add_user_meta_to_obj(obj_meta, user_meta)) {
            g_warning("Failed to attach user metadata to object");
            dx_release_user_meta(user_meta);
        }
    } else {
        g_warning("Failed to set object user metadata");
        custom_feature_free(feature_data);  // set_data failed, so the data is still owned by the caller
        dx_release_user_meta(user_meta);
    }
}
```

Complete Example: Postprocess Function with Analytics Metadata

Here's a complete example of a custom postprocess function that adds analytics metadata. In Primary Mode (`object_meta == nullptr`) the function is called once per frame, and in Secondary Mode it is called once per object, so the frame-level metadata is added only in Primary Mode:

```
// In your custom postprocess function
extern "C" void custom_analytics_postprocess(GstBuffer *buf,
                                std::vector<dxs::DXTensor> network_output,
                                DXFrameMeta *frame_meta,
                                DXObjectMeta *object_meta)
{
    gint64 start_us = g_get_monotonic_time();

    // Generate dummy features for demonstration
    auto add_features = [](DXObjectMeta *obj) {
        gfloat features[128];
        for (gint i = 0; i < 128; i++) {
            features[i] = g_random_double_range(0.0, 1.0);
        }
        add_custom_feature_to_object(obj, "custom_algorithm_v1", features, 128);
    };

    if (object_meta == nullptr) {
        // Primary Mode: called once per frame
        for (auto obj : frame_meta->_object_meta_list) {
            add_features(obj);
        }

        add_scene_analytics_to_frame(frame_meta,
                            frame_meta->_object_meta_list.size(),
                            "indoor_scene",
                            g_get_monotonic_time() - start_us);
```


© Copyright 2026 DEEPX All Rights Reserved.

```
    } else {
        // Secondary Mode: called once per object
        add_features(object_meta);
    }
}
```

9.1.3 Operations & Reliability Guide

Retrieve and Use Metadata

Reading Frame Metadata

```
void process_frame_metadata(DXFrameMeta *frame_meta) {
    auto user_metas = dx_get_frame_user_metas(frame_meta);

    for (auto user_meta : *user_metas) {
        // Check if this is frame-type metadata
        if (user_meta->user_meta_type == DXUserMetaType::DX_USER_META_FRAME) {
            SceneAnalytics *analytics = (SceneAnalytics *)user_meta->user_meta_data;

            g_print("Scene Analysis Results:\n");
            g_print("  Objects: %d\n", analytics->total_objects);
            g_print("  Scene: %s (confidence: %.2f)\n",
                  analytics->scene_type, analytics->scene_confidence);
            g_print("  Processing time: %" G_GUINT64_FORMAT " µs\n", analytics->processing_time_us)
;
        }
    }
}
```

Reading Object Metadata

```
void process_object_metadata(DXObjectMeta *obj_meta) {
    auto user_metas = dx_get_object_user_metas(obj_meta);

    for (auto user_meta : *user_metas) {
        // Check if this is object-type metadata
        if (user_meta->user_meta_type == DXUserMetaType::DX_USER_META_OBJECT) {
            CustomObjectFeature *feature = (CustomObjectFeature *)user_meta->user_meta_data;

            g_print("Custom Feature for Object %d:\n", obj_meta->_meta_id);
            g_print("  Algorithm: %s\n", feature->algorithm_name);
            g_print("  Score: %.3f\n", feature->custom_score);
            g_print("  Feature dimensions: %d\n", feature->feature_dim);

            // Access feature vector
            for (gint i = 0; i < feature->feature_dim && i < 5; i++) {
```


© Copyright 2026 DEEPX All Rights Reserved.

```
            g_print("  Feature[%d]: %.3f\n", i, feature->feature_vector[i]);
        }
    }
  }
}
```

Best Practices

Memory Management

- **Always provide both copy and release functions**: The system requires both for proper memory management
- **Deep copy all allocated memory**: Your copy function must properly duplicate all heap-allocated data
- **Clean up completely**: Your release function must free all memory allocated by your data structure

Performance Considerations

- **Allocation**: `dx_acquire_user_meta_from_pool()` allocates a new `DXUserMeta`; the name is historical and no pooling is performed, so avoid creating metadata unnecessarily
- **Minimal data size**: Keep metadata structures compact for better performance
- **Lazy evaluation**: Only compute expensive metadata when actually needed

Error Handling

- **Check return values**: Always verify that `dx_user_meta_set_data()` succeeds
- **Graceful degradation**: Handle metadata failures without breaking the pipeline
- **Logging**: Use appropriate log levels for debugging metadata issues

Integration Tips

- **Type verification**: Always check `user_meta_type` before casting metadata
- **Iterator safety**: Handle the case where metadata lists might be empty
- **Thread safety**: Be aware that metadata may be accessed from multiple threads

Troubleshooting

Common Issues

ⓒ Copyright 2026 DEEPX All Rights Reserved.

Segmentation faults when accessing metadata:

- Verify that copy and release functions are properly implemented
- Check that you're not accessing freed memory
- Ensure deep copying of all pointer data

Memory leaks:

- Verify that your release function frees all allocated memory
- Check that you're not creating circular references
- Use valgrind to identify leak sources

Metadata not found:

- Verify that metadata type matches when retrieving
- Check that metadata was successfully attached
- Ensure you're checking the correct metadata list

Debug Tips

Enable debug logging:

```
// Add debug prints in your functions
g_print("Setting user metadata: type=%u, size=%zu\n",
        meta_type, size);
```

Validate data structures:

```
// Add validation in your copy function
g_return_val_if_fail(src != nullptr, nullptr);
g_return_val_if_fail(((MyData*)src)->magic == MY_DATA_MAGIC, nullptr);
```

Memory tracking:

```
// Track allocations for debugging
static gint allocation_count = 0;

static void my_data_free(gpointer data) {
    allocation_count--;
    g_print("Free: %d allocations remaining\n", allocation_count);
    // ... actual cleanup ...
}
```


© Copyright 2026 DEEPX All Rights Reserved.

This completes the User Metadata Guide. The system is designed to be simple yet flexible, providing the essential functionality needed for most use cases while maintaining memory safety and performance.

ⓒ Copyright 2026 DEEPX All Rights Reserved.

9.2 Performance Evaluation with GstShark

GstShark is a powerful performance analysis tool for GStreamer pipelines that provides comprehensive profiling capabilities including CPU usage, processing time, frame rate, and bitrate analysis. This appendix describes how to install and use GstShark for evaluating DX-STREAM pipeline performance.

9.2.1 Installation and Environment Setup

Automated Installation Script

DX-STREAM provides an automated installation script to simplify the setup of GstShark and its dependencies. Execute the following command from the DX-STREAM project root directory:

```
./install_gstshark.sh
```

This automated script performs the following actions required for a system-wide GstShark installation:

- **Dependency Installation**: Installs necessary system packages (graphviz, libgraphviz-dev, octave, epstool, babeltrace).
- **Source Acquisition**: Clones the GstShark repository from GitHub into the `util/` directory of the project root.
- **Build and Configuration**: Compiles and installs GstShark with the appropriate GStreamer configurations.
- **PATH Update**: Adds the GstShark visualization and analysis tools (`gstshark-plot`) to your user `PATH` by appending an entry to `~/.bashrc` .

> **NOTE**
>
> The installation requires sudo privileges for system-wide installation of GstShark libraries and tools.

GstShark Environment Variables

© Copyright 2026 DEEPX All Rights Reserved.

GstShark uses environment variables to control which performance data is collected and where the results are stored.

- `GST_TRACERS` : Specifies which GstShark tracers to use (semicolon-separated)
- `GST_SHARK_LOCATION` : Directory where GstShark results will be saved
- `GST_DEBUG` : Controls debug output level for tracer information

Available Tracers

GstShark provides the following tracers for performance analysis:

Tracer	Description	Key Metric Monitored
`cpuusage`	Measures CPU usage consumed per element.	Resource Utilization
`proctime`	Measures the processing time (latency) spent inside each element.	Element Latency
`framerate`	Analyzes the actual frame rate achieved by the pipeline.	Throughput
`bitrate`	Monitors the data throughput (bitrate) of the stream.	Data Flow Rate
`interlatency`	Measures latency between connected elements.	Inter-Element Delays
`queuelevel`	Monitors Queue buffer levels to identify bottlenecks and backpressure.	Bottleneck Detection
`buffer`	Provides detailed analysis of buffer flow and timestamping.	Data Consistency

9.2.2 Pipeline Performance Measurement

Basic Performance Evaluation

ⓒ Copyright 2026 DEEPX All Rights Reserved.

This command demonstrates how to use GstShark to analyze a standard H.264 video processing pipeline, displaying the output directly to the console:

```
GST_DEBUG=GST_TRACER:7 GST_TRACERS="cpuusage;proctime;framerate;bitrate" \
    gst-launch-1.0 filesrc location=./dx_stream/samples/videos/boat.mp4 ! \
    qtdemux ! h264parse ! avdec_h264 ! videoconvert ! fakesink
```

DX-STREAM Pipeline Analysis

To analyze the performance contribution of the NPU-accelerated elements (dxpreprocess, dxinfer, dxpostprocess), include them in the pipeline and use relevant tracers like queuelevel to detect bottlenecks around the NPU element. The `queuelevel` tracer measures only `queue` elements, so place a `queue` between the processing stages as shown below.

```
GST_DEBUG="GST_TRACER:7" GST_TRACERS="cpuusage;proctime;framerate;queuelevel" \
    gst-launch-1.0 filesrc location=./dx_stream/samples/videos/boat.mp4 ! \
    qtdemux ! h264parse ! avdec_h264 ! queue max-size-buffers=1 ! \
    dxpreprocess config-file-path=./dx_stream/configs/YoloV5S_PPU/preprocess_config.json ! queue
max-size-buffers=1 ! \
    dxinfer config-file-path=./dx_stream/configs/YoloV5S_PPU/inference_config.json ! queue max-
size-buffers=1 ! \
    dxpostprocess config-file-path=./dx_stream/configs/YoloV5S_PPU/postprocess_config.json !
queue max-size-buffers=1 ! \
    dxosd ! fakesink
```

Result Storage & Graphical Report Generation

For persistent storage and graphical generation, specify the **GST_SHARK_LOCATION** environment variable.

```
# Create result directory
mkdir -p /tmp/gst-shark-results

# Run analysis with result storage
GST_DEBUG="GST_TRACER:7" \
GST_TRACERS="cpuusage;proctime;framerate;bitrate" \
GST_SHARK_LOCATION="/tmp/gst-shark-results" \
    gst-launch-1.0 filesrc location=./dx_stream/samples/videos/boat.mp4 ! \
    qtdemux ! h264parse ! avdec_h264 ! videoconvert ! fakesink

# View results
ls -la /tmp/gst-shark-results/

# Generate performance graphs (if graphics tools are installed)
gstshark-plot /tmp/gst-shark-results/ -s pdf
```


© Copyright 2026 DEEPX All Rights Reserved.

9.2.3 Result Analysis and Performance Optimization Guide

Interpreting Performance Metrics

GstShark provides real-time performance information through console output. Key metrics to monitor:

- **Processing Time**: Time spent in each element
- **CPU Usage**: CPU utilization per element
- **Frame Rate**: Actual vs expected frame rates
- **Queue Levels**: Buffer queue status for bottleneck detection

Identifying Bottlenecks

GstShark tracers help pinpoint the exact location and nature of performance bottlenecks within the GStreamer pipeline.

Tracer/Metric	Bottleneck Indication	Description
High Processing Time	Slow element processing (e.g., complex pre-processing or slow NPU execution).	Look for elements with consistently high `proctime` values.
CPU Hotspots	Host CPU is overloaded by a specific element.	Monitor `cpuusage` to identify CPU-intensive elements (e.g., software video decoding).
Queue Overflow	Backpressure issue where a producer is faster than the consumer.	Check `queuelevel` for buffer overflow issues, indicating a slow consumer.
Frame Drops	Pipeline cannot maintain the required throughput.	Compare expected vs. actual `framerate` to quantify the performance deficit.

Common Optimization Strategies

ⓒ Copyright 2026 DEEPX All Rights Reserved.

Applying these strategies can alleviate identified bottlenecks:

- **Element Configuration:** Adjust element-specific parameters, such as the inference `interval` of DxPreprocess or interpolation methods in video conversion elements.
- **Buffer Management:** Optimize queue sizes and buffer pools to balance latency and throughput. Larger queues reduce frame drops but increase latency.
- **Threading:** Utilize multi-threaded elements, where available, to leverage multiple CPU cores for parallel processing.
- **Hardware Acceleration:** Crucially, ensure hardware acceleration (e.g., GPU) available on your platform is enabled for supported non-NPU elements (e.g., video conversion) to offload the host CPU.

9.2.4 Troubleshooting and Best Practices

This section addresses common issues encountered when using GstShark for performance analysis.

Common Issues

Issue	Cause	Resolution
No Tracer Output	GstShark is running but not logging data correctly.	Ensure the `GST_SHARK_LOCATION` environment variable is set and the specified directory exists.
Missing Graphics Tools	Unable to run analysis commands like `gstshark-plot`.	Re-run the installation script (`./install_gstshark.sh`) to ensure all graphical components and dependencies are installed.
Permission Errors	Cannot save results to the specified output directory.	Check write permissions for the result directory (e.g., use `sudo` or change permissions with `chmod`).

GStreamer Registry Issues

© Copyright 2026 DEEPX All Rights Reserved.

If the GstShark tracers (e.g., cpuusage, proctime) are not recognized by GStreamer, the plugin registry cache is likely outdated. Follow these steps to force GStreamer to rescan and register the tracers.

- Clear GStreamer Registry Cache

  This step removes the old, potentially corrupt, or incomplete cache files.

  ```
  rm -rf ~/.cache/gstreamer-1.0/registry.*.bin
  ```

- Regenerate Registry

  Running the inspection tool forces GStreamer to scan all plugin paths and create a new, fresh registry file that includes GstShark.

  ```
  gst-inspect-1.0 > /dev/null 2>&1
  ```

- Verify GstShark Installation

  Check if the GstShark elements are now successfully recognized by the system. If successful, you will see a list of tracers printed in your terminal.

  ```
  gst-inspect-1.0 | grep shark
  ```

Best Practices for GstShark Analysis

- **Baseline Measurement:** Always establish baseline performance before optimization
- **Controlled Environment:** Run tests in consistent system conditions
- **Multiple Iterations:** Average results across multiple test runs
- **Resource Monitoring:** Monitor system resources (CPU, memory, GPU) during testing
- **Documentation:** Document test configurations and results for reproducibility

ⓒ Copyright 2026 DEEPX All Rights Reserved.

9.3 Building DX-STREAM on Orange Pi 5 Plus

This guide provides comprehensive instructions for building and running DX-STREAM on the Orange Pi 5 Plus single-board computer with **DEEPX DX-M1 NPU** for AI accelerator.

The Orange Pi 5 Plus is a powerful ARM-based development board designed for high-performance edge computing. When paired with the DEEPX DX-M1 NPU, it provides significant acceleration for AI and computer vision applications. This guide covers the complete setup process from SD card preparation to running DX-STREAM applications.

DX-STREAM Building Process

The overall construction and validation process is broken down into four sequential steps.

- Step 1: SD Card Setup
- Step 2: DX-RT NPU Driver Installation
- Step 3: Build DX-STREAM
- Step 4: Running Demo Applications

9.3.1 Prerequisites

This section outlines the essential hardware and software components required for building and running DX-STREAM on the Orange Pi 5 Plus with the DEEPX DX-M1 NPU. Ensuring these requirements is crucial for a successful installation.

Hardware Requirements

- Orange Pi 5 Plus development board
- DEEPX DX-M1 NPU module (M.2 form factor)
- MicroSD card (32GB or larger, Class 10 recommended)
- Power supply (5V/4A minimum)
- Host Linux machine for SD card preparation

Software Requirements

- 7-Zip or similar extraction utility
- SD card flashing tool or `dd` command access

ⓒ Copyright 2026 DEEPX All Rights Reserved.

9.3.2 [Step 1] SD Card Setup

Prepare the base Ubuntu OS environment.

[Step 1.1] Download & Extract Ubuntu Image

Download the official Orange Pi 5 Plus Ubuntu 22.04 image:

- **Download Link**: Orange Pi 5 Plus Ubuntu 22.04 Image
- **Image Version**: Ubuntu 22.04 Jammy Desktop XFCE with Linux kernel 6.1.43
- **File Size**: Approximately 4GB (compressed)

Extract the downloaded 7z archive:

```
7z x Orangepi5plus_1.2.0_ubuntu_jammy_desktop_xfce_linux6.1.43.7z
```

NOTE

If 7-Zip is not installed, install it using:

```
# Ubuntu/Debian
sudo apt install p7zip-full

# CentOS/RHEL
sudo yum install p7zip
```

[Step 1.2] Flashing the SD Card

WARNING

Replace `/dev/sda` with your actual SD card device. Use `lsblk` or `fdisk -l` to identify the correct device.

```
sudo umount /dev/sda

sudo dd if=./Orangepi5plus_1.2.0_ubuntu_jammy_desktop_xfce_linux6.1.43.img of=/dev/sda bs=4M status=progress

sync
```


© Copyright 2026 DEEPX All Rights Reserved.

Alternative Methods

- **Raspberry Pi Imager**: Cross-platform GUI tool
- **Balena Etcher**: User-friendly flashing utility
- **Win32DiskImager**: Windows-specific tool

[Step 1.3] Initial Boot & Credentials

- (1) Insert the SD card into the Orange Pi 5 Plus
- (2) Connect power supply and peripherals
- (3) Power on the device

Default Credentials (if applicable)

- Username: `orangepi`
- Password: `orangepi`

9.3.3 [Step 2] DX-RT NPU Driver & Runtime Installation

Install the necessary drivers and runtime for the DEEPX NPU accelerator.

All following steps should be performed on the Orange Pi 5 Plus itself after successful boot.

[Step 2.1] Updating Linux Kernel Headers

Install the required kernel headers for driver compilation:

```
cd /opt
sudo apt install ./linux-headers-current-rockchip-rk3588_1.2.0_arm64.deb
```

> **NOTE**
>
> The kernel headers package should be pre-installed in the Orange Pi image. If missing, contact DEEPX support.

(Optional) Configure Git

If you encounter SSL certificate issues with GitHub:

```
git config --global http.sslVerify false
```


ⓒ Copyright 2026 DEEPX All Rights Reserved.

> **NOTE**
>
> This disables SSL verification globally. For production use, consider configuring proper certificates instead.

[Step 2.2] Compiling NPU Linux Driver

```
cd ~
mkdir -p dxnn && cd dxnn
git clone https://github.com/DEEPX-AI/dx_rt_npu_linux_driver.git
cd dx_rt_npu_linux_driver/modules
./build.sh
sudo ./build.sh -c install
```

What this does:

- Downloads the NPU kernel driver source code
- Compiles the driver for the current kernel
- Installs the driver modules and creates device nodes

[Step 2.3] Installing DX-RT Runtime

```
cd ~/dxnn
git clone https://github.com/DEEPX-AI/dx_rt.git
cd dx_rt
./install.sh --all
./build.sh
```

Build Options:

- `--all` : Installs all dependencies and development tools

[Step 2.4] Verifying Installation (dxcli)

Test the NPU driver and runtime installation:

```
dxcli -s
```


ⓒ Copyright 2026 DEEPX All Rights Reserved.

> **NOTE**
>
> `dxcli` is the standard CLI tool. Legacy name `dxrt-cli` remains available as a backward-compatible alias.

Expected Output:

```
DXRT <dxrt-version>
=============================================================
 * Device 0: M1, Accelerator type
--------------------  Version  --------------------
 * RT Driver version   : <rt-driver-version>
 * PCIe Driver version : <pcie-driver-version>
-------------------------------------------------------
 * FW version         : <firmware-version>
-------------------- Device Info --------------------
 * Memory : LPDDR5 6000 MHz, 3.92GiB
 * Board  : M.2, Rev 1.5
 * Chip Offset : 0
 * PCIe   : Gen3 X4 [<bus>:<device>:<function>]

NPU 0: voltage 750 mV, clock 1000 MHz, temperature 40'C
NPU 1: voltage 750 mV, clock 1000 MHz, temperature 40'C
NPU 2: voltage 750 mV, clock 1000 MHz, temperature 40'C
=============================================================
```

Quick Diagnostics (If Device is Not Detected)

If the `dxcli -s` command fails to detect the hardware, execute the following diagnostic checks sequentially.

```
# 1. Verify if the DEEPX NPU kernel driver module is loaded into the system
lsmod | grep dx

# 2. Inspect kernel system logs for any DEEPX driver initialization or PCIe handshake errors
dmesg | grep -i deepx
```

9.3.4 [Step 3] Building DX-STREAM Framework

Compile and install the DX-STREAM GStreamer framework and plugins.

[Step 3.1] Installing Build Dependencies

© Copyright 2026 DEEPX All Rights Reserved.

The Orange Pi 5 Plus Ubuntu image includes pre-configured GStreamer packages with Rockchip hardware acceleration support. Install the additional required packages:

```
sudo apt-get update
sudo apt install -y python3-pip ninja-build
sudo pip install meson
sudo apt-get install -y libeigen3-dev libjson-glib-dev librdkafka-dev libmosquitto-dev libyuv-dev librga
-dev
```

Package Descriptions

- `python3-pip`, `ninja-build`, `meson`: Build system components
- `libeigen3-dev`: Linear algebra library for computer vision
- `libjson-glib-dev`: JSON parsing library for GLib
- `librdkafka-dev`: Apache Kafka client library
- `libmosquitto-dev`: MQTT broker client library
- `libyuv-dev`: YUV color space conversion library
- `librga-dev`: Rockchip RGA library. The RGA transform backend of DX-STREAM is built automatically when it is installed

> **NOTE**
>
> DX-STREAM also requires OpenCV 4.2.0 or higher. If the image does not include it, install it with `sudo apt install -y libopencv-dev`.

[Step 3.2] Cloning & Compiling DX-STREAM

```
cd ~/dxnn
git clone https://github.com/DEEPX-AI/dx_stream.git
cd dx_stream
./build.sh
```

Build Process:

- Configures Meson build system
- Compiles all GStreamer plugins
- Builds sample applications and utilities
- Installs plugins to system directories

Build Time: Approximately 10-15 minutes on Orange Pi 5 Plus

© Copyright 2026 DEEPX All Rights Reserved.

[Step 3.3] Verifying GStreamer Plugins

Apply the environment variables that `build.sh` added to `~/.bashrc` (or open a new terminal):

```
source ~/.bashrc
```

Check if DX-STREAM plugins are properly installed:

```
gst-inspect-1.0 | grep dx
```

Expected Output:

```
dxstream:  dxgather: DxGather
dxstream:  dxinfer: DXInfer
dxstream:  dxmsgbroker: DXMsgBroker
dxstream:  dxmsgconv: DXMsgConv
dxstream:  dxosd: DXOsd
dxstream:  dxpostprocess: DXPostprocess
dxstream:  dxpreprocess: DXPreprocess
...
```

9.3.5 [Step 4] Running Demo Applications

Verify the complete system functionality and NPU acceleration.

[Step 4.1] Setting up Model & Video Assets

```
cd ~/dxnn/dx_stream
./setup.sh
```

[Step 4.2] Executing Demo Pipeline

```
./run_demo.sh
```

Available Demo Options:

- Object detection with YOLO models
- Face detection (YOLOv5s_Face)
- Pose estimation
- Real-time video processing

ⓒ Copyright 2026 DEEPX All Rights Reserved.

For detailed demo instructions, refer to the **Installation Guide**.

> **NOTE**
>
> Hardware Acceleration. The Orange Pi 5 Plus image includes optimized drivers for:
>
> - Video Decode/Encode (VPU): Rockchip RK3588 VPU
> - GPU Acceleration: Mali-G610 MP4
> - NPU Acceleration (AI): DEEPX DX-M1 (25 TOPS)
> - 2D Acceleration (RGA): Rockchip RGA, used by the DX-STREAM transform backend (`DxPreprocess`, `DxScale`, `DxConvert`) when `librga-dev` is installed
>
> For maximum performance, ensure your pipelines utilize these specific hardware accelerators.

9.3.6 Troubleshooting & Diagnostics

This section provides guidance for resolving common issues encountered during the DX-STREAM installation or runtime.

Hardware & Plugin Issues (NPU, GStreamer Cache)

NPU Not Detected

Verify the NPU module is correctly recognized and the driver is loaded.

```
# Check PCIe connection
lspci | grep -i deepx

# Verify driver loading
dmesg | grep -i dx_rt
```

GStreamer Plugin Not Found

If DX-STREAM plugins are not registered, clear the GStreamer cache and check again.

```
# Clear plugin cache
rm -rf ~/.cache/gstreamer-1.0/
gst-inspect-1.0 dxstream
```


© Copyright 2026 DEEPX All Rights Reserved.

Build & Performance Issues

Build Failures

If the build process fails, clean the environment and rebuild.

```
# Clean and rebuild
./build.sh --clean
```

Performance Issues

Monitor hardware status if performance is slower than expected.

- Verify NPU Status

```
dxcli -s
```

- Monitor CPU usage

```
htop
```

NOTE

Next Step. After successful installation:

- Explore the **Pipeline Examples**
- Learn about **Writing Custom Applications**
- Review **Element Documentation** for advanced usage

© Copyright 2026 DEEPX All Rights Reserved.

9.4 Building DX-STREAM on Windows

This guide covers building the DX-STREAM GStreamer plugin on Windows (MSVC x64).

The Windows build produces `gstdxstream.dll`, the same GStreamer plugin that Linux builds as `libgstdxstream.so`.

9.4.1 Quick Start

```
install.bat         :: Install all dependencies (one-time)
build.bat           :: Build the plugin + custom libraries + apps
setup.bat           :: Download sample models and videos (one-time)
run_demo.bat        :: Run demo menu
```

Custom Libraries & Apps

```
dx_stream\custom_library\build.bat      :: Build postprocess/msgconv custom libraries
dx_stream\apps\build.bat                :: Build example apps (mqtt, kafka, usermeta)
```

These require the main plugin to be built first (`build.bat`).

9.4.2 Prerequisites and System Setup

Hardware Requirements

- x86_64 Windows 10/11 PC
- DEEPX DX-M1 or compatible NPU (for runtime inference)

Software Requirements

The following must be installed manually before running `install.bat`.

#	Software	Purpose	Download
1	Visual Studio 2022	C++ compiler (MSVC)	visualstudio.microsoft.com
2	Git for Windows	Source control, vcpkg	git-scm.com
3	Python 3.9+		python.org

© Copyright 2026 DEEPX All Rights Reserved.

#	Software	Purpose	Download
		Meson build system host	
4	GStreamer MSVC x86_64 1.16.3+	Multimedia framework	gstreamer.freedesktop.org
5	pkg-config (if not included with GStreamer)	Build dependency discovery	pkgconf releases
6	DEEPX SDK	NPU runtime (dxrt)	DEEPX SDK installer

Visual Studio 2022

Run the installer and select the **"Desktop development with C++"** workload.

Git for Windows

Install with default options. Ensure `git` is in your PATH.

Python

Download the official installer. Check **"Add Python to PATH"** during installation.

GStreamer

The minimum supported version is **1.16.3**; newer versions are allowed without an upper-version limit. Versions **1.16.3** and **1.28.4** have been verified on Windows; this validation history is separate from the minimum-version policy.

Download and install **both** packages from the official site: - **Runtime** (MSVC 64-bit) - **Development** (MSVC 64-bit)

`install.bat` queries the selected SDK's `bin\gst-inspect-1.0.exe --version` and displays its path and version. It stops before dependency installation if the version is below 1.16.3, the query fails or exceeds 10 seconds, or its output cannot be parsed. Errors include the minimum version and download link. `build.bat` uses Meson's existing `gstreamer-1.0 >=1.16.3` dependency constraint to validate the development SDK; no duplicate build-script check is needed. Install matching Runtime and Development versions.

Default install path: `C:\Program Files\gstreamer\1.0\msvc_x86_64`

© Copyright 2026 DEEPX All Rights Reserved.

> **IMPORTANT**
>
> The Development package is required for headers and `.pc` files.

pkg-config

After checking GStreamer, `install.bat` adds its detected `bin` directory to PATH and checks that `pkg-config --version` succeeds. If GStreamer provides a working `pkg-config`, no separate installation is needed.

Otherwise, download and install the x64 MSI from pkgconf releases. Ensure its installation directory is on PATH, open a new terminal, and verify `pkg-config --version` before running `install.bat` again. The script only checks this prerequisite; it does not download or install pkgconf.

DEEPX SDK

Install the DEEPX SDK using the official installer (`DX_SDK_*.exe`). The installer registers `DEEPX_SDK_DIR` as a system environment variable automatically — no manual path configuration is needed.

Default install path: `C:\Program Files\DEEPX\DX_SDK_<version>`

Expected directory structure (set up by the installer)

```
%DEEPX_SDK_DIR%\
    include\
    lib\dxrt.lib       (or lib\x64\dxrt.lib on older SDK versions)
    bin\dxrt.dll
```

Environment Variables Configuration

Most environment variables are configured automatically — no manual `set` commands are needed.

GStreamer root (configured by the GStreamer installer)

Windows build, demo, pipeline, and test scripts check `GSTREAMER_1_0_ROOT_MSVC_X86_64` first, then the legacy `GSTREAMER_1_0_ROOT_X86_64` variable, and finally the default `C:\Program Files\gstreamer\1.0\msvc_x86_64` directory. Candidates must contain `bin\gst-launch-1.0.exe` and `bin\gst-inspect-1.0.exe` for runtime checks or `lib\pkgconfig\gstreamer-1.0.pc` for build checks; invalid candidates are skipped. Batch entrypoints share

© Copyright 2026 DEEPX All Rights Reserved.

`scripts\setup_gstreamer.bat` so they also work in a fresh terminal without inheriting the environment from a previous build. The selected root supplies PATH; development checks also set PKG_CONFIG_PATH. After installing or changing GStreamer, open a new terminal to pick up its environment variables.

For manual `gst-inspect-1.0` or `gst-launch-1.0` commands in CMD, configure the current terminal from the project root first:

```
call scripts\setup_gstreamer.bat runtime
```

Build, demo, and pipeline batch scripts perform this setup automatically.

The build copies required vcpkg runtime DLLs, including OpenSSL 3 DLLs used by `mosquitto.dll`, into `install\bin`. GStreamer's bundled OpenSSL version may differ, so it is not a substitute for those dependencies.

DEEPX_SDK_DIR (auto-configured by DEEPX SDK installer)

The DEEPX SDK installer sets `DEEPX_SDK_DIR` as a system environment variable. All build scripts read this variable automatically to locate the NPU runtime headers and libraries.

If for any reason the variable is missing, set it manually.

```
set DEEPX_SDK_DIR=C:\Program Files\DEEPX\DX_SDK_<version>
```

DEEPX_VNPU_DIR (auto-configured by the DX-H1 V-NPU SDK installer)

Build with `--dxvnpu` requires the SDK CMake package at:

```
%DEEPX_VNPU_DIR%\lib\cmake\dxvnpu\dxvnpuConfig.cmake
```

The installer also provides the VNPU runtime DLL directory. `build.bat` and `run_demo.bat` add `%DEEPX_VNPU_DIR%\bin` to their process `PATH`.

9.4.3 Installation and Dependency Provisioning

Standard Installation (install.bat)

After the prerequisites and environment variables are set.

```
install.bat
```


© Copyright 2026 DEEPX All Rights Reserved.

This automatically:

- Step 1. Verifies prerequisites, checking `pkg-config` immediately after GStreamer
- Step 2. Installs `meson` and `ninja` via pip
- Step 3. Clones and bootstraps vcpkg (if not present)
- Step 4. Installs C++ libraries via vcpkg: eigen3, opencv4, libyuv, mosquitto, librdkafka
- Step 5. Generates `opencv4.pc` (required for meson to find OpenCV)
- Step 6. Validates DEEPX_SDK_DIR path

On success, the script updates the calling CMD's PATH with the detected GStreamer `bin` directory. If `pkg-config` is missing or cannot run, prerequisite checking fails with manual installation instructions before any automatic installation.

Air-Gapped Network (No Internet)

On an internet-connected machine.

```
C:\vcpkg\vcpkg install eigen3:x64-windows opencv4:x64-windows libyuv:x64-windows ^
                  mosquitto:x64-windows librdkafka:x64-windows
C:\vcpkg\vcpkg export eigen3:x64-windows opencv4:x64-windows libyuv:x64-windows ^
                  mosquitto:x64-windows librdkafka:x64-windows --zip
```

Copy the resulting zip to the air-gapped machine, extract `installed/x64-windows/` to:

```
dx_stream\vcpkg_installed\x64-windows\
```

Then run `build.bat` directly (vcpkg_installed already present).

9.4.4 Compilation and Ecosystem Building

Core Framework Plugin Compilation (build.bat)

Basic Build

```
build.bat
```

Output: `gst-dxstream-plugin\builddir\src\gstdxstream.dll`

Build Options

ⓒ Copyright 2026 DEEPX All Rights Reserved.

```
build.bat --clean            :: Remove builddir and rebuild from scratch
build.bat --type=debug         :: Debug build (symbols, no optimization)
build.bat --dxv3             :: Enable DX V3 transform backend (requires dxdsp; --v3 is an alias)
build.bat --dxvnpu            :: Enable DXVNPU backend (requires DEEPX_VNPU_DIR)
build.bat --rkrga            :: Enable RK RGA transform backend (requires librga)
build.bat --plugin-only        :: Build only the plugin (skip custom libraries, apps, pydxs)
build.bat --uninstall         :: Remove install directory and registered environment variables
build.bat --clean --type=debug :: Combine options
```

Building Custom Libraries and Example Applications

Custom libraries (postprocess, message_convert) and example apps have their own build scripts, separate from the main plugin build.

Custom Libraries

```
dx_stream\custom_library\build.bat            :: Build all
dx_stream\custom_library\build.bat --clean      :: Clean rebuild
dx_stream\custom_library\build.bat --type=debug :: Debug build
```

Builds every subdirectory under `postprocess_library/` and `message_convert_library/` that contains a `meson.build`.

Output DLLs are in each subdirectory's `builddir/`.

Example Apps

```
dx_stream\apps\build.bat            :: Build all
dx_stream\apps\build.bat --clean      :: Clean rebuild
```

Builds `mqtt_sub_example`, `kafka_consume_example`, `usermeta_app`.

Output executables are in each subdirectory's `builddir/`.

> **NOTE**
>
> Both scripts require the main plugin to be built first — they resolve `dependency('gstdxstream')` via the auto-generated `gstdxstream-uninstalled.pc` in the plugin builddir.

9.4.5 Asset Provisioning and Demo Execution

© Copyright 2026 DEEPX All Rights Reserved.

Fetching Demo Assets (setup.bat)

```
setup.bat
```

Downloads sample AI models and test videos required by the demo pipelines. Assets are saved to `dx_stream\samples\`.

For air-gapped environments, manually place model files (`.dxnn`) in `dx_stream\samples\models\` and video files in `dx_stream\samples\videos\`.

Running Interactive Demos (run_demo.bat)

```
run_demo.bat                :: Interactive demo menu
run_demo.bat --internal-rtsp    :: Use internal RTSP server for demo 8
```

The demo menu matches the Linux `run_demo.sh` layout:

#	Demo	Model
0	Object Detection	YOLO26n
1	Object Detection	YoloV5S PPU
2	Face Detection	YOLOv5s_Face
3	Pose Estimation	YOLO26n_Pose
4	Instance Segmentation	YOLO26n-Seg
5	Depth Estimation	YOLO26n_Depth
6	Multi-Object Tracking	YoloV5S + OC_SORT
7	Multi-Stream (4ch)	Compositor Grid
8	Multi-Channel (RTSP)	dxinputselector
9	Secondary Mode	Multi-Model Cascade

Pipeline scripts are located in `dx_stream\pipelines\windows\`. Each script is self-contained and can be run independently if `DXSTREAM_ROOT` is set:

ⓒ Copyright 2026 DEEPX All Rights Reserved.

```
set DXSTREAM_ROOT=C:\path\to\dx_stream
dx_stream\pipelines\windows\object_detection_yolo26n.bat
dx_stream\pipelines\windows\rtsp.bat --internal-rtsp
```

Full Build + Demo Workflow

```
build.bat --clean
setup.bat                    :: Download models & videos (first time only)
run_demo.bat
```

9.4.6 Production Target Deployment

Target Machine Prerequisites

The target machine must have:

- **GStreamer Runtime** (MSVC 64-bit) installed — provides core GStreamer DLLs
- **VC++ Redistributable 2015-2022** (x64) — provides MSVCP140.dll, VCRUNTIME140.dll

Source Build Deployment Routine

DX-STREAM on Windows uses source-build distribution. Clone the repository, build, and run on the target machine:

```
git clone <repo-url>
cd dx_stream
install.bat
build.bat
setup.bat
run_demo.bat
```

> **NOTE**
>
> `DEEPX_SDK_DIR` is set automatically by the DEEPX SDK installer. Install the DEEPX SDK before running `build.bat`

© Copyright 2026 DEEPX All Rights Reserved.

`build.bat` automatically:

- Builds the plugin, custom libraries, apps, and Python bindings (pydxs)
- Collects all DLLs into `install\bin\`
- Prepends this checkout's entries to User `GST_PLUGIN_PATH` and `Path`, preserving unrelated entries

Required Environment Variables & Verification

`build.bat` merges the following entries into the persistent User environment. Existing entries for other plugins, SDKs, and checkouts are retained; repeated builds do not duplicate this checkout's entries. The commands below illustrate the resulting search paths, not commands to overwrite your persistent variables:

```
set GST_PLUGIN_PATH=%PROJECT_ROOT%\install\lib\gstreamer-1.0;%GST_PLUGIN_PATH%
set PATH=%PROJECT_ROOT%\install\bin;%PROJECT_ROOT%\install\share\gstdxstream\lib;
%PROJECT_ROOT%\install\share\gstdxstream\bin;%PROJECT_ROOT%
\install\lib\gstreamer-1.0;%PATH%
```

Variable	Purpose
`GST_PLUGIN_PATH`	Tells GStreamer where to find `gstdxstream.dll`
`PATH`	Makes dependency DLLs (opencv, dxrt, etc.) discoverable at runtime

Verification

After building, open a new `cmd.exe` and run:

```
gst-inspect-1.0 dxpreprocess
```

If this prints element details, DX-STREAM is correctly installed.

Uninstall

```
build.bat --uninstall
```

This removes this checkout's install directory, build directories, dedicated `venv-dx_stream`, and Python packaging artifacts. Only this checkout's entries are removed from User

ⓒ Copyright 2026 DEEPX All Rights Reserved.

`GST_PLUGIN_PATH` and `Path`; other checkouts, shared SDK variables, and Machine environment variables are left unchanged.

Exit code `0` means cleanup completed. If files remain locked or environment updates fail, the command reports an error and returns `1`, rather than reporting success. Close applications using DX-Stream, check the reported permissions or environment error, and retry. Repeated uninstall is safe when artifacts and entries are already absent. Restart your terminal or IDE after environment changes. Environment registration failures also cause `build.bat` to return `1`, even if the build artifacts were installed successfully.

Critical DLL Naming Constraints

The plugin DLL **must** be named `gstdxstream.dll` (no version suffix). GStreamer derives the plugin entry point symbol from the filename:

- `gstdxstream.dll` → looks for `gst_plugin_dxstream_get_desc` ✓
- `gstdxstream-0.dll` → looks for `gst_plugin_dxstream_0_get_desc` ✗ (symbol not found)

The build system already handles this — `soversion` is only applied on Linux where symlinks resolve the naming. Do not rename the DLL after building.

9.4.7 Technical Reference and Troubleshooting

Dependency Mapping Summary Matrix

Dependency	Source	Method
gstreamer-1.0, gstreamer-video-1.0, json-glib-1.0, zlib	GStreamer installer	`dependency()` (pkg-config)
eigen3, opencv4	vcpkg	`dependency()` (pkg-config)
libyuv	vcpkg	`cc.find_library()`
libmosquitto, rdkafka	vcpkg	`dependency()` (pkg-config)
dxrt, dxdsp	DEEPX internal	`cc.find_library()`
dl (POSIX dynamic linking)	Linux only	Skipped on Windows (`dx_dlfcn.h` shim)

© Copyright 2026 DEEPX All Rights Reserved.

Platform Comparisons: Windows vs. Linux Build

Aspect	Linux	Windows
Dependency install	`install.sh` (apt/source)	`install.bat` (vcpkg)
Compiler	GCC / Clang	MSVC (cl.exe)
Build	`build.sh`	`build.bat`
Plugin DLL name	`libgstdxstream.so` (with .so.0 symlink)	`gstdxstream.dll` (no version suffix)
Dynamic loading	`dlopen` / `dlsym`	`LoadLibrary` / `GetProcAddress` (via `dx_dlfcn.h`)
Plugin scanner	Default	Default
Distribution	Source build	Source build
Demo launcher	`run_demo.sh`	`run_demo.bat`
Pipeline scripts	`dx_stream/pipelines/` (shell scripts)	`dx_stream/pipelines/windows/` (batch files)
Apps / custom_library	`build.sh` (meson install)	`build.bat` (meson install to `install\share\gstdxstream\`); standalone `custom_library\build.bat` and `apps\build.bat` compile only
Python bindings (pydxs)	Built with rpath	Built with Windows DLL loader

Troubleshooting and Diagnostics

GStreamer Registry Cache

GStreamer caches plugin scan results in:

```
%LOCALAPPDATA%\Microsoft\Windows\INetCache\gstreamer-1.0\registry.x86_64-msvc.bin
```

If the plugin was previously blacklisted (e.g., environment not set up correctly at first launch), delete this cache and retry:

© Copyright 2026 DEEPX All Rights Reserved.

```
del "%LOCALAPPDATA%\Microsoft\Windows\INetCache\gstreamer-1.0\registry.x86_64-msvc.bin"
```

DEEPX_SDK_DIR not set

```
[ERROR] DEEPX_SDK_DIR environment variable is not set.
```

Install the DEEPX SDK using the official installer — it sets `DEEPX_SDK_DIR` automatically. If already installed, open a new terminal so the updated system environment is loaded. As a last resort, set it manually:

```
set DEEPX_SDK_DIR=C:\Program Files\DEEPX\DX_SDK_<version>
```

install.bat fails: prerequisites missing

Install the listed software and re-run. `install.bat` will not proceed until all prerequisites are satisfied.

meson setup fails: dependency not found

Run `install.bat` first. If already run, check that `vcpkg_installed\x64-windows\lib\pkgconfig\` contains the expected `.pc` files.

Link error: unresolved external symbol

Verify `vcpkg_installed\x64-windows\lib` contains the required `.lib` files.

MSVC not found

Run from a regular `cmd.exe` prompt (not PowerShell). `build.bat` calls `vcvarsall.bat` internally.

gst-inspect-1.0 dxpreprocess: No such element

Check in order:

- Step 1. `GST_PLUGIN_PATH` points to the folder containing `gstdxstream.dll`
- Step 2. `PATH` includes `install\bin` for dependency DLLs
- Step 3. Delete registry cache: `del "%LOCALAPPDATA%\Microsoft\Windows\INetCache\gstreamer-1.0\registry.x86_64-msvc.bin"`
- Step 4. Verify DLL is named `gstdxstream.dll` (not `gstdxstream-0.dll`)

© Copyright 2026 DEEPX All Rights Reserved.

9.5 Third-Party Sample Model Licenses

9.5.1 Third-Party Sample Model Licenses

Sample models included in DX-STREAM are provided for **evaluation and development purposes only** and are **not licensed for commercial deployment**.

The `.dxnn` model files distributed with DX-STREAM are compiled from third-party open-source models, each of which is subject to its own license terms (e.g., AGPL-3.0, GPL-3.0, non-commercial research licenses, etc.). Most of these licenses impose significant restrictions on commercial use, including but not limited to full source code disclosure obligations or outright prohibition of commercial deployment.

For commercial use, users must either:

- Obtain appropriate licenses directly from the original model providers, or
- Use their own commercially licensed models.

DEEPX is not responsible for any legal issues arising from unauthorized commercial use of these models. Users are solely responsible for ensuring compliance with all applicable licenses.

ⓒ Copyright 2026 DEEPX All Rights Reserved.

10. RELEASE_NOTES

10.1 DX-STREAM v3.2.0 / 2026-10-23

10.1.1 1. Changed

- **Model List Update**: Updated the sample model list and download source to models compiled with DX-COM v2.5.0
  - Removed retired SCRFD500M PPU face-detection and YOLOv5Pose PPU pose-estimation demos, including their Linux/Windows pipeline launchers (update any scripts that call the removed demos)
  - Replaced the retired EfficientNet-Lite0 256x256 model with the supported 224x224 model in Linux/Windows secondary pipelines and regression tests
- **Transform backends**: Moved V3 DSP and RGA kernels to optional hardware backend plugins (`gstdxstream-v3` , `gstdxstream-rga`); the core retains the software path and loads installed hardware backends when available
  - To keep V3 DSP acceleration, build with `./build.sh --dxv3` ; RGA is built automatically when `librga` is installed (or force it with `./build.sh --rkrga`)
  - Without the backend plugin, the software (libyuv) path is used

10.1.2 2. Fixed

- **OC-SORT reliability**: Fixed observation history retention, assignment and association/GIoU handling, momentum weighting, invalid/unmatched predictions, and invalid tracker-config recovery
- **Documentation**: Corrected errors, typos, and missing details

10.1.3 3. Added

- **OpenGL ES transform backend**: Added the optional `gstdxstream-gles` backend plugin
  - Runs crop, resize, letterbox and color conversion on EGL/GLES 3.0 GPUs with zero-copy DMA-BUF import
  - Secondary preprocessing batches each frame's objects into one GPU submission
  - Built automatically on ARM hosts (validated on Raspberry Pi 5) when EGL/GLESv2 are installed; other hosts need `-Dwith_gles=enabled`

ⓒ Copyright 2026 DEEPX All Rights Reserved.

- **New demo pipelines**: `run_demo.sh` now supports the following new examples
  - Added a YOLO26 depth-estimation pipeline with `DXFrameMeta` depth metadata and `dxosd` depth-map rendering
  - Added Raspberry Pi camera pipelines using `libcamerasrc` for YOLO26 detection, pose, segmentation, and depth
- **DX-H1 V-NPU support**: Added hardware-accelerated decode, encode, transform, preprocessing, and inference through `dxvnpudec`, `dxvnpuenc`, and `dxinfer`
  - Build with `./build.sh --dxvnpu` (requires the DX-H1 V-NPU SDK)

10.2 DX-STREAM v3.1.0 / 2026-07-22

10.2.1 1. Changed

- **Multi-Stream Domain**: Introduced `application/x-dxvideoraw` domain caps for unified multi-stream processing — a single chain can now process streams with different resolutions and formats
  - Per-stream sticky events (STREAM_START / CAPS / SEGMENT / EOS) are preserved across the domain so each stream keeps an independent timeline
  - `dxinputselector` and `dxoutputselector` act as explicit domain boundaries; single-stream processing elements can be placed directly in multi-stream pipelines without modification
- **Base class migration**: `dxinputselector` and `dxgather` migrated to `GstAggregator` for standard N:1 multiplexing; `dxvnpuoverlay` migrated to `GstBaseSink` for proper render/preroll lifecycle
- **TransformKernelPool**: Introduced automatic src-format-based kernel selection with libyuv fallback; migrated `dxscale`, `dxconvert`, `dxpreprocess`, `dxmsgconv` to use unified kernel pool
- **InferBackend abstraction**: Refactored `dxinfer` to use Put/Get async pattern with backend property (auto/dxrt/dxvnpu) replacing direct dxrt dependency
- **Model List Update**: Updated the model list to include models compiled with DX-COM v2.4.0

10.2.2 2. Fixed

- **LATENCY reporting**: All processing elements now correctly account for their own processing time in LATENCY queries, stabilizing synchronization and QoS behavior in pipelines that include live sources
- **FLUSH recovery**: Internal queues, threads, and per-stream state are now properly reset on FLUSH, eliminating hangs and errors on seek / replay

ⓒ Copyright 2026 DEEPX All Rights Reserved.

- **Push thread lifecycle**: Improved thread cleanup during EOS / FLUSH / state transitions, resolving single-frame input hangs
- **Caps negotiation**: Pad template caps tightened so incompatible element connections fail immediately at negotiation time instead of producing runtime anomalies
- **Error handling**: `abort()` / uncaught exceptions replaced with `GST_ELEMENT_ERROR` across all elements — missing model files, NPU init failures, and errors in user-provided libraries (`dxpreprocess` / `dxpostprocess` / `dxmsgconv`) now produce proper GStreamer error messages instead of crashing the pipeline

10.2.3 3. Added

- **Windows Build and Runtime Environment**: Full Windows MSVC support including dependency check, build, demo launcher, test suite, Python binding (pydxs), and build guide documentation
- **DEEPX Agent-Driven Development (dx-agent-dev, Beta)**: Describe a pipeline in natural language and an AI agent assembles the GStreamer graph from DEEPX elements, wiring NPU inference into real-time video/stream pipelines end to end
- **Test Suite**: Reorganized test infrastructure under `test/base/{element,metadata,pipeline}/` with 73 new test binaries covering element contracts, domain boundaries, end-to-end pipelines, metadata, and pydxs scenarios
- **DxMsgConv**: Added `include-frame` property for base64 JPEG frame encoding with Kafka/MQTT consumer display support

10.3 DX-STREAM v3.0.1 / 2026-05-06

10.3.1 1. Changed

- None

10.3.2 2. Fixed

- Fixed typos in User Manual
- Fixed uninstall not removing apps build directories and pydxs build cache

10.3.3 3. Added

- Third-party sample model license notice documentation

© Copyright 2026 DEEPX All Rights Reserved.

10.4 DX-STREAM v3.0.0 / 2026-04-10

10.4.1 1. Changed

- **Model Download**: Switched to `dx-modelzoo`-managed models with updated demo lineup **(Breaking)**
  - Per-model download via `model_list.json` + `setup_sample_models.sh` (`jq` required)
  - Demo model upgrades: YOLOv5s → YOLOv26n, YOLOv5Pose → YOLOv26n_Pose, DeepLabV3+MobileNetV2 → YOLOv26n_Seg (not available in modelzoo; replaced)
  - Models not in dx-modelzoo (manually uploaded to S3, accuracy not measured): SCRFD500M PPU, YOLOv5Pose PPU
- **Installation**: Removed `/etc/profile.d/gstdxstream.sh` (no longer requires sudo); env vars now written directly to `~/.bashrc` **(Breaking)** (old profile.d-based setup no longer used)
- **C++ Standard**: Updated to C++14 (using declarations, enum classes) **(Breaking)**
- **Preprocessing Architecture**: Introduced `IVideoTransformKernel` abstraction (RGA / libyuv / V3 DSP) with `VideoTransformFactory` for auto backend selection
- **DxOsd**: Removed unnecessary color conversion, resize, and intermediate memory allocation; added YUV in-place overlay rendering
- **Dependency**: Added `gstreamer1.0-dev` build dependency; install prefix default `/usr/local`

10.4.2 2. Fixed

- NV12 CPU input stride/offset now uses GstVideoInfo instead of RGA heuristic (`dxconvert`/`dxscale`)
- DMA-Buffer zero-copy support in RGA preprocessor and `dxosd` with proper stride calculation
- Orange Pi 5 Plus (RK3588 / Debian 12) display corruption via automatic I420 conversion
- RGA preprocessing build errors on Rockchip SoC
- Software rendering buffer management: stream-specific caching for multi-stream scenarios
- Memory management with smart pointers and improved tensor handling
- Meson install failure in Python venv environment (PYTHONPATH handling and build directory standardization)

10.4.3 3. Added

- **DxScale**: New GstBaseTransform element for hardware-accelerated scaling (NV12/I420/RGB/BGR)

ⓒ Copyright 2026 DEEPX All Rights Reserved.

- **DxConvert**: New GstBaseTransform element for color format conversion (full 4×4 format matrix)
- RGA hardware-accelerated format conversion/scaling (NV12↔RGB↔BGR) with automatic software fallback
- V3 DSP preprocessor with OSD v3 RGB buffer drawing for DEEPX V3 SoC
- YOLOv26 examples
- Auto-detection and cleanup of old installation files from previous major versions
- Debugging guide documentation

10.5 DX-STREAM v2.2.1 / 2026-02-26

10.5.1 1. Changed

- Updated sample models version from DX-COM v2.2.0 to v2.2.1

10.5.2 2. Fixed

- None

10.5.3 3. Added

- None

10.6 DX-STREAM v2.2.0 / 2026-01-16

10.6.1 1. Changed

- Modified `gst-dxpreprocess` to automatically deep copy buffers when `ref_count > 1` (e.g., after tee element)
- **Metadata Architecture**: Separated existing `gst-dxmeta.cpp` / `hpp` into individual metadata files (frame, object, user metadata) and enhanced structures with user metadata lists and count fields
- **Test Framework**: Refactored metadata handling in test files to use dx_get_frame_meta for improved performance and added test plugin installation step in unit test workflow
- Updated OS requirements in the installation guide for Debian

ⓒ Copyright 2026 DEEPX All Rights Reserved.

10.6.2 2. Fixed

- Fixed race condition and segfaults in secondary inference mode when multiple streams access shared buffer metadata
- **API Migration**: Updated object metadata removal to use new API in track function
- **Error Handling**: Enhanced error handling for compilation failures in test scripts

10.6.3 3. Added

- Added `prepare_output_buffer()` override to detect and copy shared buffers
- Added `pydxs` Python binding module for DX-STREAM metadata
- Python bindings for `DXFrameMeta`, `DXObjectMeta`, `DXUserMeta`, and related value types
- Helper APIs to acquire and attach `DXObjectMeta` to frames and to manage user metadata from Python
- A `writable_buffer` context manager to safely ensure buffer writability and create/retrieve `DXFrameMeta` inside GStreamer pad probes
- A new Python sample app `usermeta_app.py` demonstrating creation, attachment, and reading of frame/object/user metadata using `pydxs`
- **User Metadata Interface**: New API allowing developers to attach custom metadata to frames and objects
- **Management Functions**: Comprehensive metadata structures and management functions (create, copy, release)
- **Test Suite**: Dedicated user metadata test suite (`test_usermeta.cpp`) with validation framework
- **Documentation**: User Metadata Guide documentation and README updates

---

10.7 DX-STREAM v2.1.0 / 2025-11-28

10.7.1 1. Changed

- **Model Configuration**: Updated default YOLOv5 from YOLOV5S_3 to YOLOV5S_4 with `models-2_1_0.tar.gz`
- **Video Sink Settings**: Disabled synchronization in secondary mode for improved performance
- **Demo Scripts**: Simplified video input paths in message broker demos
- **Message Conversion**: Simplified message conversion configuration and improved JSON payload structure

ⓒ Copyright 2026 DEEPX All Rights Reserved.

- **Buffer Processing**: Enhanced preprocessing and postprocessing to use direct buffer manipulation for better performance
- **Configuration**: Streamlined msgconv configuration by removing unnecessary sections
- Modified event handling logic in `dxpreprocess`, `dxinfer`, `dxoutputselector` and `dxosd` to align with updates to `dxinputselector`
- Enhanced dependency installation for Debian 12
  - Smart version checking for meson (apt/backports/pip fallback)
  - Handle libdrm version conflicts without breaking hardware acceleration
- Updated secondary mode pipeline for SCRFD500M_PPU_SECOND

10.7.2 2. Fixed

- **Setup Scripts**: Improved error handling and prevented excessive download retry attempts
- **Shutdown Flow**: Improved shutdown signal processing in `dxinfer` element
- **Memory Management**: Better buffer handling in preprocessing and postprocessing pipelines
- **Registry Handling**: Fixed GStreamer registry cache issues with GstShark integration
- Fixed an event processing timing issue in `dxinputselector` that caused compositor pipeline freezes
- Added detection and installation of Rockchip-specific dependencies (librga-dev)
- Updated MQTT configuration to include connection timeout and state management

10.7.3 3. Added

- **PPU Support**: Integrated Post-Processing Unit functionality for YOLOv5s, SCRFD500M, and YOLOv5Pose models
  - NPU-based bounding box decoding and NMS processing to reduce CPU overhead
  - Three new demo options showcasing PPU capabilities
- **Download Reliability**: Enhanced setup scripts with timeout limits and file integrity verification
  - Automatic verification and cleanup of corrupted archives
  - Prevents infinite retry loops that could cause hour-long hangs
- Installation Guide for Orange Pi 5 Plus

ⓒ Copyright 2026 DEEPX All Rights Reserved.

- **Performance Analysis Tools**: Added GstShark integration for comprehensive pipeline performance evaluation
  - Automated installation script `install_gstshark.sh` for easy setup
  - Complete performance evaluation documentation with sample commands
  - Support for CPU usage, processing time, frame rate, and bitrate analysis
- **Preprocessing Features**: Added preprocess skip functionality for conditional processing
- **Build Support**: Added build configuration for v3 architecture

---

10.8 DX-STREAM v2.0.0 / 2025-09-08

10.8.1 1. Changed

- **Code Examples**: The PostProcess examples have been separated and implemented on a per-model basis for clarity
- **DX-RT v3.0.0 Compatibility**: This version has been updated to ensure full compatibility with DX-RT v3.0.0
- **Model Support**: Inference is now restricted to models (DXNN v7) produced by DX-COM v2.0.0 and later versions
- Modified `dxgather` event handling logic
- Removed unnecessary print statements
- Enhanced build script and updated installation documentation
  - Added OS and architecture checks in the build script
  - Updated CPU and OS specifications in the installation documentation for clarity

10.8.2 2. Fixed

- Addressed and alleviated a processing delay issue within the `dxinputselector`
- Corrected a post-processing logic error in the SCRFD model when in secondary inference mode
- Fixed a bug in `dx_rt` that occurred when processing multi-tail models
- Improved error handling for setup scripts

© Copyright 2026 DEEPX All Rights Reserved.

- Added support for X11 video sink on Ubuntu 18.04 across multiple scripts
  - Force X11 video sink on Ubuntu 18.04
  - Improved compatibility across OS versions
  - Updated multiple pipeline scripts
  - Added OS version check for Ubuntu 18.04

10.8.3 3. Added

- Added an uninstall script and enhanced color utility functions
  - Introduced a new `uninstall.sh` script for cleaning up project files and directories

10.9 DX-STREAM v1.7.0 / 2025-07-23

10.9.1 1. Changed

- `dxinfer`: Improved the buffer queue management mechanism. Instead of locking inputs based on queue size within the push thread, the system now adds a `req_id` to the buffer and utilizes a wait function for more efficient processing
- Setup script now runs automatically when a file-not-found error occurs during example execution
- Applied colors and error handling in scripts

10.9.2 2. Fixed

- `dxpreprocess`, `dxosd`: Resolved a video corruption issue that occurred in some streams. The problem was traced to incorrect stride and offset calculations from GstVideoMeta. The calculation now correctly uses GstVideoInfo included in the caps, ensuring stable video rendering

10.9.3 3. Added

- None

ⓒ Copyright 2026 DEEPX All Rights Reserved.