DXNN Streamer (DX-STREAM) User Manual

2026/04/01 (v3.0.0)

DEEPX.ai

Copyright 2026 DEEPX All Rights Reserved.

Table of contents

1. DXNN SDK Overview 4
1.1 DEEPX SDK Architecture 4
1.2 DX-STREAM Architecture 6
1.3 DX-STREAM Key Features 6
1.4 DX-STREAM Model Workflow 8
2. DX-STREAM Installation 9
2.1 System Requirements 9
2.2 Install DX-STREAM 10
2.3 Run DX-STREAM 16
3. Elements 18
3.1 DxPreprocess 18
3.2 DxInfer 25
3.3 DxPostprocess 29
3.4 DxTracker 32
3.5 DxOsd 35
3.6 DxGather 37
3.7 DxInputSelector 39
3.8 DxOutputSelector 41
3.9 DxRate 43
3.10 DxMsgConv 45
3.11 DxMsgBroker 48
3.12 DxScale 50
3.13 DxConvert 53
4. Writing Your Own Application 56
4.1 DX-STREAM Metadata System Overview 57
4.2 Custom Library for Model Inference 60
4.3 Custom Message Convert Library 68
5. Pipeline Example 73
5.1 Single Stream Pipeline 73
5.2 Multi Stream Pipeline 76
5.3 Secondary Inference Pipeline 78

Copyright 2026 DEEPX All Rights Reserved.

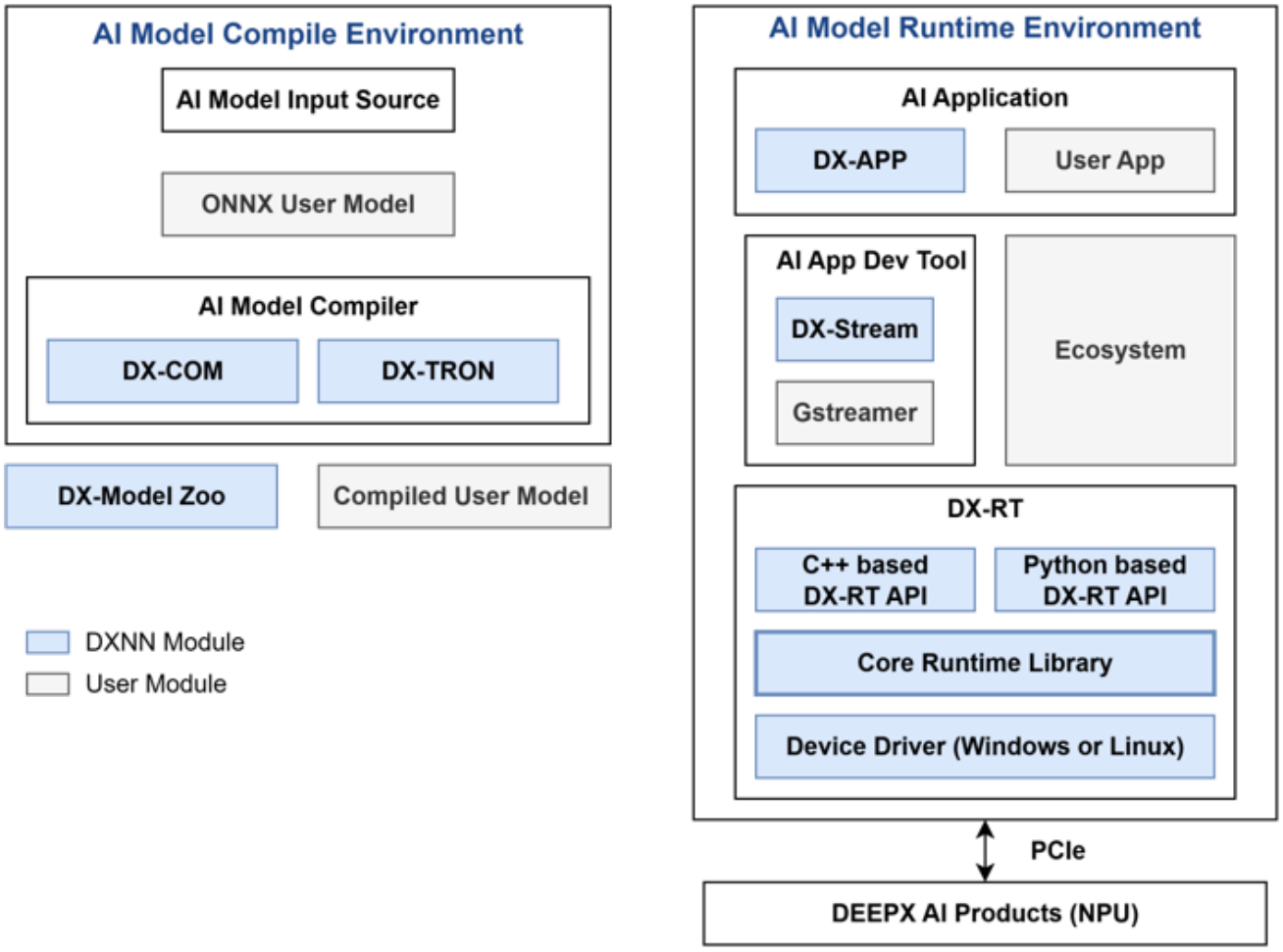
5.4 MsgBroker Pipeline 79
6. Troubleshooting and FAQ 87
6.1 Debug Logging 87
6.2 Rendering Issues 88
6.3 Display Sink Issues on Raspberry Pi 5 89
6.4 Display Sink Issues on Orange Pi 5 Plus with Debian 12 90
6.5 Buffer Delays in Sink Element 93
6.6 Troubleshooting Message Broker Issues 94
6.7 Build Issues with Meson Installation 98
7. Appendix 101
7.1 User Metadata Guide 101
7.2 Building DX-Stream on Orange Pi 5 Plus 109
7.3 Appendix: Performance Evaluation with GstShark 118
8. Change Log 125
8.1 Version 3.0.0 (Apr 2026) 125
8.2 Version 2.2.1 (Feb 2026) 126
8.3 Version 2.2.0 (Dec 2025) 126
8.4 Version 2.1.0 (Nov 2025) 127
8.5 Version 2.0.0 (Aug 2025) 129
8.6 Version 1.7.0 (Jul 2025) 130

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

Copyright 2026 DEEPX All Rights Reserved.

the model structure and weights, while the JSON file defines pre/post-processing settings and compilation parameters. DX-COM provides a fully compiled .dxnn file, optimized for low-latency and high-efficient 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.

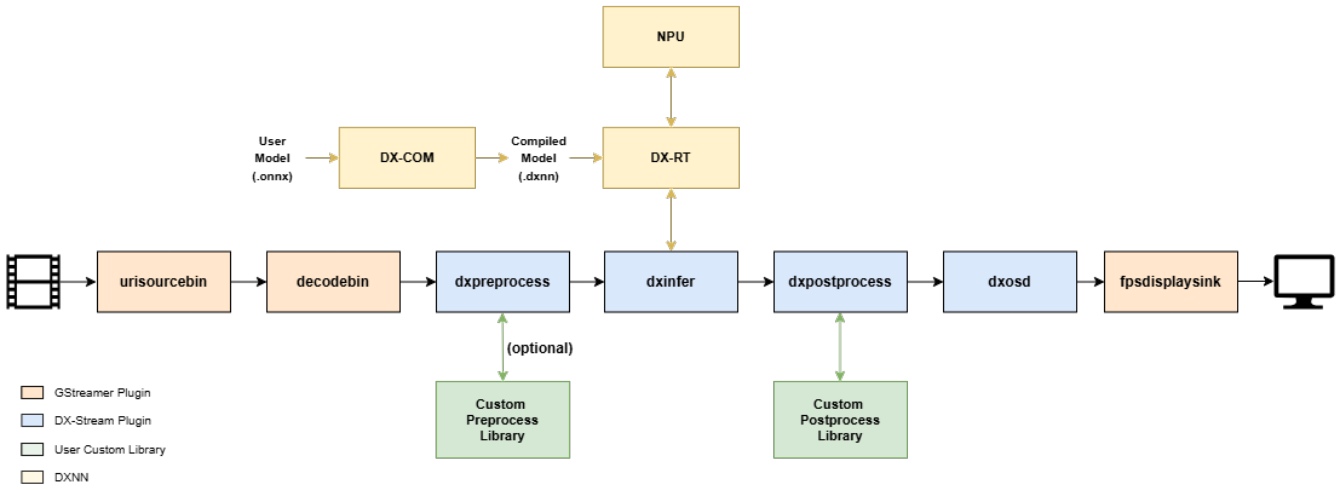
DX-ModelZoo is a curated collection of pre-trained neural network models optimized for DEEPX NPU, designed to simplify AI development for DEEPX users. It includes pre-trained ONNX models, configuration JSON files, and pre-compiled DXNN binaries, allowing developers to rapidly test and deploy applications. 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.

Copyright 2026 DEEPX All Rights Reserved.

1.2 DX-STREAM Architecture



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 for the user's unique model inference.
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 - Selects one stream among multiple inputs based on PTS and forwards it to a single output.
- DxOutputSelector - Distributes a single input stream to multiple outputs based on predefined routing.
- 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** and **DxPostprocess** elements support user-defined custom libraries, allowing developers to integrate application-specific preprocessing and postprocessing 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

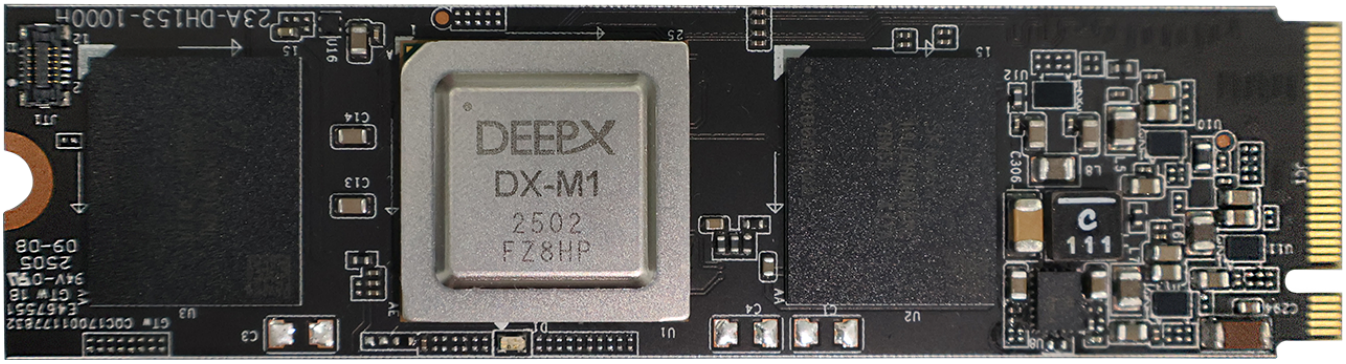
This chapter describes the installation of **DX-STREAM** in both source-based and Docker-based environments.

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 18.04 / 20.04 / 22.04 / 24.04 (x64)
  Debian 12 / Debian 13 (x64)
- **DX-RT must** be installed 3.0.0 or higher available
- 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.

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
$ ./install.sh --wo-opencv

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

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


Copyright 2026 DEEPX All Rights Reserved.

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 & Environment DX-STREAM

(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

# Build for DEEPX V3 Standalone Device
$ ./build.sh --v3

# 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

Copyright 2026 DEEPX All Rights Reserved.

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

- Plugin: `/usr/local/lib/<arch>/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.

(2) How to Apply Environment

The build script **automatically adds** required environment variables to your `~/.bashrc` during installation. After installation completes, choose one of the following to apply the changes :

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.

```
$ 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/x86_64-linux-gnu/gstreamer-1.0:$
{GST_PLUGIN_PATH}"
$ export LD_LIBRARY_PATH="/path/to/install/lib/x86_64-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/x86_64-linux-gnu/gstreamer-1.0/libgstdxstream.so

# For custom installation
$ ls -la /path/to/install/lib/x86_64-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/x86_64-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 use build.sh
$ ./build.sh --uninstall

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


Copyright 2026 DEEPX All Rights Reserved.

The uninstall process automatically removes:

- GStreamer plugin (`libgstdxstream.so`)
- Header files (`include/gstdxstream/`)
- Custom libraries and applications (`share/gstdxstream/`)
- pkg-config file (`gstdxstream.pc`)
- **Environment variables from `~/.bashrc`** (automatic cleanup)

Automatic Cleanup

The uninstall script **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 of quickly running **DX-STREAM**'s sample pipelines

2.3.1 Requirements

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.

Copyright 2026 DEEPX All Rights Reserved.

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 (YOLOv5s)
1: Object Detection (YOLOv5s with PPU)
2: Face Detection (YOLOV5S_Face)
3: Face Detection (SCRFD500M with PPU)
4: Pose Estimation (YOLOV5Pose)
5: Pose Estimation (YOLOV5Pose with PPU)
6: Multi-Object Tracking
7: Semantic Segmentation
8: Multi-Channel Object Detection
9: Multi-Channel RTSP
-: secondary mode
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 (YOLOv5s)`) will be executed automatically.

> **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**.

Copyright 2026 DEEPX All Rights Reserved.

3. Elements

3.1 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.

H/W Acceleration
If the Rockchip RGA (Raster Graphic Accelerator) module is available in the system environment, the input preprocessing step is offloaded from the CPU to the RGA, enabling hardware-accelerated processing. As a result, CPU usage is reduced and processing becomes more efficient.

Hierarchy

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


Copyright 2026 DEEPX All Rights Reserved.

```
        +----GstObject
              +----GstElement
                    +----GstBaseTransform
                          +----GstDxPreprocess
```


Copyright 2026 DEEPX All Rights Reserved.

Properties

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

Copyright 2026 DEEPX All Rights Reserved.

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`
`keep-ratio`	Maintains the original aspect ratio during resizing.	Boolean	`true`
`pad-value`	Padding color value for R, G, B pixels during	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`		Unsigned Integer	`0`

Copyright 2026 DEEPX All Rights Reserved.

Name	Description	Type	Default Value
	Minimum object width for preprocessing in Secondary Mode.		
`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`
`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
}
```


Copyright 2026 DEEPX All Rights Reserved.

NOTE

- For implementing custom preprocess logic, refer to **Chapter. Writing Your Own Application** `“Custom Pre-Process Library Documentation”`.
- All properties can also be configured using a JSON file for enhanced usability and flexibility.

Copyright 2026 DEEPX All Rights Reserved.

3.2 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.

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 `throttling_delay` value. This avoids unnecessary NPU computation in low-framerate pipelines and promotes smooth and consistent streaming.

Copyright 2026 DEEPX All Rights Reserved.

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
```


Copyright 2026 DEEPX All Rights Reserved.

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`	Specifies the ID of the input tensor to be used for **inference**.	Unsigned Integer	`0`
`inference-id`	Specifies the ID of the output tensor to be used for **postprocess**.	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`

Example JSON Configuration

```
{
    "preprocess_id": 1,
    "inference_id": 1,
    "model_path" : "./dx_stream/samples/models/YOLOV5S_1.dxnn"
}
```


Copyright 2026 DEEPX All Rights Reserved.

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.

3.3 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 both built-in and user-defined logic, depending on the configured model type. 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
       +----GstObject
             +----GstElement
```


Copyright 2026 DEEPX All Rights Reserved.

```
                  +----GstBaseTransform
                        +----GstDxPostprocess
```

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 to use for postprocessing.	Integer	`0`
`secondary-mode`	Enables Secondary Mode for processing object regions.	Boolean	`false`
`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

```
{
    "inference_id": 1,
    "library_file_path": "/usr/local/share/gstdxstream/lib/libpostprocess_yolo.so",
```


Copyright 2026 DEEPX All Rights Reserved.

```
    "function_name": "YOLOV5Pose_1"
}
```

NOTE

- All properties can also be configured using a JSON file for enhanced usability and flexibility.
- For implementing custom preprocess logic, refer to **Chapter. Writing Your Own Application** `"Custom Post-Process Library Documentation"`.

Copyright 2026 DEEPX All Rights Reserved.

3.4 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 Object Tracking Algorithms document.

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

Default Behavior

If no config file is provided, **DxTracker** uses the `OC_SORT` algorithm with built-in default values.

Copyright 2026 DEEPX All Rights Reserved.

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
```

Properties

Name	Description	Type	Default Value
`name`	Sets the unique name of the DxTracker element.	String	`"dxtracker0"`
`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"`

Copyright 2026 DEEPX All Rights Reserved.

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 parameters 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.

3.5 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, human pose, and facial landmarks

Hierarchy

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

Properties

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

Copyright 2026 DEEPX All Rights Reserved.

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, poses, or face landmarks, depending on available metadata.

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.

3.6 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 Timestaps (PTS) match.
- Ensures that only synchronized frames are merged together.

Buffer and Metadata Handling

- Assumes that buffer content is identical across all input branches.
- Retains one buffer while others are unreferenced.
- Merges metadata from all branches into the resulting output buffer.

Source Stream Assumption

- Only supports streams split from the same source (e.g., using GstTee).
- **Not** suitable for merging streams from different sources.

Hierarchy

```
GObject
 +----GInitiallyUnowned
       +----GstObject
             +----GstElement
                   +----GstDxGather
```


Copyright 2026 DEEPX All Rights Reserved.

Properties

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

Copyright 2026 DEEPX All Rights Reserved.

3.7 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.

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.

Custom Event Handling

- To properly route sticky events received from multiple sink pads, it first sends a custom routing event before forwarding the sticky event downstream.
- The EOS (End-of-Stream) events received from each sink pad are **not** forwarded downstream directly. instead, they are converted into custom EOS events. This ensures that when EOS is received from one of the N connected input streams, the pipeline does **not** receive a global EOS, and only the EOS for that specific stream is handled.
- Only after receiving EOS events from all input channels does the element forward a global EOS event downstream.

Hierarchy

```
GObject
 +----GInitiallyUnowned
       +----GstObject
             +----GstElement
                   +----GstDxInputSelector
```


Copyright 2026 DEEPX All Rights Reserved.

Properties

Name	Description	Type	Default Value
`name`	Sets the unique name of the DxInputSelector element.	String	`"dxinputselector0"`
`max-queue-size`	Maximum number of buffers to queue per input stream.	Unsigned Integer	`2`

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`.

Downstream Element Restrictions

`GstBaseTransform`-based elements that depend on stable caps negotiation (e.g., **DxScale**, **DxConvert**) must **not** be placed directly downstream of DxInputSelector.
Because DxInputSelector interleaves buffers from multiple streams with potentially different resolutions and color formats, downstream caps would change on every buffer, causing repeated renegotiation, kernel re-creation, and potential caps/buffer mismatch.
Elements designed to handle per-buffer stream variation via `DXFrameMeta` (e.g., **DxPreprocess**, **DxInfer**, **DxPostprocess**, **DxOsd**) are safe to use downstream.

Copyright 2026 DEEPX All Rights Reserved.

3.8 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`.

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.

Custom Event Handling

- `DxOutputSelector` processes sticky events (e.g., caps, segment) using custom routing events generated by `DxInputSelector`. This ensures that each `src` pad receives the correct contextual information.

Hierarchy

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

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. Therefore, if the `stream_id` does **not** correspond to a valid src pad index in `DxOutputSelector`, an error will occur.
- `DxOSD` removes the original buffer and creates a new one for drawing. During this process, `DXFrameMeta` is discarded, which can cause issues if `DxOSD` is placed upstream of `DxOutputSelector`.

Copyright 2026 DEEPX All Rights Reserved.

3.9 DxRate

DxRate is an element that adjusts the framerate of a video stream to match a defined target framerate.
DxRate archives this by dropping or duplicating frames based on the timestamps of incoming buffers.
If the input stream already matches the desired framerate, no frames are altered.
Otherwise, **DxRate** modifies the output buffer timestamps to maintain consistent intervals, ensuring the smooth and accurate playback at the specified framerate.

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.

Throttle QoS Event

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

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

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


Copyright 2026 DEEPX All Rights Reserved.

Properties

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

NOTE

- The `framerate` property is mandatory and **must** be explicitly set for the element to function.

Copyright 2026 DEEPX All Rights Reserved.

3.10 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 Documentation".**

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

- Message format conversion can be customized using a configuration file.

Copyright 2026 DEEPX All Rights Reserved.

Example Output Format

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.

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 converted.	Integer	`1`

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.

3.11 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.

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
```


Copyright 2026 DEEPX All Rights Reserved.

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`

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.

Copyright 2026 DEEPX All Rights Reserved.

3.12 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 (RGA on RK3588) or falls back to the libyuv software backend.

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

- On RK3588 platforms with RGA hardware: attempts RGA-accelerated scaling first.
- Falls back to libyuv software scaling if the hardware backend does not support the source format.

Hierarchy

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

Pad Templates

Sink (input)

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

Copyright 2026 DEEPX All Rights Reserved.

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 ...
```

NOTE

- `dxscale` does **not** perform color conversion. Input and output must be the same format.
- For color conversion, use **DxConvert**.
- Both `width` and `height` must be set together. Setting only one is not supported.

Copyright 2026 DEEPX All Rights Reserved.

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.

3.13 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 (RGA on RK3588) or falls back to the libyuv software backend.

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

- On RK3588 platforms with RGA hardware: attempts RGA-accelerated conversion first (currently supports NV12→RGB and NV12→BGR).
- Falls back to libyuv software conversion if the hardware backend does not support the source format or format combination.

Hierarchy

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


Copyright 2026 DEEPX All Rights Reserved.

Pad Templates

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 BGR before inference:

```
gst-launch-1.0 \
  filesrc location=input.mp4 ! decodebin ! \
  video/x-raw,format=I420 ! \
  dxconvert ! \
  video/x-raw,format=BGR ! \
  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**.
- On RK3588, RGA hardware acceleration is available for NV12→RGB and NV12→BGR conversions. Other combinations 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 **Differences in Post-Processing Logic Based on Inference Mode**

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.

API Migration Notes:

- The `frame_meta->_buf` member has been removed to eliminate circular references.
- All 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.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**.

Copyright 2026 DEEPX All Rights Reserved.

4.1 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.

4.1.1 **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

4.1.2 **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;

    // 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;
```


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
};
```

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;

    // 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.
- `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.

Copyright 2026 DEEPX All Rights Reserved.

4.1.3 **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);

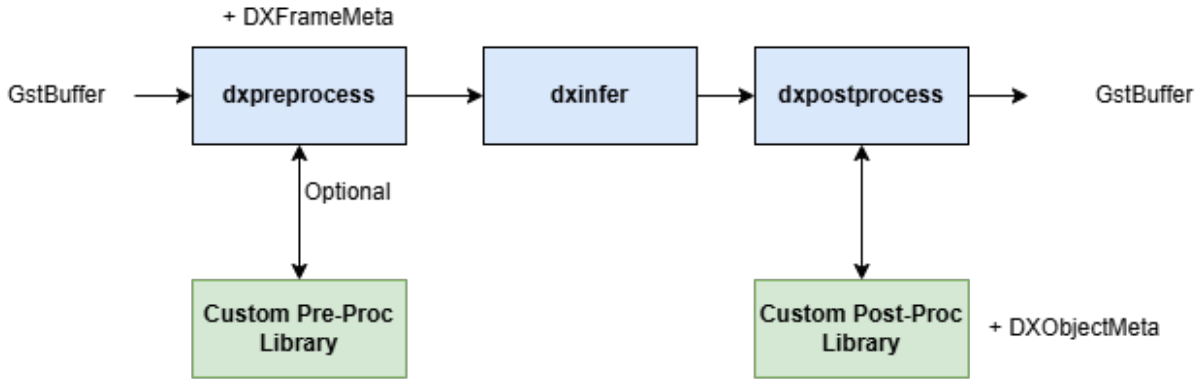
// Data management with required safety functions
gboolean dx_user_meta_set_data(DXUserMeta *user_meta,
                               gpointer data,
                               gsize size,
                               DXUserMetaType meta_type,
                               GDestroyNotify release_func,    // 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);
```


Copyright 2026 DEEPX All Rights Reserved.

4.2 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 elements 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.

DxPostprocess

- Receives the input tensor created by `dxpreprocess`.
- Performs inference using the `dxinfer` element (**DX-RT**).
- Access the output tensor from `dxinfer` and 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`.

4.2.1 **DX-STREAM Metadata Architecture**

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

Copyright 2026 DEEPX All Rights Reserved.

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

Using User Meta System

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 {
    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;
```


Copyright 2026 DEEPX All Rights Reserved.

```
    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);
```

Adding Custom Metadata to Object:

```
// Define object-specific data
typedef struct {
    gint feature_count;
    gfloat *features;
    gchar *feature_name;
} MyObjectFeature;

// Copy function for object data
static gpointer my_object_feature_copy(gconstpointer src) {
    const MyObjectFeature *src_data = (const MyObjectFeature *)src;
    MyObjectFeature *dst_data = g_new0(MyObjectFeature, 1);
    dst_data->feature_count = src_data->feature_count;
    dst_data->features = g_new(gfloat, src_data->feature_count);
    memcpy(dst_data->features, src_data->features,
           src_data->feature_count * sizeof(gfloat));
    dst_data->feature_name = g_strdup(src_data->feature_name);
    return dst_data;
}

// Cleanup function for object data
static void my_object_feature_free(gpointer data) {
    MyObjectFeature *obj_data = (MyObjectFeature *)data;
    g_free(obj_data->features);
```


Copyright 2026 DEEPX All Rights Reserved.

```
    g_free(obj_data->feature_name);
    g_free(obj_data);
}

// Create user meta for object
DXUserMeta *obj_user_meta = dx_acquire_user_meta_from_pool();

MyObjectFeature *feature_data = g_new0(MyObjectFeature, 1);
feature_data->feature_count = 128;
feature_data->features = g_new(gfloat, 128);
// ... populate features array ...
feature_data->feature_name = g_strdup("resnet_features");

dx_user_meta_set_data(obj_user_meta,
                      feature_data,
                      sizeof(MyObjectFeature),
                      DXUserMetaType::DX_USER_META_OBJECT,
                      my_object_feature_free,      // Required cleanup function
                      my_object_feature_copy);     // Required copy function

// Add to object
dx_add_user_meta_to_obj(obj_meta, obj_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);
    }
}

// Get all object user metadata
auto obj_metas = dx_get_object_user_metas(obj_meta);
for (auto user_meta : *obj_metas) {
    // Check if this is object-type metadata
    if (user_meta->user_meta_type == DXUserMetaType::DX_USER_META_OBJECT) {
        MyObjectFeature *feature = (MyObjectFeature *)user_meta->user_meta_data;
        g_print("Object feature: %s with %d dimensions\n",
                feature->feature_name, feature->feature_count);
    }
}
```


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

4.2.2 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;
}
```

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. Therefore, users **must not** free or reallocate this memory.

Library Integration

To build the custom object 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: plugins_install_dir + '/lib'
)
```

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

```
{
    "library_file_path": "./install/gstreamer-1.0/lib/libcustompreproc.so",
    "function_name": "CustomPreprocessFunc"
}
```

4.2.3 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.

```
$ parse_model -m YOLOv7.dxnn

Example output:
```


Copyright 2026 DEEPX All Rights Reserved.

```
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

```cpp
extern "C" void YOLOV7(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);
}
```

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)

Copyright 2026 DEEPX All Rights Reserved.

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_yolov5s', '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')
opencv_dep = dependency('opencv4', required: true)

yolo_postprocess_lib = shared_library('postprocess_yolo',
    'postprocess.cpp',
    dependencies: [opencv_dep, 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": "./install/gstreamer-1.0/lib/libyolo_postprocess.so",
    "function_name": "yolo_post_process"
}
```

4.2.4 Differences in Post-Processing Logic Based on Inference Mode

Primary Mode

- Inference is performed on the entire frame.
- Postprocessing is responsible for creating new objects (`DXObjectMeta`) based on the model's output.
- 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 metadata.
- 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 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**: pixel-level classification maps with height, width, and data
- **Pose Estimation**: 17 keypoints with coordinates (kx, ky) and confidence scores (ks)
- **Face Detection**: landmarks, face bounding box, confidence, and face feature vectors

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

Implementation Example

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

```
#include "dx_msgconvl_priv.hpp"

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


Copyright 2026 DEEPX All Rights Reserved.

```
    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.

```
#### **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:**
```


Copyright 2026 DEEPX All Rights Reserved.

```cpp
// 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,
        "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": {
```


Copyright 2026 DEEPX All Rights Reserved.

```
          "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]
        }
      }
    }
  ]
}
```

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

```
#### **Library Integration**

Build the custom message convert library with proper dependencies:
```


Copyright 2026 DEEPX All Rights Reserved.

```meson
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', required: true)

custom_msgconv_lib = shared_library('dx_msgconvl',
    ['dx_msgconvl.cpp', 'dx_msgconvl_priv.cpp'],
    dependencies: [gst_dep, dx_stream_dep, json_glib_dep],
    include_directories: [include_directories('.')],
    install: true,
    install_dir: '/opt/dx_stream/msgconv/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

Usage in Pipeline:

```
dxmsgconv library-file-path=/opt/dx_stream/msgconv/lib/libdx_msgconvl.so
```

NOTE

The `config-file-path` property is no longer required as configuration parsing has been removed from the library implementation.

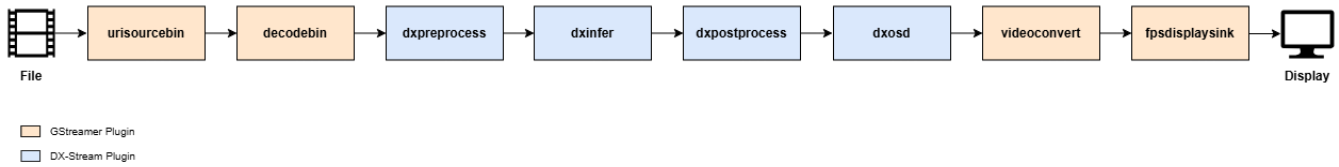
Copyright 2026 DEEPX All Rights Reserved.

5. Pipeline Example

5.1 Single Stream Pipeline

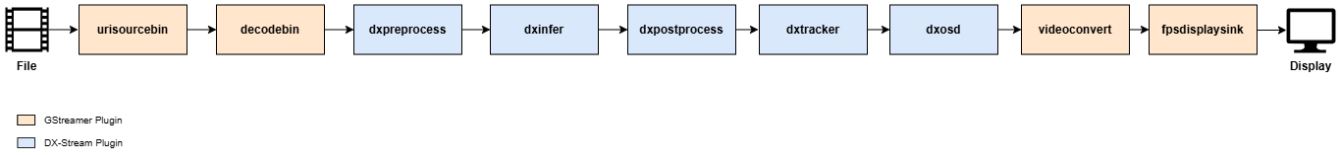
This section describes model inference for a single video stream.
Although the example pipeline performs inference using the YOLOv5s model, it can be easily adapted to various vision tasks with different models. The pipeline below reads video frames from a file and performs a pre-infer-post process tailored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

Copyright 2026 DEEPX All Rights Reserved.

Explanation

Element Descriptions

- `urisourcebin` : Specifies the input video file. The `uri` property **must** be set to the file path of the video you wish to process.
- `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**: Ppixel-wise semantic 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 synchronously because it does **not** include `queue` elements. Each element waits for the previous one to finish processing before continuing. This simplifies data flow but may affect performance on multi-core systems.

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`.

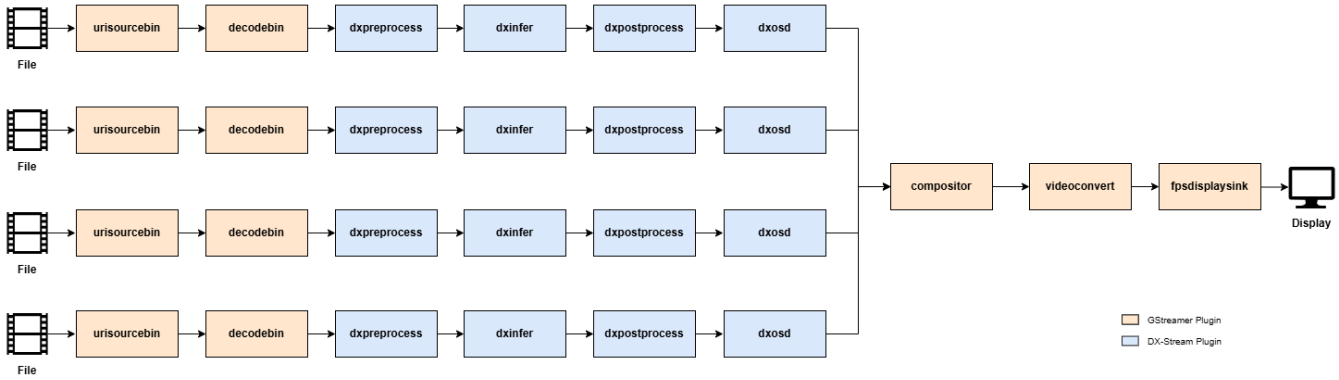
Copyright 2026 DEEPX All Rights Reserved.

5.2 Multi Stream Pipeline

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

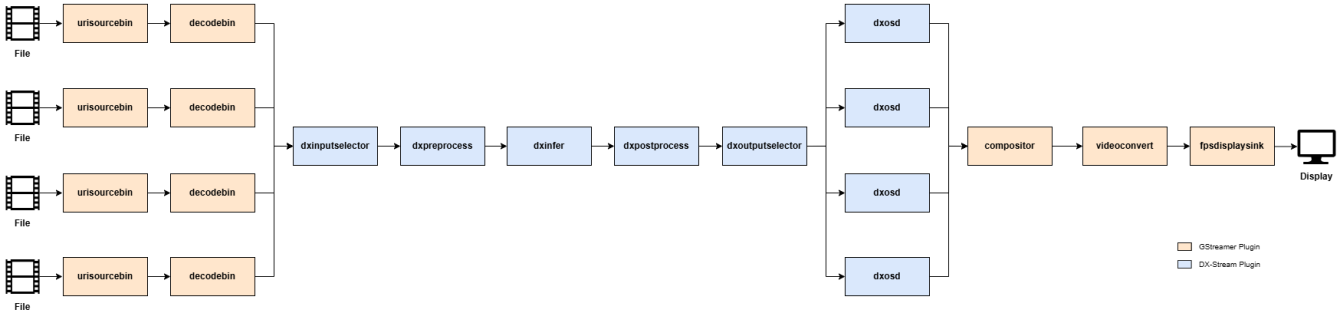
One approach is to configure an independent inference sub-pipeline for each stream, as shown in the figure below.

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_moulti_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.
- **DxScale** and **DxConvert** must be placed **before** DxInputSelector or **after** DxOutputSelector (i.e., per-stream). These elements require stable caps and cannot handle the interleaved multi-stream output of DxInputSelector where caps change on every buffer.

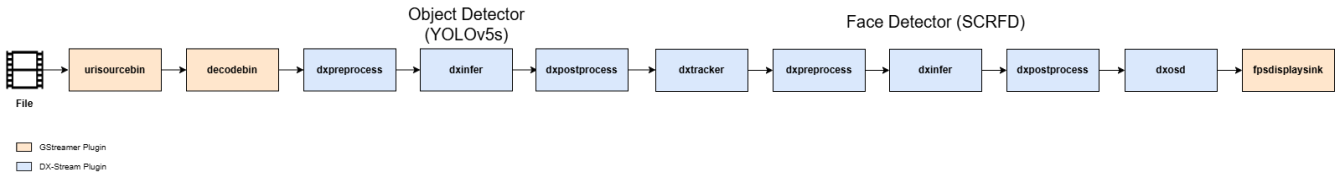
Copyright 2026 DEEPX All Rights Reserved.

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 such as face detection are performed on each detected object, with the results stored in each object's metadata.

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 Face Detection**: Performs face detection on each tracked object

Usage Notes

Configure file setting

- This pipeline runs multiple models sequentially. Be careful to configure each model's preprocess_id and inference_id properly to avoid unexpected behavior.

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.

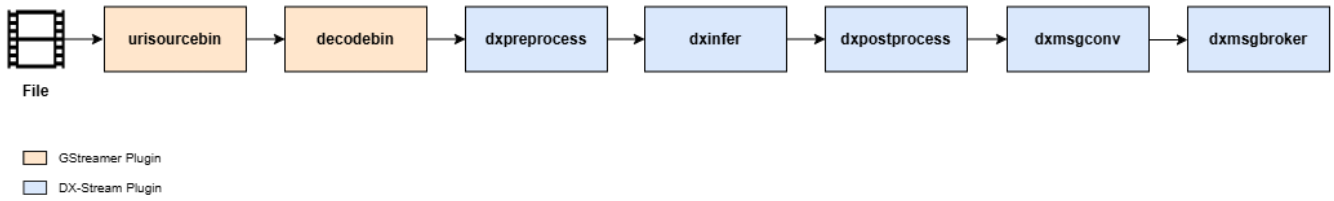
Copyright 2026 DEEPX All Rights Reserved.

5.4 MsgBroker Pipeline

The pipeline demonstrates how to process a local video file with the YOLOv7 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
```

2. Run DX-STREAM Pipeline (Server Side)

```
# Execute the broker pipeline on processing server
# The pipeline uses YOLOv7 model for object detection and converts to JSON messages
cd /path/to/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/Object_Detection/YoloV7/
preprocess_config.json ! \
  dxinfer config-file-path=configs/Object_Detection/YoloV7/inference_config.json ! \
  dxpostprocess config-file-path=configs/Object_Detection/YoloV7/
postprocess_config.json ! \
  dxmsgconv library-file-path=/usr/lib/libdx_msgconvl.so config-file-path=configs/
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
```


Copyright 2026 DEEPX All Rights Reserved.

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

KAFKA DEMO

1. Server Setup (Processing Server with Message Broker)

```
# Install Java and Kafka
sudo apt install default-jdk
wget https://downloads.apache.org/kafka/3.9.0/kafka_2.13-3.9.0.tgz
tar -xzf kafka_2.13-3.9.0.tgz
cd kafka_2.13-3.9.0

# Start Zookeeper and Kafka server
bin/zookeeper-server-start.sh config/zookeeper.properties &
bin/kafka-server-start.sh config/server.properties &
```

2. Run DX-STREAM Pipeline (Server Side)

```
# Execute the broker pipeline on processing server
cd /path/to/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/Object_Detection/YoloV7/
preprocess_config.json ! \
  dxinfer config-file-path=configs/Object_Detection/YoloV7/inference_config.json ! \
  dxpostprocess config-file-path=configs/Object_Detection/YoloV7/
postprocess_config.json ! \
  dxmsgconv library-file-path=/usr/lib/libdx_msgconvl.so config-file-path=configs/
msgconv_config.json ! \
  dxmsgbroker broker-name=kafka conn-info=localhost:9092 topic=test
```

3. Client Application (Any Machine - Consumer Only)

```
# Install Python Kafka client library (client machine only needs this)
pip install kafka-python

# 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 test

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


Copyright 2026 DEEPX All Rights Reserved.

NETWORK ARCHITECTURE

Processing Server:

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

Client Machine:

- Consumer Application (Python/C++)
- Only needs client library (paho-mqtt/kafka-python)
- 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/kafka-python)

> **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 configuration file containing message format properties (optional)
- `library-file-path`: Path to custom message converter library (**required**)
- `message-interval`: Frame interval for message conversion (default: 1)

Copyright 2026 DEEPX All Rights Reserved.

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)

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/
```


Copyright 2026 DEEPX All Rights Reserved.

```
# 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

```
# 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
```


Copyright 2026 DEEPX All Rights Reserved.

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"
```

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/Object_Detection/
YoloV7/preprocess_config.json ! queue ! \
                    dxinfer config-file-path=$SRC_DIR/configs/Object_Detection/YoloV7/
inference_config.json ! queue ! \
                    dxpostprocess config-file-path=$SRC_DIR/configs/Object_Detection/
YoloV7/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
```


Copyright 2026 DEEPX All Rights Reserved.

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,TLSv1.1,TLSv1
ssl.keystore.type=JKS
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
1	ERROR	Critical failures preventing operation
2	WARNING	Non-fatal issues that may affect behavior
3	INFO	State changes and configuration confirmation
4	DEBUG	Detailed operational information for troubleshooting

Copyright 2026 DEEPX All Rights Reserved.

Common Scenarios

Monitor inference performance:

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

Debug configuration loading:

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

Track buffer flow:

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

Identify dropped frames:

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

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

Copyright 2026 DEEPX All Rights Reserved.

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
#     if [ "$(lsb_release -rs)" = "12" ]; then
```


Copyright 2026 DEEPX All Rights Reserved.

```
#         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.

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

Copyright 2026 DEEPX All Rights Reserved.

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"
```


Copyright 2026 DEEPX All Rights Reserved.

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
dx_stream_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.

Message Format Issues

• Check if custom library implements all required functions

• Verify that the final JSON output format matches expected structure

Copyright 2026 DEEPX All Rights Reserved.

Performance Issues

- Adjust `message-interval` property 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 required dependencies (Java JDK is essential). Download and extract the Kafka distribution:

```
$ mkdir utils && cd utils
$ sudo apt update
$ sudo apt-get install default-jdk
$ wget https://downloads.apache.org/kafka/3.9.0/kafka_2.13-3.9.0.tgz
$ tar -xzf kafka_2.13-3.9.0.tgz
$ cd kafka_2.13-3.9.0
```

  Start Zookeeper (terminal 1): Kafka requires Zookeeper to be running first

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

  Start Kafka Broker (Terminal 2): In a separate terminal session, start the Kafka broker

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

> **NOTE**
>
> Keep both terminal sessions running while the DX-STREAM pipeline is active to ensure proper operation.

Copyright 2026 DEEPX All Rights Reserved.

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/`	
`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/`	

Copyright 2026 DEEPX All Rights Reserved.

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:

```
# 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
```


Copyright 2026 DEEPX All Rights Reserved.

```
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)
```


Copyright 2026 DEEPX All Rights Reserved.

7. Appendix

7.1 User Metadata Guide

This guide provides comprehensive information on using the DX-STREAM User Metadata system for storing custom data alongside inference results.

7.1.1 Overview

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

7.1.2 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

7.1.3 Basic Usage Pattern

1. Define Your Data Structure

```
// 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;
```

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;
    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);
```


Copyright 2026 DEEPX All Rights Reserved.

```
    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);
}
```

3. Create and Attach Metadata

Frame Metadata Example:

```
void add_scene_analytics_to_frame(DXFrameMeta *frame_meta,
                                  gint object_count,
                                  const gchar *scene_type) {
    // Acquire user metadata from pool
    DXUserMeta *user_meta = dx_acquire_user_meta_from_pool();

    // 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 = g_get_monotonic_time();

    // Set data with required functions
    gboolean success = dx_user_meta_set_data(user_meta,
                                             analytics,
                                             sizeof(SceneAnalytics),
                                             DXUserMetaType::DX_USER_META_FRAME,
```


Copyright 2026 DEEPX All Rights Reserved.

```
                                          scene_analytics_free,    // Required cleanup
                                          scene_analytics_copy);   // Required copy

    if (success) {
        // Attach to frame
        dx_add_user_meta_to_frame(frame_meta, user_meta);
    } else {
        g_warning("Failed to set user metadata");
        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) {
    // Acquire user metadata from pool
    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
                                             custom_feature_copy);    // Required copy

    if (success) {
        // Attach to object
        dx_add_user_meta_to_obj(obj_meta, user_meta);
    } else {
        g_warning("Failed to set object user metadata");
        dx_release_user_meta(user_meta);
    }
}
```


Copyright 2026 DEEPX All Rights Reserved.

4. 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: %lu 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++) {
                g_print("  Feature[%d]: %.3f\n", i, feature->feature_vector[i]);
            }
        }
    }
}
```


Copyright 2026 DEEPX All Rights Reserved.

7.1.4 Complete Example: Custom Analytics Element

Here's a complete example of a custom element that adds analytics metadata:

```
// 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)
{
    // Add frame-level analytics
    add_scene_analytics_to_frame(frame_meta,
                                 frame_meta->_object_meta_list.size(),
                                 "indoor_scene");

    // Add custom features to each object
    for (auto obj : frame_meta->_object_meta_list) {

        // Generate dummy features for demonstration
        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);
    }
}
```

7.1.5 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

- **Pool allocation**: Use `dx_acquire_user_meta_from_pool()` for efficient memory allocation
- **Minimal data size**: Keep metadata structures compact for better performance
- **Lazy evaluation**: Only compute expensive metadata when actually needed

Copyright 2026 DEEPX All Rights Reserved.

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

7.1.6 Troubleshooting

Common Issues

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

Copyright 2026 DEEPX All Rights Reserved.

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 ...
}
```

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.

7.2 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

7.2.1 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

Copyright 2026 DEEPX All Rights Reserved.

- SD card flashing tool or `dd` command access

7.2.2 Building Process

Step 1: SD Card Setup

Prepare the base Ubuntu OS environment.

STEP 1.1 DOWNLOAD 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)

STEP 1.2 EXTRACT IMAGE

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
```


Copyright 2026 DEEPX All Rights Reserved.

STEP 1.3 FLASH IMAGE TO 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
```

Alternative Methods:

- **Raspberry Pi Imager**: Cross-platform GUI tool
- **Balena Etcher**: User-friendly flashing utility
- **Win32DiskImager**: Windows-specific tool

STEP 1.4 INITIAL BOOT

- (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`

Step 2: DX-RT NPU Driver 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.

Copyright 2026 DEEPX All Rights Reserved.

STEP 2.1 UPDATE 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.

STEP 2.2 CONFIGURE GIT (OPTIONAL)

If you encounter SSL certificate issues with GitHub:

```
git config --global http.sslVerify false
```

> **NOTE**
>
> This disables SSL verification globally. For production use, consider configuring proper certificates instead.

STEP 2.3 BUILD AND INSTALL 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

Copyright 2026 DEEPX All Rights Reserved.

STEP 2.4 BUILD AND INSTALL 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
- Alternative: `./install.sh --minimal` for runtime-only installation

STEP 2.5 VERIFY DX-RT INSTALLATION

Test the NPU driver and runtime installation:

```
dxrt-cli -s
```

Expected Output:

```
DXRT v3.0.0
=======================================================
 * Device 0: M1, Accelerator type
---------------------   Version   ---------------------
 * RT Driver version   : v1.7.1
 * PCIe Driver version : v1.4.1
-------------------------------------------------------
 * FW version          : v2.1.4
---------------------- Device Info ----------------------
 * Memory : LPDDR5 6000 MHz, 3.92GiB
 * Board  : M.2, Rev 1.5
 * Chip Offset : 0
 * PCIe   : Gen3 X4 [01:00:00]

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
dvfs Disabled
=======================================================
```

Troubleshooting:

- If no device is detected, check M.2 module connection
- Verify driver installation with `lsmod | grep dx`
- Check system logs with `dmesg | grep -i deepx`

Copyright 2026 DEEPX All Rights Reserved.

Step 3: Build DX-Stream

Compile and install the DX-Stream GStreamer framework and plugins.

STEP 3.1 INSTALL BUILD DEPENDENCIES

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
```

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

STEP 3.2 CLONE AND BUILD 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 VERIFY INSTALLATION

Check if DX-Stream plugins are properly installed:

```
gst-inspect-1.0 | grep dx
```

Expected Output:

```
dxstream:  dxgather: DX Gather
dxstream:  dxinfer: DX Inference
dxstream:  dxmsgbroker: DX Message Broker
dxstream:  dxmsgconv: DX Message Converter
dxstream:  dxosd: DX On-Screen Display
dxstream:  dxpostprocess: DX Post-process
dxstream:  dxpreprocess: DX Pre-process
...
```

Step 4: Running Demo Applications

Verify the complate system functionality and NPU acceleration.

STEP 4.1 SETUP SAMPLE MODELS AND VIDEOS

```
cd ~/dxnn/dx_stream
./setup.sh
```

STEP 4.2 RUN DEMO PIPELINE

```
./run_demo.sh
```

Available Demo Options:

- Object detection with YOLO models
- Face detection and recognition
- Pose estimation
- Real-time video processing

For detailed demo instructions, refer to the **Installation Guide**.

Copyright 2026 DEEPX All Rights Reserved.

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)

For maximum performance, ensure your pipelines utilize these specific hardware accelerators.

7.2.3 Troubleshooting

This section provides guidance for resolving common issues encountered during the DX-Stream installation or runtime.

Common Issues

1. NPU Not Detected

Verify the NPU module is correctly recongnized and the driver is loaded.

```
# Check PCIe connection
lspci | grep -i deepx
# Verify driver loading
dmesg | grep -i dx_rt
```

2. GStreamer Plugin Not Found

If DX-Stream plugins are not registered, manually refresh the plugin cache.

```
# Refresh plugin cache
gst-inspect-1.0 --plugin-path=/usr/local/lib/gstreamer-1.0
```

3. Build Failures

If the build process fails, clean the environment and rebuild.

```
# Clean and rebuild
./build.sh
```

4. Performance Issues

Copyright 2026 DEEPX All Rights Reserved.

Monitor hardware status if performance is slower than expected.

- Verify NPU Status

```
dxrt-cli -s
```

- Monitor CPU usage

```
htop
```

NOTE

Next Steps. fter successful installation:

- Explore the **Pipeline Examples**
- Learn about **Writing Custom Applications**
- Review **Element Documentation** for advanced usage

Copyright 2026 DEEPX All Rights Reserved.

7.3 Appendix: 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.

7.3.1 Installation

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 (e.g., graphviz, libgraphviz-dev).
- **Source Acquisition**: Clones the GstShark repository from GitHub.
- **Build and Configuration**: Compiles and installs GstShark with the appropriate GStreamer configurations.
- **PATH Update**: Adds GstShark visualization and analysis tools to your system's PATH environment variable.

> **NOTE**
>
> The installation requires sudo privileges for system-wide installation of GstShark libraries and tools.

7.3.2 Usage and Analysis Methodology

This section details how to use GstShark to analyze the performance of GStreamer pipelines, ranging from standard video processing to complex DX-Stream AI pipelines.

Basic Usage

GstShark uses environment variables to control which performance data is collected and where the results are stored.

Copyright 2026 DEEPX All Rights Reserved.

ENVIRONMENT VARIABLES

- `GST_TRACER` : 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

Sample Pipeline Analysis

BASIC PERFORMANCE EVALUATION

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/
codec_test_clip_h264_16Mbps.mp4 ! \
    qtdemux ! h264parse ! avdec_h264 ! videoconvert ! fakesink
```


Copyright 2026 DEEPX All Rights Reserved.

ADVANCED ANALYSIS WITH RESULT STORAGE

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/
codec_test_clip_h264_16Mbps.mp4 ! \
    qtdemux ! h264parse ! avdec_h264 ! videoconvert ! fakesink

# View results
ls -la /tmp/gst-shark-results/
```

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.

```
GST_DEBUG="GST_TRACER:7" GST_TRACERS="cpuusage;proctime;framerate;queuelevel" \
    gst-launch-1.0 filesrc location=./dx_stream/samples/videos/
codec_test_clip_h264_16Mbps.mp4 ! \
    qtdemux ! h264parse ! avdec_h264 ! \
    dxpreprocess config-file-path=./dx_stream/configs/Object_Detection/YoloV7/
preprocess_config.json ! \
    dxinfer config-file-path=./dx_stream/configs/Object_Detection/YoloV7/
inference_config.json ! \
    dxpostprocess config-file-path=./dx_stream/configs/Object_Detection/YoloV7/
postprocess_config.json ! \
    dxosd ! fakesink
```


Copyright 2026 DEEPX All Rights Reserved.

Result Analysis

CONSOLE OUTPUT ANALYSIS

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

GRAPHICAL ANALYSIS

If GstShark graphics tools (such as gstshark-plot) were installed with the necessary dependencies, you can generate visual reports for clearer performance trending and bottleneck visualization.

```
# Generate performance graphs (if graphics tools are installed)
gstshark-plot /tmp/gst-shark-results/ -s pdf
```

7.3.3 Performance Optimization Tips

This section provides strategies for interpreting GstShark results and applying common optimization techniques to enhance DX-Stream pipeline performance.

Copyright 2026 DEEPX All Rights Reserved.

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

Applying these strategies can alleviate identified bottlenecks:

- Element Configuration: Adjust element-specific parameters, such as quantization settings in the NPU elements 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 GPU acceleration (Mali-G610 MP4) is enabled for supported non-NPU elements (e.g., video conversion) to offload the host CPU.

7.3.4 Troubleshooting

This section addresses common issues encountered when using GstShark for performance analysis.

Copyright 2026 DEEPX All Rights Reserved.

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

If the GstShark tracers (e.g., sharktime, sharklog) 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
```


Copyright 2026 DEEPX All Rights Reserved.

NOTE

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.

8. Change Log

8.1 Version 3.0.0 (Apr 2026)

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)**
- **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`

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
- SW 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)

Copyright 2026 DEEPX All Rights Reserved.

Added

- **DxScale**: New GstBaseTransform element for HW-accelerated scaling (NV12/I420/RGB/BGR)
- **DxConvert**: New GstBaseTransform element for color format conversion (full 4×4 format matrix)
- RGA HW-accelerated format conversion/scaling (NV12↔RGB↔BGR) with automatic SW 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

8.2 Version 2.2.1 (Feb 2026)

Changed

- Update sample models version from dx_com v2.2.0 to v2.2.1

8.3 Version 2.2.0 (Dec 2025)

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
- docs: update OS requirements in installation guide for debian

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

Copyright 2026 DEEPX All Rights Reserved.

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

8.4 Version 2.1.0 (Nov 2025)

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
- **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'.

Copyright 2026 DEEPX All Rights Reserved.

Fixed

- **Setup Scripts**: Improved error handling and prevented excessive download retry attempts
- **Shutdown Flow**: Improved shutdown signal processing in dx-infer 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

- **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
- 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

Known Issues. - DeepLabV3 Semantic Segmentation model accuracy may be slightly degraded in dx-compiler(dx_com) v2.1.0. This will be fixed in the next release. The DeepLabV3 model used in the demo was converted using dx-compiler v2.0.0. - When using the PPU model for face detection & pose estimation, dx-compiler v2.1.0 does not currently support converting face and pose models to PPU format. This feature will be added in a future release. The PPU models used in the demo were converted using dx-compiler v1.0.0(dx_com v1.60.1).

Copyright 2026 DEEPX All Rights Reserved.

8.5 Version 2.0.0 (Aug 2025)

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 dx-gather event handling logic.
- Removed unnecessary print statements.
- feat: enhance build script and update installation documentation
    - Added OS and architecture checks in the build script
    - Updated CPU and OS specifications in the installation documentation for clarity

Fixed

- Bug Fix: Addressed and alleviated a processing delay issue within the dx-inputselector.
- 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.
- feat: improve error handling for setup scripts
- feat: add 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

Added

- feat: add uninstall script and enhance color utility functions
    - Introduced a new uninstall.sh script for cleaning up project files and directories

Copyright 2026 DEEPX All Rights Reserved.

8.6 Version 1.7.0 (Jul 2025)

Changed

- 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.
- auto run setup script when a file not found error occurs during example execution
- apply colors and handle errors in scripts

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.

Copyright 2026 DEEPX All Rights Reserved.