Edge Vision Analytics SDK Programming Guide

Manual Rev.: 1.1
Revision Date: April 12, 2021
Part Number: 50M-00009-1010

LEADING EDGE COMPUTING

Preface

Copyright

Disclaimer

The information in this document is subject to change without prior notice in order to improve reliability, design, and function and does not represent a commitment on the part of the manufacturer. In no event will the manufacturer be liable for direct, indirect, special, incidental, or consequential damages arising out of the use or inability to use the product or documentation, even if advised of the possibility of such damages.

Trademarks

Product names mentioned herein are used for identification purposes only and may be trademarks and/or registered trademarks of their respective companies.

Revision History

Revision	Description	Date
1.0	Initial release	2020-09-30
1.1	Release for EVA SDK R3	2021-04-12

Table of Contents

Preface ii

1 Introduction 1

2 Developing Elements/Plugins with C 3

2.1 Class Declaration 3
2.2 Constructors and Deconstructors 4
2.3 Object Lifecycle 4
2.4 Class Implementation 5
2.5 Sink and Source Pad Association 6
2.6 Overriding GstVideoFilter transform_frame_ip 7
2.7 Plugin Registration 8

3 Developing Elements/Plugins with Python 9

3.1 Import Glib Module 9
3.2 Class Declaration 9
3.3 Class Implementation 9
3.4 Register Python Element 11
3.5 Install a Python Element 11

4 Python Sample to Interpret Inference Result as a Yolov3 Box Detection 13

4.1 Python Sample Code 13
4.2 Draw Boxes in an Image 15
4.3 Custom Translatation of Code 15
4.4 Python Application Example 15

5 How to Use ADLINK Metadata 17

5.1 ADLINK Metadata Architecture 17
5.2 Using ADLINK Metadata 20

6 Integrating the GStreamer Plugin 21

6.1 Method 1 21
6.2 Method 2 23
6.3 Python Method 25

Safety Instructions 29

Getting Service 30

This page intentionally left blank.

1 Introduction

The purpose of this programming guide is to demonstrate the concepts of the GStreamer element/plugin/application program architecture in order to help users develop their own customized products with the ADLINK Edge Vision Analytics SDK. GStreamer is built on top of the GObject (for object-orientation) and Glib (for common algorithms) libraries, so some prior knowledge of these object-oriented concepts will be helpful before continuing with this guide.

This programming guide covers the following topics.

- Developing Elements/Plugins with C illustrates the architecture of the GObject-style objective oriented program to explain the overall view of the element and plugin in C.
- Developing Elements/Plugins with Python illustrates the architecture of the GObject-style objective oriented program to explain the overall view of the element and plugin in Python.
- How to Use ADLINK Metadata illustrates how to retrieve and save the ADLINK defined metadata structure within the GStreamer buffer.
- Integrating the GStreamer Plugin includes examples of how to integrate code with Gstreamer to give a clearer understanding of how GStreamer is used in applications.

This page intentionally left blank.

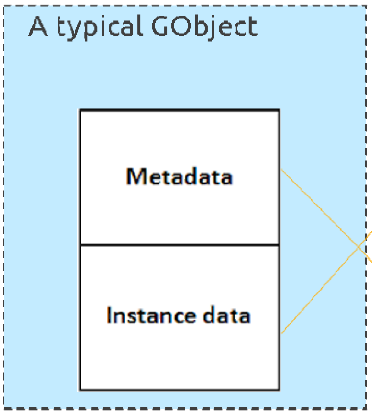
2 Developing Elements/Plugins with C

This section begins with a general summary of the GObject-style definition for objects with a focus on the information required to build one class and functions and concludes with an introduction to the assembly of the GStreamer constructions to help developers who do not know how to implement custom elements.

2.1 Class Declaration

In GObject, class declaration is quite different between C++, C#, JAVA, and other object-oriented programming languages. The GObject system implements its object-oriented based system on C which does not support object-orientation.

In order to use C syntax to support object-oriented semantics standardized by GObject, first describe the class metadata and then the instance data belonging to the class, as shown in the following example (see also adaifiltertemplate.h and adaifiltertemplate.cpp).




```c
typedef struct _AdFilterTemplate AdFilterTemplate;
typedef struct _AdFilterTemplateClass AdFilterTemplateClass;
typedef struct _AdFilterTemplatePrivate AdFilterTemplatePrivate;

struct _AdFilterTemplate
{
  GstVideoFilter base;
  AdFilterTemplatePrivate *priv;
};

struct _AdFilterTemplateClass
{
  GstVideoFilterClass base_ad_filter_template_class;
};
```

The GObject system combines these two struct definitions at execution time. These two struct definitions must have a typedef declaration with the same name without an underline; this is the format commonly used by GObject in GStreamer or other libraries. The private structure is defined in _AdFilterTemplatePrivate in .cpp in case of wrapping the source code.

GObject also requires defining the parent class and instance first in each class metadata structure and instance data structure. This will let GObject know which class declarations to inherit from. In this example, _AdFilterTemplate and _ADFilterTemplateClass both inherit from the parent classes GstVideoFilter and GstVideoFilterClass, respectively.

2.1.1 Register to GObject

After defining the class content, register the class type for the GObject system.

```c
GType ad_filter_template_get_type(void);
```

This step allows the GObject system to identify the class by returning GType and casting the result to the right class at execution time.

2.1.2 Casting Macros

```
#define AD_TYPE_FILTER_TEMPLATE \
  (ad_filter_template_get_type())

#define AD_FILTER_TEMPLATE(obj) \
  (G_TYPE_CHECK_INSTANCE_CAST((obj), AD_TYPE_FILTER_TEMPLATE, AdFilterTemplate))

#define AD_FILTER_TEMPLATE_CLASS(klass) \
  (G_TYPE_CHECK_CLASS_CAST((klass), AD_TYPE_FILTER_TEMPLATE, AdFilterTemplateClass))

#define AD_IS_FILTER_TEMPLATE(obj) \
  (G_TYPE_CHECK_INSTANCE_TYPE((obj), AD_TYPE_FILTER_TEMPLATE))

#define AD_IS_FILTER_TEMPLATE_CLASS(klass) \
  (G_TYPE_CHECK_CLASS_TYPE((klass), AD_TYPE_FILTER_TEMPLATE))
```

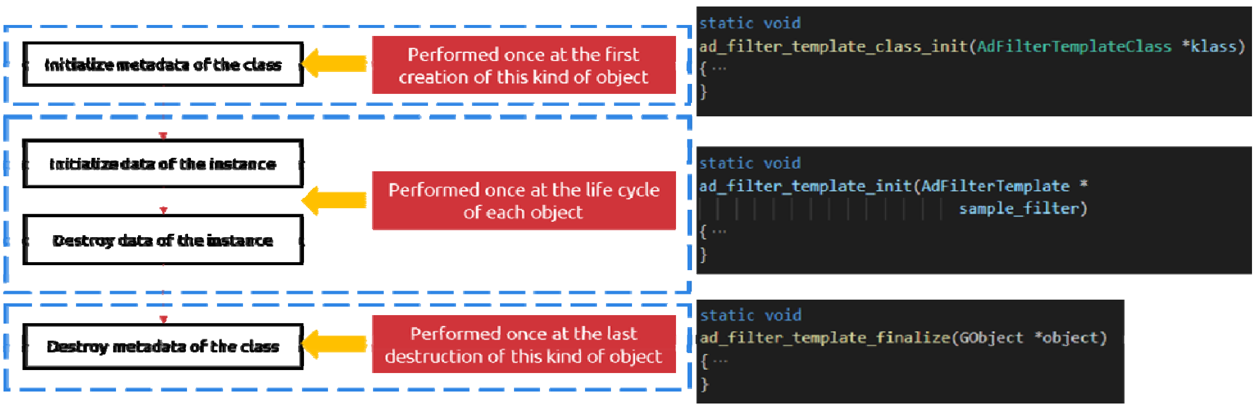
The above are the required macro definitions for casting the GObject and must be defined between the G_BEGIN_DECLS and G_END_DECLS tags. It is common with GObject that the inherited functions and members are called or used.

2.2 Constructors and Deconstructors

The file adaifiltertemplate.cpp implements the ad_filter_template_class_init and ad_filter_template_init constructors. When the class object memory is allocated, GInstanceInit() is called to initialize the class, then the constructors. When a class is going to be destroyed, the deconstructor will first be called. In the GObject, there is no clear connection between constructor and deconstructor. The GObject separates the deconstructor into dispose and finalize. If the class references another object, it is required to release the reference of that object in the dispose stage (refer to gst_object_unref for more details). In the finalize stage, all the allocated memory for this class is released.

2.3 Object Lifecycle

The chart below shows the lifecycle of the GObject class described above. Class initialization is only performed once when the class is first used in the lifecycle, and each instance of the class is individually initialized and destroyed in its lifecycle. Once the class is not used, it is permanently destroyed.



2.4 Class Implementation

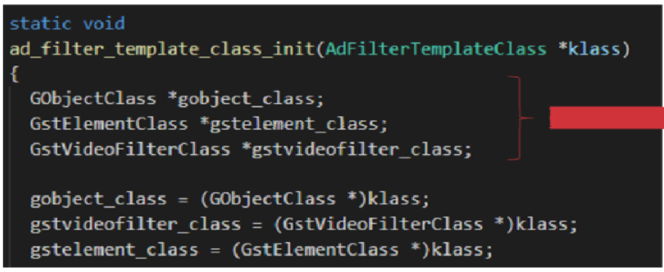
In the GObject, you are required to cast the type to the parent and assign the implemented function to perform an override.

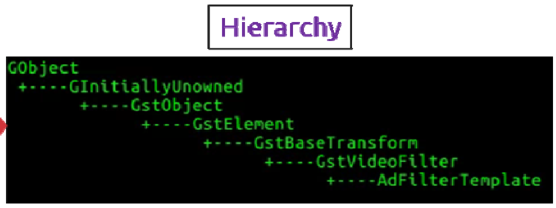
2.4.1 Casting in class_init

```
GObjectClass *gobject_class;
GstElementClass *gstelement_class;
GstVideoFilterClass *gstvideofilter_class;

gobject_class = (GObjectClass *)klass;
gstvideofilter_class = (GstVideoFilterClass *)klass;
gstelement_class = (GstElementClass *)klass;
```

Take ad_filter_template as an example, which has the following inheritance:





2.4.2 Override the Method of the Parent

Casting to the right parent to override the method you are going to use is the way that the GObject system runs. The class function or virtual function descriptions can be queried by the GStreamer API references. For the GObjectClass in ad_filter_template_class_init, there are four main functions that must be implemented for the class:

1. _set_property
2. _get_property
3. _dispose
4. _finalize

parent method overriding

```
gobject_class->set_property = ad_filter_template_set_property;
gobject_class->get_property = ad_filter_template_get_property;
gobject_class->dispose = ad_filter_template_dispose;
gobject_class->finalize = ad_filter_template_finalize;
```

2.4.3 Implement Object Properties

The GObject implements a get/set mechanism for object properties. This mechanism allows the user to read through GObject's g_object_get_property or write data with g_object_set_property by knowing the name of the object property. The class supports this mechanism by registering each class property through g_object_class_install_property and overriding the _set_property and _get_property functions, For example:

get/set property install

```
g_object_class_install_property(gobject_class, PROP_FILTER_TYPE,
                                g_param_spec_int("type", "Type",
                                                 "Filter type 1.Edge 2.Gray",
                                                 0, 1, DEFAULT_FILTER_TYPE,
                                                 (GParamFlags)G_PARAM_READWRITE));

g_object_class_install_property(gobject_class, PROP_EDGE_VALUE,
                                g_param_spec_int("edge-value", "edge value",
                                                 "Threshold value for edge image",
                                                 0, 255, DEFAULT_EDGE_VALUE,
                                                 (GParamFlags)G_PARAM_READWRITE));
```

2.5 Sink and Source Pad Association

There are two kinds of pads in GStreamer: sink and source.

2.5.1 Pad Definition

Define sink or source pad factories under the GStreamer pad template. Be sure to define the name of the pad, the direction, presence, and capabilities.

```
static GstStaticPadTemplate sink_factory = GST_STATIC_PAD_TEMPLATE(
                                    "sink",
                                    GST_PAD_SINK,
                                    GST_PAD_ALWAYS,
                                    GST_STATIC_CAPS(GST_VIDEO_CAPS_MAKE("{ BGR }")));

static GstStaticPadTemplate src_factory = GST_STATIC_PAD_TEMPLATE(
                                    "src",
                                    GST_PAD_SRC,
                                    GST_PAD_ALWAYS,
                                    GST_STATIC_CAPS(GST_VIDEO_CAPS_MAKE("{ BGR }")));
```

2.5.2 Adding a Pad

Add the factories into the class_init of the GObject class. Once the class has been initialized, the GStreamer factory is created.

```
gst_element_class_add_pad_template(gstelement_class,
                                   gst_static_pad_template_get(&src_factory));
gst_element_class_add_pad_template(gstelement_class,
                                   gst_static_pad_template_get(&sink_factory));
```

2.6 Overriding GstVideoFilter transform_frame_ip

The `ad_filter_template`, inherited from GstVideoFilter, is useful for focusing on the video algorithm without additional GStreamer tasks like negotiation, capability checks, or status checks. This element is focused on processing frame data, so it is suitable to inherit the parent GstVideoFilter class. To achieve this implementation, override the virtual method transform_frame_ip.

```
gstvideofilter_class->transform_frame_ip = GST_DEBUG_FUNCPTR(
                                                 ad_filter_template_transform_frame_ip);
```

In `ad_filter_template_transform_frame_ip`, `GstVideoFilter` passes the `GstVideoFrame` wrapper for the user to directly access the data related to the frame, unlike other GStreamer elements that pass the buffer directly. The buffer does not have to be parsed into the generic frame format.

```
static GstFlowReturn
ad_filter_template_transform_frame_ip(GstVideoFilter *filter, GstVideoFrame *frame)
{
    AdFilterTemplate *sample_filter = AD_FILTER_TEMPLATE(filter);
    GstMapInfo info;
    Mat output_image;
    int filter_type;
    int edge_threshold;
    gst_buffer_map(frame->buffer, &info, GST_MAP_READ);
    ad_filter_template_initialize_images(sample_filter, frame, info);
    AD_FILTER_TEMPLATE_LOCK(sample_filter);
    filter_type = sample_filter->priv->filter_type;
    edge_threshold = sample_filter->priv->edge_value;
    AD_FILTER_TEMPLATE_UNLOCK(sample_filter);
    if (filter_type == 0)
    {
      GST_DEBUG("Calculating edges");
      Canny((*sample_filter->priv->cv_image), output_image,
            edge_threshold, 255);
    }
    else if (filter_type == 1)
    {
      GST_DEBUG("Calculating black&white image");
      cvtColor((*sample_filter->priv->cv_image), output_image, COLOR_BGR2GRAY);
    }
    if (output_image.data != NULL)
    {
      GST_DEBUG("Updating output image");
      ad_filter_template_display_background(sample_filter, output_image);
    }
    gst_buffer_unmap(frame->buffer, &info);
    return GST_FLOW_OK;
}
```

`gst_buffer_map` and `gst_buffer_unmap` deal with the mapping tasks from frame data to `info`. `map` and `unmap` also deals with read/write management to ensure the data is exclusively occupied. In the example, the processing option `filter_type` decides what the process is going to do: 1 for color to gray; 0 for Canny edge detection.

After the process is done, `ad_filter_template_display_background` will copy back to the frame referenced at the beginning. Then, the video frame buffer is passed downstream.

2.7 Plugin Registration

GStreamer uses a registration script to wrap elements into the plugin. The plugin is a .so file stored in the operating system accessed by GStreamer.

```
gboolean
ad_filter_template_plugin_init(GstPlugin *plugin)
{
    return gst_element_register(plugin, PLUGIN_NAME, GST_RANK_NONE,
                                AD_TYPE_FILTER_TEMPLATE);
}

GST_PLUGIN_DEFINE(
      GST_VERSION_MAJOR,
      GST_VERSION_MINOR,
      adfiltertemplate,
      "ADLINK filter template plugin",
      ad_filter_template_plugin_init,
      PACKAGE_VERSION,
      GST_LICENSE,
      GST_PACKAGE_NAME,
      GST_PACKAGE_ORIGIN)
```

The plugin's information is stored in config.h included in a .cpp file. Add an element register inside plugin_init, and provide the required plugin definition.

So far, we have introduced the basic concepts of the GObject structure used in C and the way to implement a basic element in GStreamer. There are other kinds of the elements described in GStreamer such as the sink element, src element, and multi-Pad element which are implemented in a similar way. Refer to the Gstreamer API reference for more information on creating custom GStreamer elements.

3 Developing Elements/Plugins with Python

Relative to programming in C, it is easier to develop elements/plugins in Python. The following uses the classifier_sample element as an example of programming elements/plugins in Python.

3.1 Import Glib Module

GStreamer is built on Glib and GObject which are compatible across platforms and programming languages. However, the following modules must still be included.

```
from gi.repository import Gst, GObject
```

Developing a GStreamer application in Python requires a Gst version and initialization before using the Gst function because the GStreamer Python element loader will handle this step.

```
import gi
gi.require_version('Gst', '1.0')
from gi.repository import Gst, GObject
Gst.init([])
```

3.2 Class Declaration

Defines the class and inherits a subclass of Gst.Element.

```
class ClassifierSamplePy(Gst.Element):
```

3.3 Class Implementation

3.3.1 Initialize Class Metadata

```
class ClassifierSamplePy(Gst.Element):

    # MODIFIED - Gstreamer plugin name
    GST_PLUGIN_NAME = 'classifier_sample'

    __gstmetadata__ = ("Name",
                       "Transform",
                       "Description",
                       "Author")

    __gsttemplates__ = (Gst.PadTemplate.new("src",
                                            Gst.PadDirection.SRC,
                                            Gst.PadPresence.ALWAYS,
                                            Gst.Caps.new_any()),
                        Gst.PadTemplate.new("sink",
                                            Gst.PadDirection.SINK,
                                            Gst.PadPresence.ALWAYS,
                                            Gst.Caps.new_any()))

    _sinkpadtemplate = __gsttemplates__[1]
    _srcpadtemplate = __gsttemplates__[0]

    # MODIFIED - Gstreamer plugin properties
    __gproperties__ = {
      "class-num": (int,  # type
                    "class-num",  # nick
                    "Class number",  # blurb
                    1,  # min
                    65536,  # max
                    1001,  # default
                    GObject.ParamFlags.READWRITE  # flags
      ),
      ...
    }
    ...
```

3.3.2 Initialize Class Instance

Initialize properties before base class initialization.

```
class ClassifierSamplePy(Gst.Element):

    ...

    def __init__(self):
      self.class_num = 1001
      self.batch_num = 1
      self.label = ""
      self.labels = None

      super(ClassifierSamplePy, self).__init__()
      ...
```

3.3.3 Sink and src Pad Association

New sink and src pads from template, including register callbacks for events, queries, or dataflow on the pads.

```
class ClassifierSamplePy(Gst.Element):

    ...

    def __init__(self):
      ...

      self.sinkpad = Gst.Pad.new_from_template(self._sinkpadtemplate, 'sink')
      self.sinkpad.set_chain_function_full(self.chainfunc, None)
      self.sinkpad.set_event_function_full(self.eventfunc, None)

      self.add_pad(self.sinkpad)
      self.srcpad = Gst.Pad.new_from_template(self._srcpadtemplate, 'src')
      self.add_pad(self.srcpad)
```

3.3.4 Override set and get Property Function

Override property function to implement get and set property features.

```
class ClassifierSamplePy(Gst.Element):

    ...

    def __init__(self):
      ...

    def do_get_property(self, prop: GObject.GParamSpec):
      # Implement your get property
      ...

    def do_set_property(self, prop: GObject.GParamSpec, value):
      # Implement your get property
      ...
```

3.3.5 Implement Chain Function

When a sink pad pushes the buffer, the pad will call the chainfunc callback function. Implement logical frame operations in this function and push the buffer into the src pad to pass the buffer into the next element.

```
class ClassifierSamplePy(Gst.Element):

    ...

    def chainfunc(self, pad: Gst.Pad, parent, buffer: Gst.Buffer) -> Gst.FlowReturn:
      # Implement your frame operate logical here
      ...

      return self.srcpad.push(buffer)
```

3.4 Register Python Element

You need to register the Python element after implementing the element class, and then GStreamer can scan the element.

```
class ClassifierSamplePy(Gst.Element):

    ...

    GObject.type_register(ClassifierSamplePy)
    __gstelementfactory__ = (ClassifierSamplePy.GST_PLUGIN_NAME,
                             Gst.Rank.NONE, ClassifierSamplePy)
```

The Python element must define the `__gstelementfactory__` variable because the GStreamer Python loader will scan all Python modules in the plugin path and check whether this module defines `__gstelementfactory__`.

Modules that do not implement the variable can be skipped.

3.5 Install a Python Element

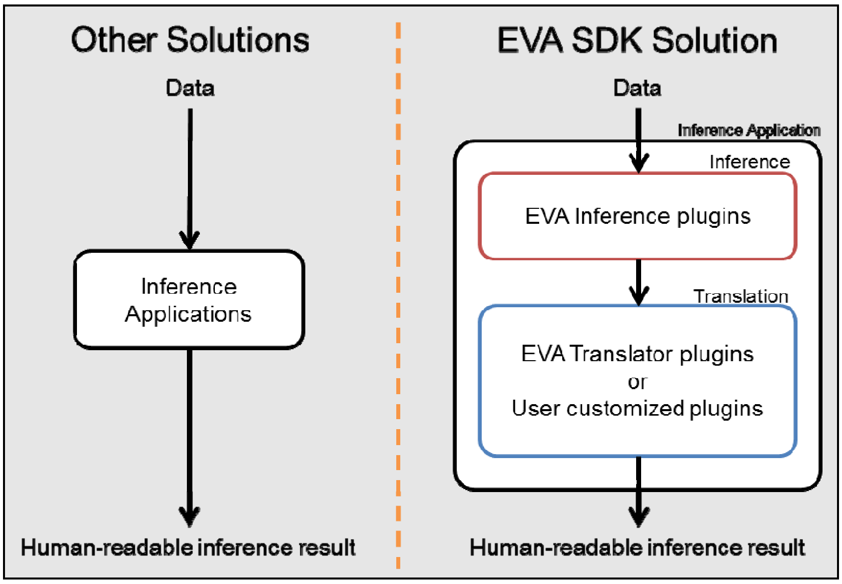
Usually, GStreamer scans plugins under the GST_PLUGIN_PATH environment variable. However, Python elements must be installed in the "python" folder under GST_PLUGIN_PATH. In the example below, the GST_PLUGIN_PATH is /plugins, and there is a Python element named classifier_sample.py.

```
plugins
├── libadfiltertemplate.so
├── ...
├── libpylonsrc.so
└── python
    ├── ...
    └── classifier_sample.py
```

This page intentionally left blank.

4 Python Sample to Interpret Inference Result as a Yolov3 Box Detection

A deep learning inference application will infer input data with specific deep learning models. Each deep learning model will have a different inference result format. The application needs to change code to interpret inference results after changing models. This causes the inference application to have a high dependency on the deep learning model.



The EVA SDK separates the inference application into two parts: inference and translation. The translation part is to translate the raw inference result acquired from an inference part to a more human-readable format. Because the translation plugins require the raw inference result from EVA inference plugins, the translator plugin must follow the inference plugin. Users can easily swap between inference parts, such as OpenVINO or TensorRT, without needing to modify the translation part, or swap the translation part according to the deep learning model without needing to modify the inference part.

The EVA SDK inference plugins support most inference models, so users can focus on developing their own translation part for translating the inference results into human-readable format.

The following section includes Python sample code on how to integrate translation code with the EVA SDK inference application.

4.1 Python Sample Code

Normally, a GStreamer element will send only a single buffer with image data. However, the inference element will send a buffer list with the first index of the list being image data, and the second index being the inference data. Therefore, the translation element needs to receive the buffer list to extract both image and inference data.

```
...

class AdYoloPy(Gst.Element):

...

  def __init__(self):
    ...

    self.sinkpad.set_chain_function_full(self.chainfunc, None)

    self.sinkpad.set_chain_list_function_full(self.chainlistfunc, None).
    ...

  def chainlistfunc(self, pad: Gst.Pad, parent, buff_list: Gst.BufferList) ->
Gst.FlowReturn:
    ...
```

YoloV3 is a special deep learning architecture having multiple output blobs with different target box sizes. The code needs to separate inference data into multiple buffers depending on the output blob size. By default, YoloV3 will have three output blobs and YoloV3-Tiny will have two output blobs. The buffer size will change according to the batch number, class number, and blob output size. The YoloV3 author's pre-train model has 80 class numbers and the output blob's sizes are 26x26, 52x52, and 13x13 (width x height). The following code is to calculate the size of each blob.

```
class AdYoloPy(Gst.Element):

  ...

  def chainlistfunc(self, pad: Gst.Pad, parent, buff_list: Gst.BufferList) ->
Gst.FlowReturn:
    class_coord_dim = (self.class_num + 5) * 3
    out_sizes = list(map(lambda bs: self.batch_num * class_coord_dim * bs[0] * bs[1],
self.blob_size))
    ...
```

The following code will get the size of each blob, extract the second index of the buffer list and convert it to a big buffer. The EVA SDK provides an API to extract image data from a buffer list.

```
class AdYoloPy(Gst.Element):

  ...

  def chainlistfunc(self, pad: Gst.Pad, parent, buff_list: Gst.BufferList) ->
Gst.FlowReturn:
    ...

    with gst_helper.get_inference_data_to_numpy(buff_list, (sum(out_sizes))) as data:
      ...
```

Next, separate the big buffer into multiple buffers and reshape each buffer to the corresponding blob dimension. The blob dimension is Batch x Class (including the coordinate information) x Blob width x Blob height.

```
...

with gst_helper.get_inference_data_to_numpy(buff_list, (sum(out_sizes))) as data:
  offset = 0
  for idx, s in enumerate(out_sizes):
    blob = data[offset:offset+s].reshape(self.batch_num, class_coord_dim,
                                          *self.blob_size[idx])
    mask = self.mask[idx]
    anchor = list(map(lambda m: self.anchor[m], mask))
    _boxs = parse_yolo_output_blob(blob, self.input_width, self.input_height, mask, anchor,
threshold=self.threshold)
    boxs += _boxs
    offset += s

...
```

After getting the blob buffer, interpret blobs as boxes. Parsing the YoloV3 format buffer is out of the scope of this document. For more details, refer to
https://github.com/pjreddie/darknet/blob/f6d861736038da22c9eb0739dca84003c5a5e275/src/yolo_layer.c#L275

Below is an example of parsing code.

```
def parse_yolo_output_blob(blob, iw, ih, mask, anchor, threshold=0.8):
  ...
```

4.2 Draw Boxes in an Image

After obtaining the boxes, the program needs to draw the boxes in image data, so it will extract the first index of the buffer list. The EVA SDK provides two APIs to get writable buffers from the buffer list and convert the buffers to NumPy like data. The data then can be used like an image in the OpenCV API.

```
class AdYoloPy(Gst.Element):

  ...

  def chainlistfunc(self, pad: Gst.Pad, parent, buff_list: Gst.BufferList) ->
Gst.FlowReturn:
    ...

    buf = gst_helper._gst_get_buffer_list_writable_buffer(buff_list, 0)
    img = gst_cv_helper.pad_and_buffer_to_numpy(pad, buf, ro=False)
    # Draw yolo results
    draw_boxs(img, boxs, self.labels)
    ...
```

Finally, release inference data and send image data to the next element.

```
class AdYoloPy(Gst.Element):

  ...

  def chainlistfunc(self, pad: Gst.Pad, parent, buff_list: Gst.BufferList) ->
Gst.FlowReturn:
    ...

    buff_list.remove(1, 1)

    return self.srcpad.push(buff_list.get(0))
```

4.3 Custom Translatation of Code

The custom translation of code requires the following steps.

1. Define your own properties. The properties of the sample code were used for the YoloV3 model. Each model has its own required information.
2. Understand how to exact inference data, as YoloV3 has multiple output blobs, and divide a big buffer into multiple buffers. Models with only one output blob are easier to extract image data from.
3. Implement code to interpret the blob buffer into human-readable data such as classification, detection box, or segmentation. Each model has its own output format.
4. After changing the parts above, users can integrate their own translation application with the EVA inference application.

4.4 Python Application Example

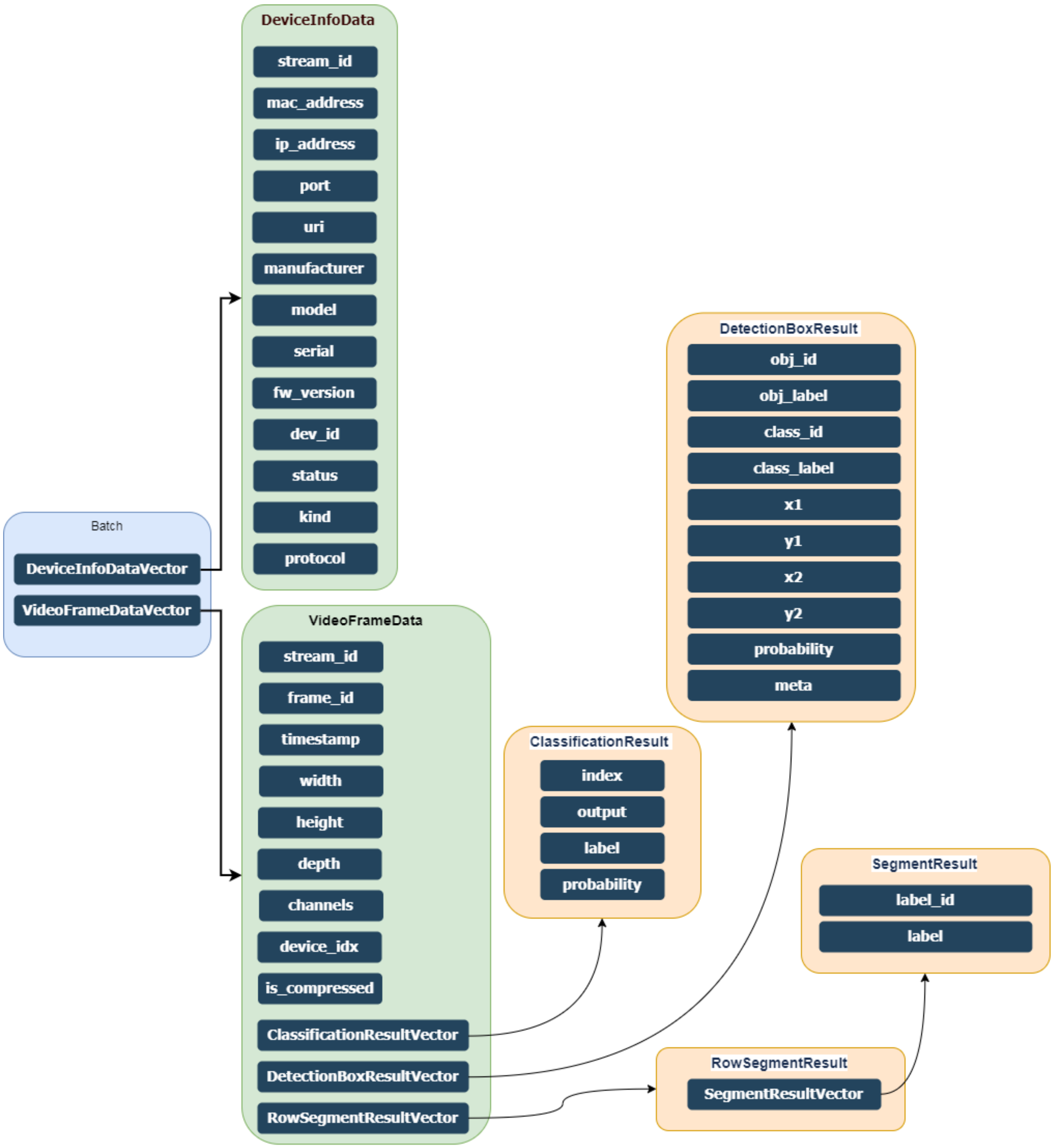
After finishing the designed plugin, we are going to describe the python examples to show two parts: (1) Pure python application and (2) use the python plugin in the python application example. After you had finished the plugin, you can simply create a python application to use the python plugin to integrate the python program inside. Refer to **6.3 Python Method on page 25** for more details.

This page intentionally left blank.

5 How to Use ADLINK Metadata

5.1 ADLINK Metadata Architecture

ADLINK provides structured metadata within the GStreamer pipeline, storing information about the frame, device, and inference results as shown below.



There are six different structures:

- ADBatch
- DeviceInfoData
- VideoFrameData
- DetectionBoxResult
- ClassificationResult
- SegmentResult

Each structure has its own items, as described in the following sections.

5.1.1 AdBatch Structure

Field Name	Type	Description
DeviceInfoDataVector	vector<DeviceInfoData>	The device information of each frame in this batch.
VideoFrameDataVector	vector<VideoFrameData>	Frames in this batch.

5.1.2 DeviceInfoData Structure

Field Name	Type	Description
stream_id	gchar*	Stream publisher ID
mac_address	gchar*	Host address
ip_address	gchar*	Host machine IP Address
port	gint32	Connection port
uri	gchar*	Video Interface URI (rtsp://xx/h264)
manufacturer	gchar*	Vision Device manufacturer
model	gchar*	Vision Device model
serial	gchar*	Vision Device serial identifier
fw_version	gchar*	Vision Device firmware version
dev_id	gchar*	Vision Device host interface (e.g. /dev/video0 or /dev/ttyUSB0)
status	gchar*	DeviceStatus enum (OASYS defined)
kind	gchar*	Vision device kind enum (OASYS defined)
protocol	gchar*	ProtocolKind enum describing how the device communicates

5.1.3 VideoFrameData Structure

Field Name	Type	Description
stream_id	gchar*	Stream publisher ID
frame_id	guint32	Frame sample ID
timestamp	gint64	Time of image capture event
width	guint32	Frame width
height	guint32	Frame height
depth	guint32	Bit per pixel
channels	guint32	Channels
device_idx	guint32	Index of the DeviceInfoDataVec of this Batch
is_compress	gboolean	Compression used for video frame
ClassificationResultVector	vector<ClassificationResult>	The inference result of classification
DetectionBoxResultVector	vector<DetectionBoxResult>	The inference result of detection boxes

5.1.4 ClassificationResult Structure

Field Name	Type	Description
index	gint32	Classification index
output	gchar*	Output type - used when classification model has multiple types of labels for each output index
label	gchar*	Classification label name
probability	gfloat32	Network confidence

5.1.5 DetectionBoxResult Structure

Field Name	Type	Description
obj_id	gint32	Detected object's id
obj_label	gchar*	Detected object's proper name
class_id	gint32	Detected object's classification type as raw id
class_label	gchar*	Detected object's classification as proper name
x1	gfloat32	Top Left X Coordinate (% from 0,0). (frame base, not batch base)
y1	gfloat32	Top Left Y Coordinate (% from 0,0). (frame base, not batch base)
x2	gfloat32	Bottom Right X Coordinate (% from 0,0). (frame base, not batch base)
y2	gfloat32	Bottom Right Y Coordinate (% from 0,0). (frame base, not batch base)
probability	gfloat32	Network confidence
meta	gchar*	Buffer for extra inference metadata

5.1.6 SegmentResult Structure

Field Name	Type	Description
label_id	gint32	Label id
label	gchar*	Label string
label_id	gint32	Label id

5.2 Using ADLINK Metadata

The Gstreamer element stream provides a simple way to get ADLINK metadata with `gst_buffer_get_ad_batch_meta`.

```
GstAdBatchMeta *adbatchmeta = gst_buffer_get_ad_batch_meta(buf);
```

NOTE: In this release version, the utility `gst_buffer_get_ad_batch_meta(GstBuffer* buffer)` is not yet available for use. In samples, ex_getAdMetadata.cpp, line 27 illustrates how to create a `gst_buffer_get_ad_batch_meta(GstBuffer* buffer)` for assessing GstAdBatchMeta.

Once you get the ADLINK metadata pointer from a buffer, you can directly get/set the content of the data inside.

Set Metadata

```
VideoFrameData video_info;
video_info.stream_id = "from-dumper-b5d84236-a23d-49fc-a574-e0cd944490bb";
video_info.frame_id = frame_counter;
video_info.timestamp = GetDigitUTCTime();
video_info.width = 800;
video_info.height = 600;
video_info.depth = 8;
video_info.channels = 3;
video_info.device_idx = 0;
video_info.is_compress = false;
adbatchmeta->batch.frames.push_back(video_info);
```

Get Metadata

```
frame_vec_size = meta->batch.frames.size();
if(adbatchmeta->batch.frames.size() > 0)
    classification_n = meta->batch.frames[0].class_results.size();
```

If the `AdBatch` metadata frame information exists, information like the frame vector size or the number of the classification results can be gotten directly.

6 Integrating the GStreamer Plugin

GStreamer includes the libgstapp plugin containing the `appsink` and `appsrc` elements to interact with the application. `appsink` is used to allow the application to get access to raw buffer data and `appsrc` is used to allow the application to feed buffers to the pipeline. Refer to GStreamer tutorials for more information on how to establish communication with `appsink` and `appsrc`.

Access to `appsink` and `appsrc` is through `VideoCapture` and `VideoWriter` from the OpenCV wrapper. We can directly provide the pipeline with `appsink` to `VideoCapture` for retrieving frames, and provide the pipeline with `appsrc` to send the frame into the pipeline from the application. This wrapper is much more simple for those who want to use algorithms that GStreamer does not provide, like motion extraction, video content analytics, or image saliency calculation.

Refer to the EVA SDK installation path /samples folder for more information on Python and C++ application sample code, compiling processes described in readme.md, and for how to build the sample code.

6.1 Method 1

To grab the frame from the pipeline with `appsink`, the constructor provided by `VideoCapture` requires two signatures, the pipeline string and the API preference (enum cv::CAP_GSTREAMER). To grab the frame from the `v4l2src` element of the pipeline, provide the pipeline definition to `VideoCapture` as in the following example.

appsink

```
VideoCapture cap("v4l2src ! videoscale ! video/x-raw, width=1024, height=768 ! videoconvert
! appsink", CAP_GSTREAMER);
```

The frame can then be captured via OpenCV.

VideoCapture

```
Mat frame;
while(true)
{
    cap.read(frame);
    // do your process ...
}
```

To send the frame to the pipeline with `appsrc`, the function provided by the `VideoWriter` requires the pipeline string, the API preference, and other parameters like fps, frame size, and color flag. The signatures are required for the pipeline caps filter settings. To send the frame data to the pipeline with `appsrc`, the pipeline definition must be provided to `VideoWriter`, as in the following example.

appsrc

```
cv::VideoWriter writer;
writer.open("appsrc ! videoconvert ! video/x-raw, format=BGR, width=640, height=480,
framerate=30/1 ! clockoverlay ! ximagesink sync=false", CAP_GSTREAMER, 0, 30, cv::Size(640,
480), true);
```

The target pipeline settings include, frame size (640x480), frame rate (30), and color format that fits the OpenCV default BGR color format. The pipeline then shows the frame in xwindows via the `ximagesink` element. Feed the frame into the pipeline by writing it directly, as in the following example.

VideoWriter

```
writer.write(frame);
```

The example code below shows the combination of the pipeline using `appsink` and `appsrc` to read a frame from the v4l2 pipeline and resizing the frame to simulate the custom algorithm process, then passing the resulting frame into the pipeline.

Combined Example

```
#include "opencv2/opencv.hpp"
#include <iostream>
#include <stdio.h>
#include <thread>
#include <chrono>

using namespace cv;
using namespace std;
int main(int, char**)
{
      Mat frame;

      VideoCapture cap("v4l2src ! videoscale ! video/x-raw, width=1024, height=768 ! videoconvert ! appsink", CAP_GSTREAMER);

      int deviceID = 0;
      int apiID = cv::CAP_ANY;

      cap.open(deviceID + apiID);

      if (!cap.isOpened())
      {
          cerr << "ERROR! Unable to open camera\n";
          return -1;
      }

      cv::VideoWriter writer;
      writer.open("appsrc ! videoconvert ! video/x-raw, format=BGR, width=640, height=480, framerate=30/1 ! clockoverlay ! ximagesink sync=false", CAP_GSTREAMER, 0, 30, cv::Size(640, 480), true);
      if (!writer.isOpened())
      {
          printf("=ERR= can't create writer\n");
          return -1;
      }

      //--- GRAB AND WRITE LOOP
      cout << "Start grabbing" << endl;

      for (;;)
      {
          cap.read(frame);
          if (frame.empty())
          {
              cerr << "ERROR! blank frame grabbed\n";
              break;
          }
          cv::resize(frame,frame,Size(640,480)); // do some image process here ...
          writer.write(frame);

          this_thread::sleep_for(chrono::milliseconds(1000));
      }
      return 0;
 }
```

6.2 Method 2

OpenCV provides a convenient way for developers wanting to utilize their own API, algorithm, or unique processing. Based on the examples in Method 1, another pipeline in the thread can be created to request user padding frame data to overlay clock information on the top-left of the frame via the GStreamer `clock overlay` element.

pipeline thread

```
thread pipethread(establish_appsrc_appsink_pipeline);
```

The `establish_appsrc_appsink_pipeline` function builds the pipeline: `appsrc ! clockoverlay ! videoconvert ! appsink`, shown in the code fragment below.

establish_appsrc_appsink_pipeline

```
1.static void establish_appsrc_appsink_pipeline()
2.{
3.    /* init GStreamer */
4.    gst_init (NULL, NULL);
5.    loop = g_main_loop_new (NULL, FALSE);
6.
7.    /* setup pipeline */
8.    pipeline = gst_pipeline_new ("pipeline");
9.    appsrc = gst_element_factory_make ("appsrc", "source");
10.   clockoverlay = gst_element_factory_make("clockoverlay", "clockoverlay");
11.   conv = gst_element_factory_make ("videoconvert", "conv");
12.   appsink = gst_element_factory_make ("appsink", "appsink");
13.
14.   /* setup */
15.   g_object_set(G_OBJECT (appsrc),
16.                "caps",
17.                gst_caps_new_simple("video/x-raw", "format", G_TYPE_STRING, "BGR",
18.                                    "width", G_TYPE_INT, 640, "height", G_TYPE_INT,
19.                                    480, "framerate", GST_TYPE_FRACTION,30,1,NULL),
20.                    NULL);
21.   gst_bin_add_many (GST_BIN (pipeline), appsrc, clockoverlay, conv, appsink, NULL);
22.   gst_element_link_many (appsrc, clockoverlay, conv, appsink, NULL);
23.
24.   /* setup appsrc */
25.   g_object_set (G_OBJECT (appsrc), "stream-type", 0,
26.                     "format", GST_FORMAT_TIME, NULL);
27.   g_signal_connect (G_OBJECT (appsrc), "need-data",
28.                         G_CALLBACK (cb_need_data), NULL);
29.
30.   /* setup appsink */
31.   g_object_set (G_OBJECT(appsink), "emit-signals", TRUE, NULL);
32.   g_signal_connect (appsink, "new-sample", G_CALLBACK (new_sample), NULL);
33.
34.   /* play */
35.   gst_element_set_state (pipeline, GST_STATE_PLAYING);
36
37.   while(true)
38.   {
39.     this_thread::sleep_for(chrono::milliseconds(10));
40.   }
41.
42.   free_appsrc_appsink_pipeline();
43.}
```

Refer to the GStreamer tutorials for more information on how to build the pipeline. Here the `appsrc` and `appsink` signal properties connect through the GObject API in line 27 and 32.

Lines 25 and 26 set the `stream-type` property to push mode. Line 27, hooks the `cb_need_data` callback function to `need-data` to wait for the `appsrc` notification to feed the data and then push it to `appsrc`.

cb_need_data

```
static void cb_need_data(GstElement *appsrc, guint unused_size, gpointer user_data)
{
    // ......
    // code omit

    memcpy((guchar *)map.data, grabframe.data, gst_buffer_get_size(buffer));

    // ......
    // code omit
        g_signal_emit_by_name (appsrc, "push-buffer", buffer, &ret);
 }
```

Once the function is called back, the data for `appsrc`'s buffer can be padded and then the signal called to push the data to `appsrc`.

Similarly, line 31 in `establish_appsrc_appsink_pipeline` sets the `appsink` property `emit-signals` to ejection mode. Line 32, hooks the `new_sample` callback function to wait for the notification to access the output frame data sample.

new_sample

```
static GstFlowReturn new_sample(GstElement *sink, gpointer *udata)
{
    GstSample *sample;

    g_signal_emit_by_name (sink, "pull-sample", &sample);
    if (sample)
    {
        // ......
        // code omit
        memcpy(processedframe.data, (guchar *)map.data, gst_buffer_get_size(buffer));

        gst_sample_unref (sample);
        return GST_FLOW_OK;
    }
    return GST_FLOW_ERROR;
 }
```

As long as `appsink` indicates the sample is ready and accessible, the data can be gotten from `appsink`'s buffer.

Relative to Method 1, Method 2 is provided for those who required leverage to GStreamer's elements, like clock overlay, to process the frame data and then return to the application.

Both Method 1 and Method 2 have introduced effective ways to integrate custom applications with GStreamer. For more information on the usage of `appsrc` and `appsink`, refer to the GStreamer tutorials.

6.3 Python Method

This section describes how to integrate python code with EVA with an example showing how to establish a python application and then involve the python plugin in this application. Refer to Chapter 3 for how to modify the python plugin of the translator.

6.3.1 Python Application

In pipeline_app.py, first create a thread to generate the pipeline in `establish_thread_pipeline`.

```
def establish_thread_pipeline():
    print('Start establish pipeline.')
    # GStreamer init and declare the pipeline
    Gst.init(sys.argv)
    pipeline = Gst.Pipeline().new("example-pipeline")

    # Start to declare the elements
    ## element: videotesetsrc
    src = Gst.ElementFactory.make("videotestsrc", "src")
    src.set_property("pattern", 18)
    ## element: capsfilter
    filtercaps = Gst.ElementFactory.make("capsfilter", "filtercaps")
    filtercaps.set_property("caps", Gst.Caps.from_string("video/x-raw, format=BGR, width=640, height=480"))
    ## element: admetadebuger
    debuger = Gst.ElementFactory.make("admetadebuger", "debuger")
    debuger.set_property("type", 1)
    debuger.set_property("id", 187)
    debuger.set_property("class", "boy")
    debuger.set_property("prob", 0.876)
    debuger.set_property("x1", 0.1)
    debuger.set_property("y1", 0.2)
    debuger.set_property("x2", 0.3)
    debuger.set_property("y2", 0.4)
    ## element: appsink - for console out to verify debuger
    dumper = Gst.ElementFactory.make("admetadumper", "dumper")
    ## element: videoconvert - for console out to verify debuger
    videoconvert = Gst.ElementFactory.make("videoconvert", "videoconvert")
    ## element: appsink
    sink = Gst.ElementFactory.make("appsink", "sink")
    sink.set_property('emit-signals', True)
    sink.connect('new-sample', new_sample, None)
    ### elements
    pipeline_elements = [src, filtercaps, debuger, dumper, videoconvert, sink]

    establish_pipeline(pipeline, pipeline_elements)

    bus = pipeline.get_bus()

    # allow bus to emit messages to main thread
    bus.add_signal_watch()

    # Start pipeline
    pipeline.set_state(Gst.State.PLAYING)

    loop = GLib.MainLoop()

    bus.connect("message", on_message, loop)

    try:
        print("Start to run the pipeline.\n")
        loop.run()
    except Exception:
        traceback.print_exc()
        loop.quit()

    # Stop Pipeline
    pipeline.set_state(Gst.State.NULL)
    del pipeline
```

```
59.     print('pipeline stopped.\n')
```

Similar to the example above, follow these steps in creating the pipeline in C/C++:

1. **Initialize and declare the pipeline.**

   This is the essential step while creating the pipeline at the beginning in lines 4 and 5.

2. **Create each element used in the pipeline.**

   For example, create the source element `videotestsrc` from the factory in line 9.

3. **Set values to an element's properties if necessary.**

   After creating the elements of the pipeline, some elements require setting a property value. For example, in line 15, `admetadebuger` (an element used to test adding the metadata) is created and required to set the pseudo inference data to it. Lines 16 to 23 show how to add those values to the element admetadebuger.

4. **Set callback function to element signal if necessary.**

   Similarly, some elements required setting a callback function, like `appsink` in lines 28 to 31. `appsink` requires setting its notification mode to true if new data exists (line 30). If the data is ready, `appsink` will call the application function callback connected in line 31.

5. **Add and link the elements.**

   After all the elements are set, line 35 adds and links each element one by one in `establish_pipeline`.

6. **Deal with the pipeline message.**

   Bus is the layer dealing with the messages between the application and the pipeline. In line 37, we get the bus from the pipeline and allow the bus to emit messages to the loop thread and then connect the callback function to the bus for handling the message from the pipeline in line 47.

7. **Start the pipeline.**

   When all the elements and pipeline bus are ready, set the pipeline status to `PLAYING` (line 43) and start the loop to run the pipeline (line 51).

8. **Stop the pipeline**

   When the pipeline terminated, it must set the pipeline status to `NULL` and release all the resources that the pipeline created in lines 57 and 58.

This example will copy the image data to the queue and save it to a file. The callback function set for `appsink` will copy the stream data (image data, in this example) to the queue (line 39). Then the main thread will check the queue for data. If image data exists, the image will be saved to the current working directory.

6.3.2 Python Plugin

There is a modified plugin which is used to retrieve the metadata and draw the detection result. In plugin_sample.py, `chainfunc()` is used to get images and metadata from the buffer (see Chapter 2 to retrieve the detection metadata like adyolo_sample.py). The difference in this example is to draw the box information onto the image in `draw_boxes()`. Users can replace this function with post-processing or other custom-designed algorithm.

6.3.3 Python Plugin used in Python Application

Once the python plugin file is put in the GST_PLUGIN_PATH, the user can use it directly in a python application. For example, pipeline_app_call_python_plugin.py is provided to demonstrate the usage of this plugin. This is very similar to the example pipeline_app.py describe above. The difference between these two examples is that the input source is changed to `filesrc` to provide the data, and an additional inference element is added.

The video source is required to separate the video and audio in line 5 and then to encode it with the relative decoder element like `avdec_h264` in line 9. Between the demux and decoder is the queue element (line 7) to queue the source data waiting for decoding processing.

The `videoconvert` element (lines 11 and 29) is used to transfer the specific stream into a compatible format. `videoconvert` is commonly used in transferring video streams.

```
1. ## element: filesrc
2.     src = Gst.ElementFactory.make("filesrc", "src")
3.     src.set_property("location", "./face.mp4")
4.     ## element: qtdemux
5.     demux = Gst.ElementFactory.make("qtdemux", "demux")
6.     ## element: queue
7.     queue = Gst.ElementFactory.make("queue", "queue")
8.     ## element: avdec_h265
9.     decoder = Gst.ElementFactory.make("avdec_h264", "decoder")
10.     ## element: videoconvert
11.     convert1 = Gst.ElementFactory.make("videoconvert", "convert1")
12.     ## element: adrt
13.     adrt = Gst.ElementFactory.make("adrt", "adrt")
14.     adrt.set_property("model", "facemask_tx2.engine")
15.     adrt.set_property("scale", 0.0039)
16.     adrt.set_property("mean", "0 0 0")
17.     adrt.set_property("device", 0)
18.     adrt.set_property("batch", 1)
19.     ## element: adtranslator
20.     translator = Gst.ElementFactory.make("adtranslator", "translator")
21.     translator.set_property("topology", "yolov3")
22.     translator.set_property("dims", "1,24,13,13,1,24,26,26,1,24,52,52")
23.     translator.set_property("input_width", 416)
24.     translator.set_property("label", "mask.txt")
25.     translator.set_property("engine-type", 2)
26.     ## element: adlink_plugin_sample
27.     drawer = Gst.ElementFactory.make("adlink_plugin_sample", "drawer")
28.     ## element: videoconvert
29.     convert2 = Gst.ElementFactory.make("videoconvert", "convert2")
30.     ## element: ximagesink
31.     sink = Gst.ElementFactory.make("ximagesink", "sink")
32.
33.     ### elements
34.     pipeline_elements = [src, demux, queue, decoder, convert1, adrt, translator,
drawer, convert2, sink]
```

`adrt` is the NVidia inference element to let the user load the optimized NVidia TensorRT model. Properties such as `model`, `scale`, `mean`, `device`, and `batch` (lines 14 to 18) can be found in the user's manual. These properties are required to be set while loading the model `facemask_tx2.engine`. (The model can be changed and have different properties depending on the model architecture.)

`adtranslator` is the translator element to decode the inference output blob into metadata. This element is designed to interpret the blob to human-readable data based on the model output format. Properties such as `topology`, `dims`, `input width`, `label`, and `engine-type` can be found in user's manual. Different models require different properties, like `adrt`.

In line 27, the generated plugin is used. Once the python plugin is set, we can directly use it as the other default element does. Simply create it and set its property if required. Then add and link the pipeline elements one after another. There is no difference between this custom python element and other elements.

The consuming sink element here is `ximagesink` (line 31) used in Linux based operating systems. Different operating systems use different sink elements like Windows might use `d3dvideosink` or `glimagesink`.

One thing to note about the element `qtdemux` in line 5, this element requires dynamic linking the pad to the next element. Use the `connect` element to set the dynamic link callback function (line 8 below). Once the pipeline starts to run, the `qtdemux` element will set the video pad to its next element.

```
1. def link_element(pipeline, pipeline_elements):
2.     ## Link element one by one.
3.     for i in range(len(pipeline_elements) - 1):
4.         if pipeline_elements[i].name != "demux":
5.             pipeline_elements[i].link(pipeline_elements[i + 1])
6.         else:
7.             if i+1 < len(pipeline_elements) - 1:
8.                 pipeline_elements[i].connect("pad-added", demuxer_dynamic_callback,
pipeline_elements[i+1])
```

Other steps are the same as the pipeline_app.py example. There is another example commented out that uses `v4l2src` (use `v4l2src` on Ubuntu, replace with `ksvideosrc` on Windows.) in pipeline_app_call_python_plugin.py.

Safety Instructions

Read and follow all instructions marked on the product and in the documentation before you operate your system. Retain all safety and operating instructions for future use.

- Please read these safety instructions carefully.
- Please keep this User‘s Manual for later reference.
- Read the specifications section of this manual for detailed information on the operating environment of this equipment.
- When installing/mounting or uninstalling/removing equipment, turn off the power and unplug any power cords/cables.
- To avoid electrical shock and/or damage to equipment:
  - Keep equipment away from water or liquid sources.
  - Keep equipment away from high heat or high humidity.
  - Keep equipment properly ventilated (do not block or cover ventilation openings).
  - Make sure to use recommended voltage and power source settings.
  - Always install and operate equipment near an easily accessible electrical socket-outlet.
  - Secure the power cord (do not place any object on/over the power cord).
  - Only install/attach and operate equipment on stable surfaces and/or recommended mountings.
  - If the equipment will not be used for long periods of time, turn off and unplug the equipment from its power source.
- Never attempt to fix the equipment. Equipment should only be serviced by qualified personnel.

Getting Service

Ask an Expert: http://askanexpert.adlinktech.com

ADLINK Technology, Inc.
Address: 9F, No.166 Jian Yi Road, Zhonghe District
New Taipei City 235, Taiwan
Tel: +886-2-8226-5877
Fax: +886-2-8226-5717
Email: service@adlinktech.com

Ampro ADLINK Technology, Inc.
Address: 5215 Hellyer Avenue, #110, San Jose, CA 95138, USA
Tel: +1-408-360-0200
Toll Free: +1-800-966-5200 (USA only)
Fax: +1-408-360-0222
Email: info@adlinktech.com

ADLINK Technology (China) Co., Ltd.
Address: 300 Fang Chun Rd., Zhangjiang Hi-Tech Park, Pudong New Area
Shanghai, 201203 China
Tel: +86-21-5132-8988
Fax: +86-21-5132-3588
Email: market@adlinktech.com

ADLINK Technology GmbH
Address: Hans-Thoma-Straße 11
D-68163 Mannheim, Germany
Tel: +49-621-43214-0
Fax: +49-621 43214-30
Email: germany@adlinktech.com

Please visit the Contact page at www.adlinktech.com for information on how to contact the ADLINK regional office nearest you.