Edge Video Analytics SDK Programming Guide

PM Review

Manual Rev.: 0.2 Preliminary
Revision Date: June 31, 2020
Part Number: 50-1Z322-1000

Leading EDGE COMPUTING

Preface

Revision History

Revision	Description	Date	By
0.1	Preliminary release for Edge Video Analytics SDK Programming Guide	2020-06-10	RA
0.2	2nd draft from R&D	2020-07-31	AT

Table of contents

PREFACE 2

TABLE OF CONTENS 3

INTRODUCTION 5

HOW TO DEVELOP ELEMENTS/PLUGINS WITH C 6

Class Declaration 6

Constructors and Deconstructors 7

Object Life Cycle 8

Class Implementation 8

Sink and Source Pad association 10

Overriding GstVideoFilter transform_frame_ip 10

Plugin Registration 12

HOW TO DEVELOP ELEMENTS/PLUGINS WITH PYTHON 13

Import Glib module 13

Class Declaration 13

Class Implementation 13

Initialize class metadata 13

Initialize class instance 14

Sink and src pad association 14

Override set and get property function 15

Implement chain function 15

Register python element 15

Install a Python element 16

PYTHON SAMPLE TO INTERPRET INFERENCE RESULT AS YOLOV3 BOX DETECTION 17

Explain python sample code 17

Draw boxes in image 19

How to custom translate part of code 20

HOW TO USE ADLINK METADATA 21

ADLINK metadata architecture 21
AdBatch Structure 22
DeviceInfoData Structure 22
VideoFrameData Structure 23
ClassificationResult Structure 23
DetectionBoxResult Structure 24
SegmentResult Structure 24

Using ADLINK metadata 25

HOW TO INTEGRATE THE GSTREAMER PLUGIN WITH YOUR CODE 26

Method 1 26

Method 2 28

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 Video 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.

- How to develop 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. Again, so some prior knowledge of object-oriented concepts will be helpful to understanding this information.

- How to develop 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.

- How to integrate GStreamer with your code includes examples of how to integrate code with Gstreamer to give a clearer understanding of how GStreamer is used in applications.

How to develop 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 how to implement custom elements.

註解 [annie1]:
Hi Ron,
RD reply: “Only one class”
So, I modify it. Please help review it again.
Thanks.

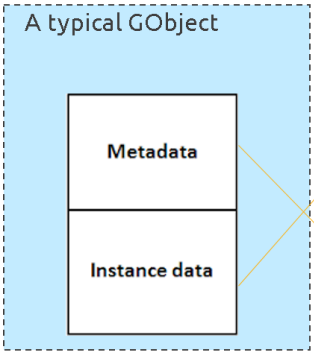
註解 [annie2]:
Hi Ron,
RD reply: “Suggest creating a new sentence for simplification for the technical readers.”
Please help modify again.

Class Declaration

In GObject, class declaration is quite different between C++, C#, JAVA, and other object-oriented programing 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 the GObject, first describe the class metadata and then the instance data belonging to the class, as shown in the following example (see adaifiltertemplate.h and adaifiltertemplate.cpp).

註解 [annie3]:
Hi Ron,
Added by RD.
Thanks.




```
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.

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

Register to GObject

```
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.

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.

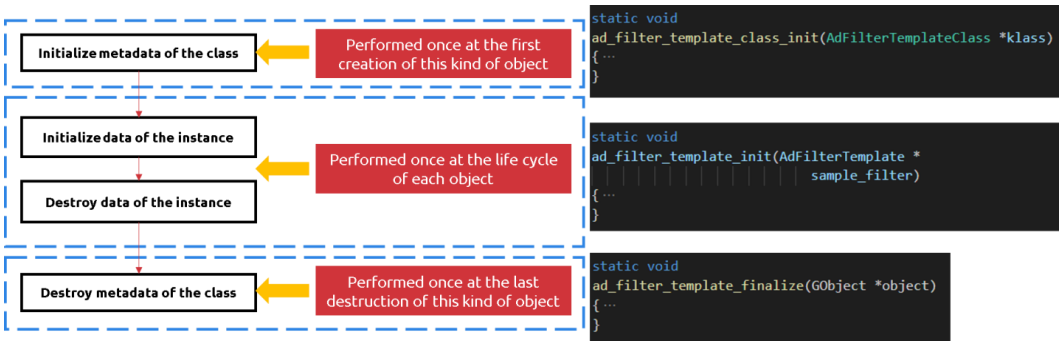
Constructors and Deconstructors

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.

註解 [annie4]: RD reply. Constructor and deconstructor won't have any "connection" in OO. Suggest using original sentence: In the GObject, there's no clear correspond constructor to deconstructor.

Object Life Cycle

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



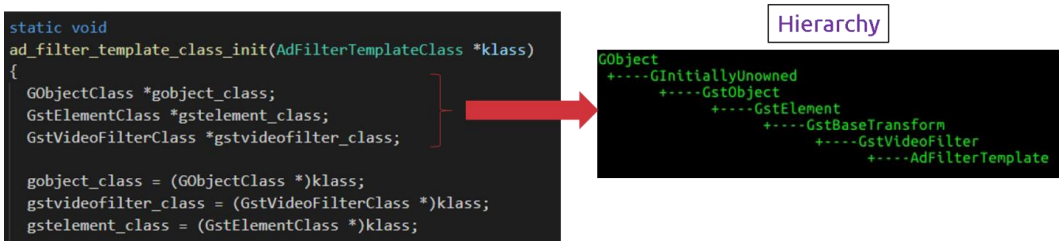
Class Implementation

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

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:



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;
```

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));
```

Sink and Source Pad association

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.

define pad

```
  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 }")));
```

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

add pad

```
  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));
```

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.

transform_ip override

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

ad_filter_template_transform_frame_ip

```
1.  static GstFlowReturn
2.  ad_filter_template_transform_frame_ip(GstVideoFilter *filter, GstVideoFrame *frame)
3.  {
4.    AdFilterTemplate *sample_filter = AD_FILTER_TEMPLATE(filter);
5.    GstMapInfo info;
6.    Mat output_image;
7.    int filter_type;
```

```
8.    int edge_threshold;
9.    gst_buffer_map(frame->buffer, &info, GST_MAP_READ);
10.   ad_filter_template_initialize_images(sample_filter, frame, info);
11.   AD_FILTER_TEMPLATE_LOCK(sample_filter);
12.   filter_type = sample_filter->priv->filter_type;
13.   edge_threshold = sample_filter->priv->edge_value;
14.   AD_FILTER_TEMPLATE_UNLOCK(sample_filter);
15.   if (filter_type == 0)
16.   {
17.     GST_DEBUG("Calculating edges");
18.     Canny((*sample_filter->priv->cv_image), output_image,
19.           edge_threshold, 255);
20.   }
21.   else if (filter_type == 1)
22.   {
23.     GST_DEBUG("Calculating black&white image");
24.     cvtColor((*sample_filter->priv->cv_image), output_image, COLOR_BGR2GRAY);
25.   }
26.   if (output_image.data != NULL)
27.   {
28.     GST_DEBUG("Updating output image");
29.     ad_filter_template_display_background(sample_filter, output_image);
30.   }
31.   gst_buffer_unmap(frame->buffer, &info);
32.   return GST_FLOW_OK;
33. }
```

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.

"gst_buffer_map" and "gst_buffer_unmap" deal with the mapping tasks from frame data to “info”. "map" and "unmap" also deal 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.

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.

plugin registration

```
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 .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.

How to develop 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.

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([])
```

Class Declaration

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

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

Class Implementation

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
      ),
      ...
    }
    ...
```

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__()

      ...
```

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')
```

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
      ...
```

Implement chain function

When a sink pad pushes the buffer, then 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)
```

Register python element

You need to register the Python element after implementing the element class, so 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.

Install a Python element

Usually, GStreamer scans plugins under GST_PLUGIN_PATH. However, Python elements must be installed under the Python GST_PLUGIN_PATH folder. For example:

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

Python sample to interpret inference result as 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 format of inference result. Application needs to change code to interpret inference results after change different models. This causes the inference application will have a high dependency on the deep learning model.

The EVA SDK separates the inference application into two parts, one is the inference part and the other one is the translation part. The translation part must follow the inference part. For the same inference purpose such as the detection, the user can easily swap the inference part such as OpenVINO or TensorRT and does not need to modify the translation part. For different models, the user can easily swap the translation part according to the deep learning model and do not need to modify the inference part.

The EVA SDK provides the inference part's application that can be executed with most of the inference models. The user can focus on their domain that how to translate the inference results into the human-readable format. Here is a Python sample code on how to integrate the translation code with the EVA SDK inference app.

Explain python sample code

Normal GStreamer element will send only one buffer that is image data. However, the inference element will send a buffer list, the first index of the list is image data, and the second index of the list is inference data. Therefore, the translation element needs to receive buffer a list and extract both image data 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:
    ...
```

Here is a YoloV3 interpret sample code. The YoloV3 is a special deep learning architecture because it will have multiple output blobs that will have different target box sizes for each output blob. Therefore, the sample needs to separate inference data into multiple buffers depend on the output blob size. The default YoloV3 will have three output blobs and the YoloV3-Tiny will have two output blobs. The buffer size will change on the batch number, the class number, and the 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 (the format is 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))

    ...
```

After getting 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:
      ...
```

Separate 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. Here will not explain how to the parse YoloV3 format buffer.

For more details of the author's analyze, refer to https://github.com/pjreddie/darknet/blob/f6d861736038da22c9eb0739dca84003c5a5e275/src/yolo_layer.c#L275

Or refer to the parse sample code

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

Draw boxes in image

After obtaining those boxes, the program needs to draw those 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. And then the data can be used like an image and can be used 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))
```

How to custom translate part of code

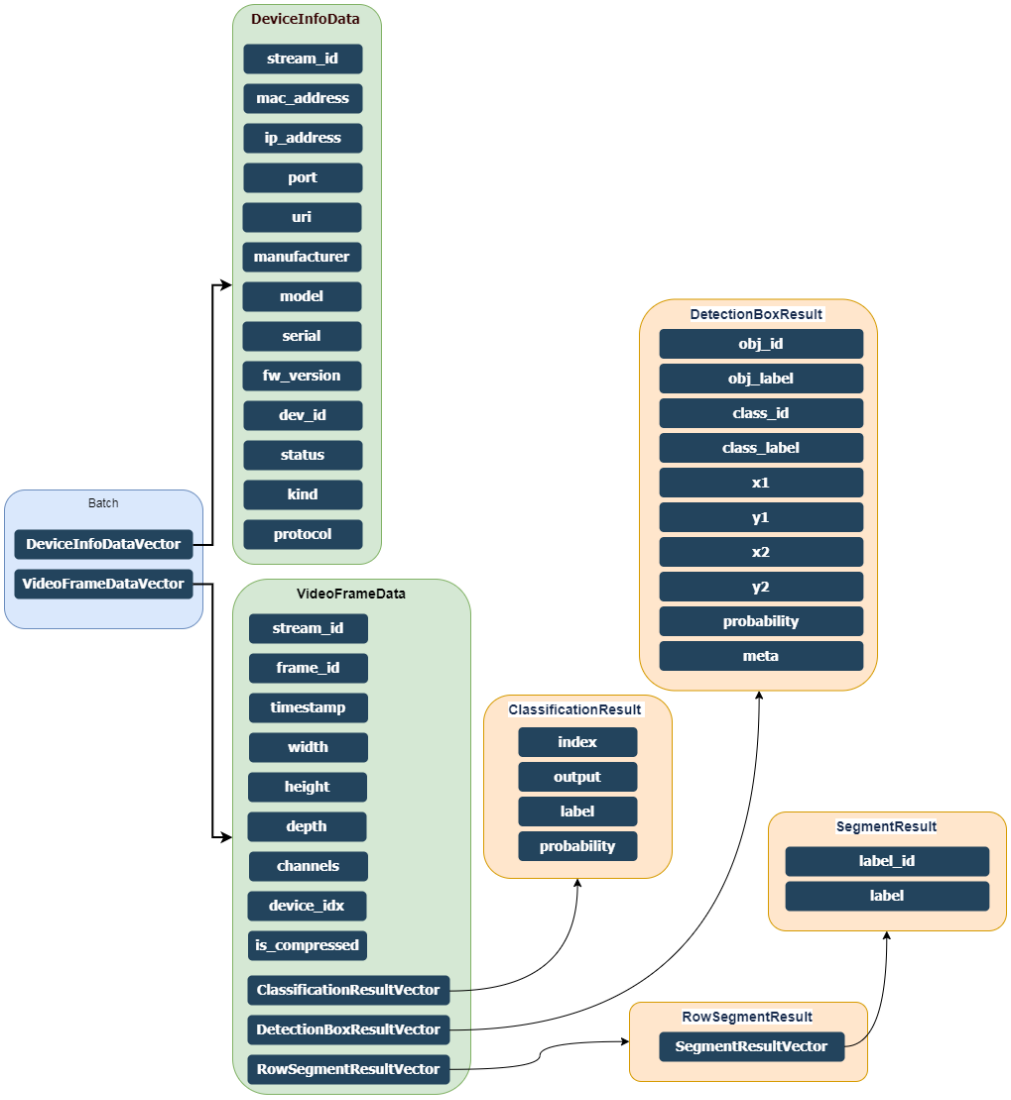
There are only two main parts to be changed. The first part is to define your own properties because the properties of this sample code were used for the YoloV3 model. The different models will have their own required information. The second one is how to exact inference data because the YoloV3 has multiple output blobs. Need to divide a big buffer into multiple buffers. Maybe your model only has one output blob and it will be easier to extract image data. Finally, implement code to interpret blob buffer into human-readable data such as the classification, the detection box, or the segmentation. Each model has its output format.

After changing those parts, the user can integrate their own translation application with the EVA inference app

How to use ADLINK metadata

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 five different structures:

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

Each struct has its own items. The content of each metadata struct is described below.

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.

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

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

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

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

SegmentResult Structure

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

Using ADLINK metadata

In the Gstreamer element stream, we provide a simple way to get the ADLINK metadata using gst_buffer_get_ad_batch_meta from the buffer.

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

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

For example, setting the frame metadata:
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);
```

For example, getting the frame metadata:

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, we can directly get the information we want, like the frame vector size in the metadata and the number of the classification results.

How to integrate the GStreamer plugin with your code

GStreamer includes the libgstapp plugin containing the appsink and appsrc elements to deal with the interaction 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 the 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. To find the sample codes in the installed path of EVA SDK under Samples folder. This folder contains the python and C++ application example codes inside. The compile processes are described in readme.md under Samples folder. Refer it to build the sample codes.

註解 [annie5]:
Hi Ron,
RD reply: "It is not limited to use opencv wrapper to get/feed data buffer through GStreamer appsink/appsrc. Here we just use OpenCV wrapper, VideoCapture, for simplifying the instruction. Suggest to modify this sentence."
Please help modify. Thanks.

註解 [annie6]:
Hi Ron,
RD Reply.
"Here need to add some sample code compilation descriptions:"
Please help review again. Thanks.

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 (the 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 example

```
VideoCapture cap("v4l2src ! video/x-
raw, format=BGR, width=640, height=480, framerate=30/1 ! appsink", CAP_GSTREAMER);
```

Then, the frame can 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 to fulfill 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 example

```
cv::VideoWriter writer;
writer.open("appsrc ! videoconvert ! video/x-
raw, format=BGR, width=640, height=480, framerate=30/1 ! videoconvert ! ximagesink", CA
P_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 the xwindows via the ximagesink element. To feed the frame into the pipeline by writing it directly, follow this 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.

Combination 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 ! video/x-
    raw, format=BGR, width=1024, height=768, framerate=30/1 ! 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 ! videoconvert ! ximagesink"
    , 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;
}
```

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 element, “clock overlay”.

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

```
static void establish_appsrc_appsink_pipeline()
{
    /* init GStreamer */
    gst_init (NULL, NULL);
    loop = g_main_loop_new (NULL, FALSE);

    /* setup pipeline */
    pipeline = gst_pipeline_new ("pipeline");
    appsrc = gst_element_factory_make ("appsrc", "source");
    clockoverlay = gst_element_factory_make("clockoverlay", "clockoverlay");
    conv = gst_element_factory_make ("videoconvert", "conv");
    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.     g_main_loop_run (loop);
37.
38.     free_appsrc_appsink_pipeline();
39. }
```

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.

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

註解 [annie7]:
Hi Ron,
“RD reply: The original sentence want to explain that there is a property named “stream-type” and it should be set to “push mode” in Line 25 and 26.”
I modified this sentence, please help review it. Thanks a lot. ☺ .

註解 [annie8]:
Hi Ron,
I added “named”. If it cannot incorrect, please help modify it. Thanks.

cb_need_data

```
1.  static void cb_need_data(GstElement *appsrc, guint unused_size, gpointer user_data)
2.  {
3.      // ......
4.      // code omit
5.
6.      memcpy((guchar *)map.data, grabframe.data, gst_buffer_get_size(buffer));
7.
8.      // ......
9.      // code omit
10.     g_signal_emit_by_name (appsrc, "push-buffer", buffer, &ret);
11. }
```

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 the establish_appsrc_appsink_pipeline sets the appsink property named “emit-signals” to ejection mode. Line 32, hooks the callback function named “new_sample” to wait for the notification to access the output frame data sample.

註解 [annie9]: I added “named”. If it cannot incorrect, please help modify it. Thanks.

註解 [annie10]: I added “named”. If it cannot incorrect, please help modify it. Thanks.

new_sample

```
1.  static GstFlowReturn new_sample(GstElement *sink, gpointer *udata)
2.  {
3.      GstSample *sample;
4.
5.      g_signal_emit_by_name (sink, "pull-sample", &sample);
6.      if (sample)
7.      {
8.          // ......
9.          // code omit
10.         memcpy(processedframe.data, (guchar *)map.data, gst_buffer_get_size(buffer));
11.
12.         gst_sample_unref (sample);
13.         return GST_FLOW_OK;
14.     }
15.     return GST_FLOW_ERROR;
16. }
```

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.