# AUTOHIGHLIGHT ALGORITHM INTEGRATION

**ON TASMANIA**

## 1. Purpose

This document provides the overview of the Autohighlight algorithms (Engines) that needs to be integrated to Tasmania platform through Qualcomm's video analytics manager (VAM) framework. The Auto highlight algorithm is based on the captured audio and IMU data.

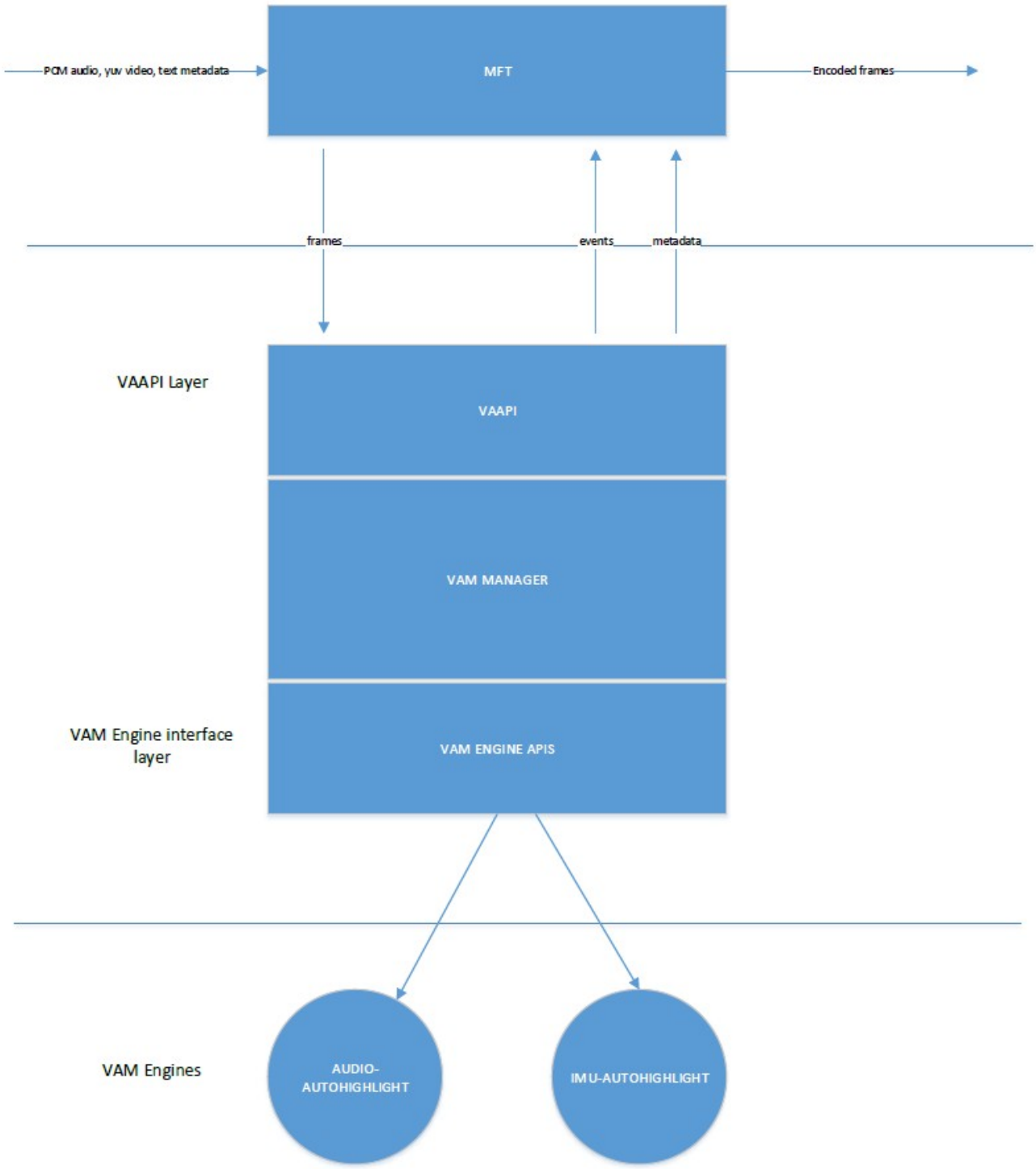
## 2. Requirements

- IMU Rate: 200Hz.
  Audio Rate: 48Khz, 2 channel, 32bit signed (this is the input) and Algorithm downsamples it to 24khz.
- IMU data is fed to the IMU algorithm every 1 sec and audio pcm data is fed to the Audio algorithm every 22msec. i.e. 1k samples every 22msec.

## 3. Overview

Media formatter(MFT) initializes and performs start/stop of VAM. IMU and PCM audio data are passed to the VAM through MFT.
VAM internally feeds these data to VAM engines(algorithm) integrated and passes on the events(Autohighlight) generated through registered
call-backs back to MFT. MFT takes care of muxing these events as part of MP4 in GPMF format.

**Figure 1:** VAM Framework.




General events that VAM Framework generates are:

1) Audio related events: `audio_auto_highlight_event`.
2) Sensor data related events: `imu_auto_highlight_event`.

Types of VAM Engines are:

1) **Audio VAM Engine:** The input for this engine is pcm audio frames, this generates events.
E.g.: Audio-Auto-Highlight Engine.

2) **Sensor Data VAM Engine:** The input for this engine is metadata (GYRO, ACCL…) as text frames, this generates events processing the input frames.
E.g.: IMU-Auto-Highlight Engine.

# 4. OVERVIEW OF VAM

The application layer intakes encoded video from the camera and configuration settings from the VASim_rule_config.json file and translates this information into C structure. This data is then passed to VAM through VAAPI. From there, it can be passed to VA engines (for example, object tracking and facial recognition) using the VAM engine API. Customers can also develop custom VA engines using the following VA utility SDKs:

- FastCV – Currently available
- SVCE – Currently unavailable, will be added in a later version of VAM
- VASS-ext – Currently unavailable, will be added in a later version of VAM

VAM produces encoded metadata frames and event data in the .json format. Using the VASim tool, customers can view a rendering of the output data on a PC.

# 5. PASSSING DATA TO VAM USING VAAPI

Following is the highlight data structure passed to VAM,
Highlight Data Structure:

```
typedef struct high_light
{
     uint32_t time; // highlight time
     uint32_t in_time; //event start time
     uint32_t out_time; //even end time
     int32_t  longitude;  //event location (1E7 format)
     int32_t  lattitude;  //event location (1E7 format)
     float  altitude; //event height (meters)
     uint32_t event_type; //4CCs: AIMU, AACD, FUSE, JUMP,
                          //MANL, etc.
     float  confidence; //0 to 100.0%
```

```
        float  score;
    } high_light;
```

QTI provides the following pseudocode template for passing data to VAM using VAAPI:

```
    vaapi_init();
    vaapi_start();
    vaapi_enroll_obj();

    // call back function registrations
    vaapi_register_audio_event_cb();
    vaapi_register_imu_event_cb();
    vaapi_register_metadata_cb();
    vaapi_register_frame_processed_cb ();

    //for(video frame input)
    vaapi_process();
    vaapi_stop();
    vaapi_deinit();
```

VAM Framework creates one engine instance for each of the algorithms that are integrated and each of them runs on their own thread. The frame data which VAM framework receives will be passed onto the engines and engines process the frame and generates the relevant events and metadata. These events then passed on to the upper layer through registered call-backs.

## vaapi_init:

This function begins VAM initialization. The `dyn_lib_path` structure contains the engine plug-in folder.

```
    int32_t vaapi_init(const vaapi_source_info *info, const char
*dyn_lib_path);
```

The `vaapi_source_info` data structure is as follows:

```
    struct vaapi_source_info
    {
        char data_folder[VAAPI_PATH_LEN]; //Path length is 512.

        vaapi_img_format img_format; //It contains one of
    //the following formats:
    /*   enum vaapi_img_format
        {
            vaapi_format_invalid = 0,
            vaapi_format_yv12,
            vaapi_format_nv12,
            vaapi_format_nv21,
            vaapi_format_YUVJ420P,
            vaapi_format_YUVJ422P,
            vaapi_format_YUVJ444P,
            vaapi_format_GRAY8,
            vaapi_format_RGB24
        }; */
```

```
        uint8_t frame_l_enable;
        uint32_t frame_l_width[3];
        uint32_t frame_l_pitch[3];
        uint32_t frame_l_height[3];
        uint32_t frame_l_scanline[3];

        uint8_t frame_s_enable;
        uint32_t frame_s_width[3];
        uint32_t frame_s_pitch[3];
        uint32_t frame_s_height[3];
        uint32_t frame_s_scanline[3];

        char is_test_mode;
    };
```

## vaapi_start:

This function starts the VAM framework, allocates the thread for each of the engine objects, each engine objects run on their own thread.

```
    int32_t vaapi_start();
```

## vaapi_enroll_obj:

This function enrols a single object for VAM. It should be called once per item.

```
    int32_t vaapi_enroll_obj( vaapi_event_type type,
vaapi_enrollment_info *enroll_info);
```

The `vaapi_enrollment_info` data structure is as follows:

```
    struct vaapi_enrollment_info
    {
        char id[VAAPI_UUID_LEN]; //FR: personID
        char display_name[VAAPI_NAME_LEN];
        vaapi_object_type type; //Type of the object
    /*  enum vaapi_object_type
        {
            vaapi_object_type_unknown = 0,   //default type,
         //should alwyas be replaced
            vaapi_object_type_people = 1,
            vaapi_object_type_vehicle = 2,
            vaapi_object_type_face = 3,
         }; */
        char img_id[VAAPI_UUID_LEN]; //FR: image id
        vaapi_img_format img_format;
        uint8_t *img_data[3];
        uint32_t img_width[3];
        uint32_t img_pitch[3];
        uint32_t img_height[3];
    };
```

## vaapi_register_audio_event_cb:

This function registers the VAM event call back function when an audio event is detected.

```
int32_t vaapi_register_audio_event_cb(vaapi_audio_event_cb_func func, void *usrData);
```

## vaapi_register_imu_event_cb:

This function registers the VAM event call back function when an imu event is detected.

```
int32_t vaapi_register_imu_event_cb(vaapi_imu_event_cb_func func, void *usrData);
```

## vaapi_register_frame_processed_cb:

This function registers the VAM frame process finished callback function. Callback happens once per frame.

```
int32_t vaapi_register_frame_processed_cb(vaapi_frame_processed_cb_func func, void* usrData);
```

## vaapi_process:

This function feeds one frame of data to VAM framework, which is passed onto the VAM Engine and it takes care of processing it and generating corresponding events and metadata.

```
int32_t vaapi_process(struct vaapi_frame_info *frame_info);
```

The `vaapi_frame_info` data structure is as follows:

```
struct vaapi_frame_info
{
    uint64_t time_stamp;

    uint8_t *frame_l_data[3];
    uint8_t *frame_s_data[3];

    uint32_t obj_count;
    vaapi_object *objects; //Object data structure is as
//follows:
/*   struct vaapi_object
     {
         uint32_t id;
         char display_name[VAAPI_NAME_LEN];
         vaapi_position pos; //Position is as follows:
     /*   struct vaapi_position
          {
              uint32_t x;
              uint32_t y;
              uint32_t width;  // 0 <= (X + Width) <= 10000
              uint32_t height; // 0 <= (Y + Height) <= 10000
          }; */
```

```
        vaapi_object_type type;
        uint8_t confidence;

        int64_t reserve[VAAPI_RESERVE_ITEM];
        char reserve_str[VAAPI_RESERVE_ITEM]....
    [VAAPI_NAME_LEN];
    }; */
};
```

**vaapi_stop:**

This function stops the VAM framework, cleans up the thread context.

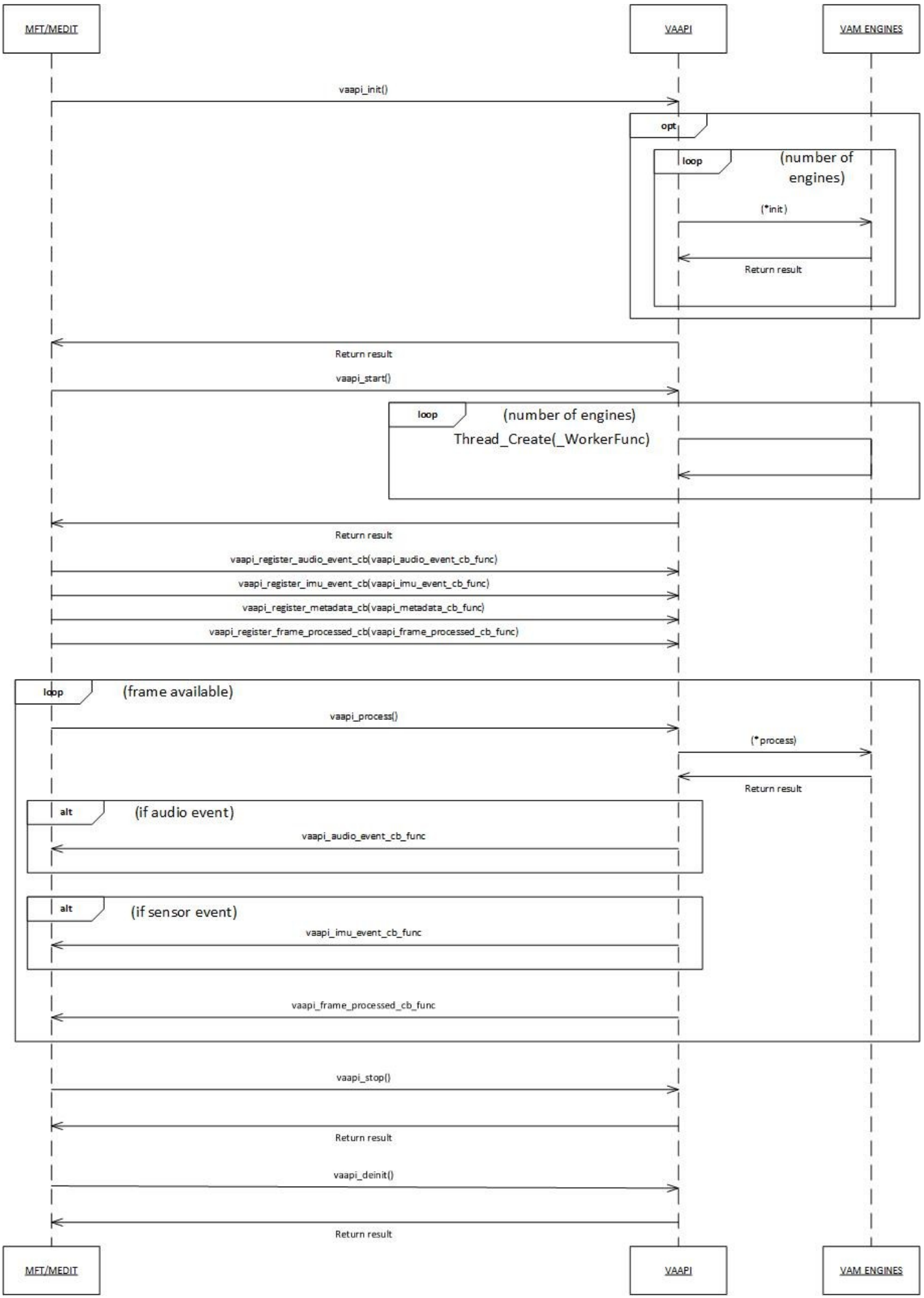
```
int32_t vaapi_stop();
```

**vaapi_deinit:**

This function deinitializes the VAM Engines/Algorithms.

```
int32_t vaapi_deinit();
```

## 6. Sequence Diagram


MFT/MEDIT
VAAPI
VAM ENGINES
vaapi_init()
opt
loop
(number of engines)
(*init)
Return result
Return result
vaapi_start()
loop
(number of engines)
Thread_Create(_WorkerFunc)
Return result
vaapi_register_audio_event_cb(vaapi_audio_event_cb_func)
vaapi_register_imu_event_cb(vaapi_imu_event_cb_func)
vaapi_register_metadata_cb(vaapi_metadata_cb_func)
vaapi_register_frame_processed_cb(vaapi_frame_processed_cb_func)
loop
(frame available)
vaapi_process()
(*process)
Return result
alt
(if audio event)
vaapi_audio_event_cb_func
alt
(if sensor event)
vaapi_imu_event_cb_func
vaapi_frame_processed_cb_func
vaapi_stop()
Return result
vaapi_deinit()
Return result
MFT/MEDIT
VAAPI
VAM ENGINES

# 7. Assumptions

Following assumptions are made based on the existing CV framework and proposed VAM framework:

- We use mft_wrapper.cpp and mft_wrapper_video.cpp for binding the VAM calls from MFT module (mft_main.c, mft_video.c).
- Fused autohighlight data structure given back to the media-formatter through registered call-backs.
- Autohighlights are saved as part of MP4 track in GPMF format.

# 8. References

| TITLE | AUTHOR |
|---|---|
| *CV_Document_v0.3.pdf* | *GoPro.* |
| *video_analytics_manager_developer_guide.pdf* | *Qualcomm Technologies, Inc.* |

# 9. Revision History

| Version | Date | Author | Details |
|---|---|---|---|
| V0.1 | 11-Aug-17 | Shrihari A | Initial Draft |
| V0.2 | 16-Aug-17 | Shrihari A | Internal review comments incorporated |
| V0.3 | 19-Aug-17 | Shrihari A | GoPro review comments incorporated |