

# 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

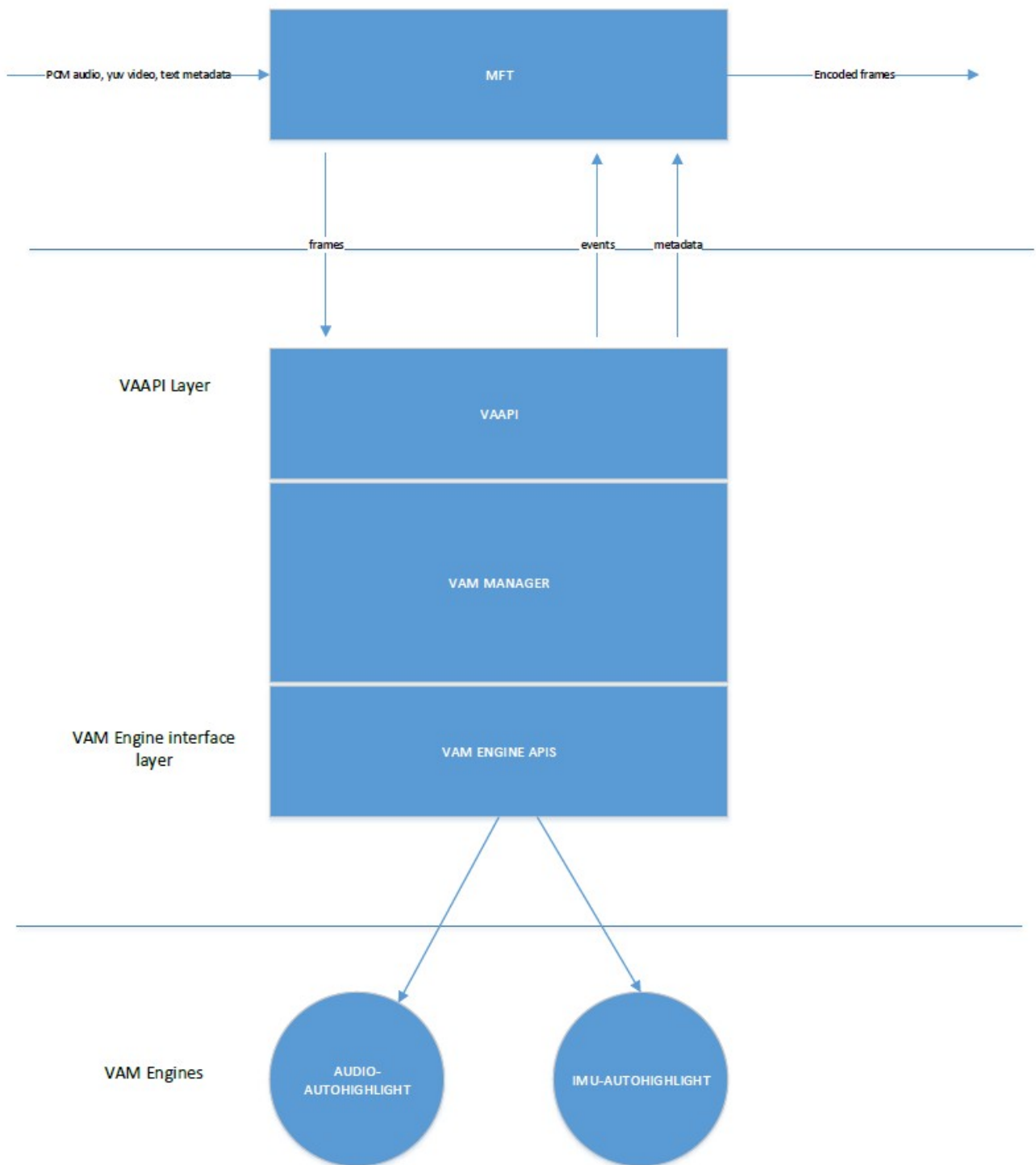
- IMU Rate: 200Hz.  
Audio Rate: 48Khz, 2 channel, 32bit signed (this is the input) and Algorithm downsamples it to 24khz.
- IMU data is fed to the IMU algorithm every 1 sec and audio pcm data is fed to the Audio algorithm every 22msec. i.e. 1k samples every 22msec.

## 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.PASSING 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);
```

#### **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 enrolls 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);
```

#### **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);
```

#### **vaapi\_stop:**

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

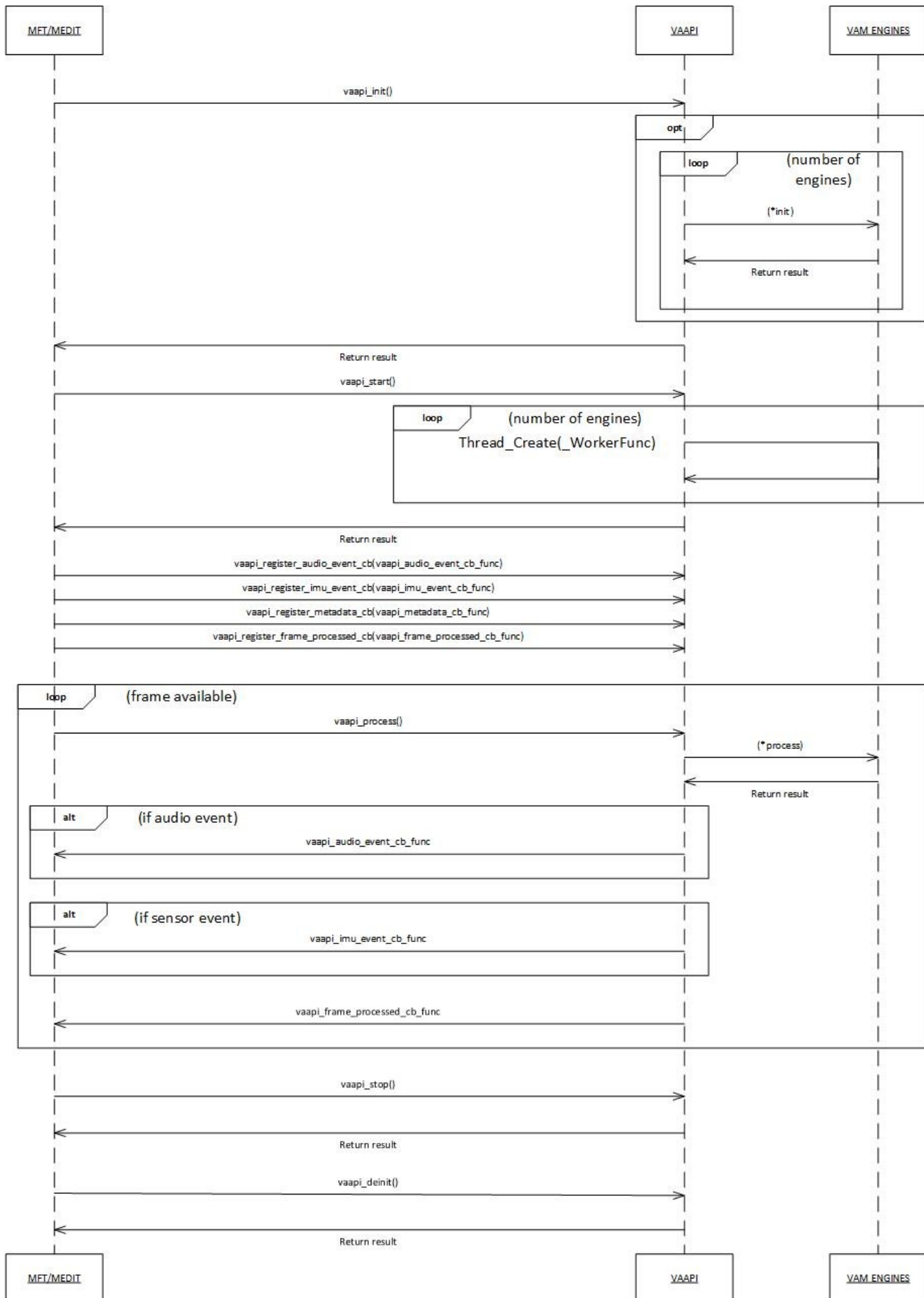
```
int32_t vaapi_stop();
```

#### **vaapi\_deinit:**

This function deinitializes the VAM Engines/Algorithms.

```
int32_t vaapi_deinit();
```

## 5.Sequence Diagram



## 6.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.

## 7.References

<b>TITLE</b>	<b>AUTHOR</b>
<i>CV_Document_v0.3.pdf</i>	<i>GoPro.</i>
<i>video_analytics_manager_developer_guide.pdf</i>	<i>Qualcomm Technologies, Inc.</i>

## 8.Revision History

<b>Versio n</b>	<b>Date</b>	<b>Author</b>	<b>Details</b>
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	Internal review comments incorporated