OMX Core Integration Guide
OpenCORE 2.0, rev. 1
Jan 21, 2009

References

1 *OpenMAX Integration Layer Application Programming Interface Specification.* Version 1.1.2, http://www.khronos.org/openmax/

2 OpenMAX Call Sequences. OpenCORE 2.0, rev. 1. http://android.git.kernel.org/?p=platform/external/opencore.git;a=summary

Table of Contents

List of Figures

1. Introduction

There are several ways to integrate a codec into the PV OpenCORE multimedia framework including as a compressed media I/O component, as a node, and as an OpenMAX component integrated into the OpenMAX codecs nodes that are part of the framework. Many codecs, especially those that include hardware acceleration, implement the OpenMAX IL interface making the OpenMAX interface the most straightforward method of integration in those cases.

One of the issues encountered in the integration process is the existence of multiple OMX cores. Even if all OMX components that come from different vendors are standard compliant, different vendors typically implement the same set of OpenMAX core methods (e.g. OMX_Init, OMX_Deinit, OMX_GetHandle, …). This poses a problem in terms of linking and executing the desired OMX core methods.

This document describes the details to help integration between the PV OpenCORE framework and OpenMAX IL 1.1 compliant components (encoders and decoders). While the OHA document "OpenMAX call sequences" describes the detailed interaction and call sequences between PV framework and OMX components, this document focuses on the integration of OMX core methods into the PV OpenCORE framework without causing conflicts with any other integrated OMX cores. Throughout this document the names PV framework and OpenCORE will be used interchangeably.

2. PV OMX test harness

2.1. General notes about the PV OMX test harness

Before integration of an OMX component begins, verification of the correct operation should be verified as a prerequisite. Using a set of conformance tests, the proper behavior of the component can be checked in isolation before introducing the additional complexity of the other components in the system. PV provides the PV OMX test harness for conformance testing of OMX components. .

The test harness **only allows one OMX core**, which may contain multiple OMX components, to be linked-in and tested at a time. In other words, the OMX core is tested in isolation at this step so there is no concern about co-existence of multiple OMX cores at this point.

2.2. PV OMX test harness location

The code for the PV OMX test harness are located in

```
.../codecs_v2/omx/omx_testapp/
```

for decoder components and

```
.../codecs_v2/omx/omx_testapp_enc
```

for encoder components.

2.3. Building the test harness with OMX plugins

Makefiles are provided to build the test harness including any OMX plugins. While the makefiles are setup to build the included test harness source code, they need to be modified to link in the OMX plugin library to be tested. These OMX plugin libraries must contain the standard OMX core methods such as OMX_init, OMX_Deinit, OMX_GetHandle, OMX_FreeHandle, etc.

The above mentioned makefiles are located in:

```
.../codecs_v2/omx/omx_testapp/build/makefile_omx_plugin
```

and in:

```
.../codecs_v2/omx/omx_testapp_enc/build/makefile_omx_plugin
```

2.4. Running test cases

The PV OMX test harness can be run from the command line. The documents provided in

```
.../codecs_v2/omx/omx_testapp/doc/...
.../codecs_v2/omx/omx_testapp_enc/doc/...
```

describe the PV OMX test harness command line arguments and test cases in elaborate detail. Once the tests in the test harness pass, the integration can proceed to the next step.

2.5. Handling partial compliance of OMX components

In some cases, an OMX component may not be fully compliant with the OpenMAX specification, but it is not feasible to correct the issues right away. For example, the component may not handle “OMX_UseBuffer” call, or it may not be capable of assembling partial input frames etc. In such cases, it may still be possible to integrate the component. However, the OMX component must inform the PV OpenCORE framework about its capabilities. This procedure is described in Section 3.2. of OHA document “OpenMAX call sequences”.

3. Building the OMX core library

3.1. OMX core methods

The OpenMAX Specification lists the following 9 methods that belong to the OMX core. All of the methods must be present in the library - although not all of them need to be fully supported. For example, if only OMX Base Profile is supported – the method OMX_SetupTunnel must be present in the OMX core library, but may simply return “OMX_ErrorNotImplemented”.

1)OMX_Init
2)OMX_Deinit
3)OMX_GetHandle
4)OMX_FreeHandle
5)OMX_ComponentNameEnum
6)OMX_GetComponentsOfRole
7)OMX_GetRolesOfComponent
8)OMX_SetupTunnel
9)OMX_GetContentPipe

3.2. Co-existence of multiple OMX cores

The main problem with the co-existence of multiple OMX cores – belonging to different vendors – is that all of them must implement the same set of OMX core methods. This can cause link issues they were simply statically linked. One simple (but very impractical) method to resolve linking issues due to this problem would be to rename all OMX core methods in each vendor OMX core library to enable proper linking.

PV framework implements the method of dynamic loading/linking of multiple OMX cores. The OMX core methods are called through a thin wrapper layer (provided by PV). This enables all OMX cores to keep the naming of OMX core methods, however – it imposes certain requirements on the OMX core library that is to be integrated. The requirements are described below.

3.3. Compiling the OMX core library

Because the PV OpenCORE framework is utilizing dynamic loading and linking to handle potentially multiple OMX cores simultaneously, the OMX core libraries must be built as shared objects, which typically means that the compiler flags must be set for position-independent code. It is fine to incorporate pre-built libraries (I.e., libraries built with an external build system) as long as it built in a compatible way as a shared object.

3.4. The OMX core library wrapper

The final shared object that contains the OMX core methods must also include the OMX core wrapper methods. The OMX core wrapper code is provided:

```
.../codecs_v2/omx/omx_core_plugins/
```

The provider of the final OMX core library must ensure that makefiles that create the final OMX library include the OMX core library wrapper as well. The purpose of OMX core library wrapper is to provide the PV OpenCORE framework with common APIs to dynamically load and communicate with OMX core plugins.

4. Dynamic loading and registration of OMX cores

An additional step is needed to register OMX core plugin libraries and enable the proper dynamic loading of these OMX cores into the PV OpenCORE framework.

The name of the OMX core library, which was built in the manner described above, needs to be recorded in the configuration file - together with the unique OsclUuid interface ID that corresponds to the PV OpenCORE framework API used for communicating with OMX core interfaces.

It is possible to place the information in an existing configuration file (e.g. pvplayer.cfg) or a new configuration file can be created that contains the required information (OsclUuid & library name). If a new configuration file is created, then the configuration file MUST have an extension ".cfg" and needs to be placed in "/system/etc" Android device directory. In either case, a new line is added to the *.cfg file containing:

```
(OMX Core API OsclUuid), "shared library name.so"
```

For example, to properly register and use the OMX core shared library named "libomx_core_vendorXYZ.so", the following line must be placed into the *.cfg file located in /system/etc directory.

```
(0xa054369c,0x22c5,0x412e,0x19,0x17,0x87,0x4c,0x1a,0x19,0xd4,0x5f),
"libomx_core_vendorXYZ.so"
```

NOTE: The OsclUuid provided is the unique API ID that identifies the OMX core interface and cannot be modified:

```
(0xa054369c,0x22c5,0x412e,0x19,0x17,0x87,0x4c,0x1a,0x19,0xd4,0x5f)
```

5. Details on data formats and OMX input buffers

5.1. General Notes about Data Formatting

This section provides additional information about the organization of codec data that is provided inside OMX buffers. Generally the OpenMAX IL spec [1] should provide sufficient details on the data formats, but there are some ambiguities and gaps in the specification that warrant the additional clarification.

5.1.1. Frame (or NAL) start codes

The term start codes refers to sequences in the bitstream that mark different boundaries such as frames or NALs in H.264. In general, start codes are NOT provided in most file formats and network protocols, and therefore the PV OpenCORE framework does NOT generally include start codes in the input buffers to OMX component. There are also no headers (e.g., headers that would provide the frame length, etc) artificially inserted into the bitstream within the input buffers.

5.1.2. OMX buffer fields

The “nFilledLen” OMX buffer field, “nTimestamp” OMX buffer field and OMX_BUFFERFLAG_ENDOFFRAME OMX buffer flag are used to communicate the size of data provided in the OMX buffer as well as frame (or NAL) boundaries. The OMX components should rely on this information to reconstruct the full frames/NALs if necessary. The nOffset is generally set to 0 by the PV OpenCORE IL client, but the OMX component should check the nOffset value in case it is non-zero.

5.1.3. Multiple frames in a single OMX input buffer

Exchanging a large number of relatively small buffers may negatively affect performance. Certain codec formats inherently pack data into packets that contain multiple frames (e.g., AMR IETF). In cases where input data frames are relatively small in size (e.g., AMR and most speech formats) – PV framework may pack multiple FULL frames into a single OMX input buffer. This is the case for most speech codecs. See sections below for details on which exact formats use packing of multiple frames of data into a single buffer. In such cases, OMX input buffers are still marked with OMX_BUFFERFLAG_ENDOFFRAME flag. This is illustrated in Figure 1 below.

Note that if packing of multiple frames into one OMX input buffer is used for a particular format – OMX input buffer contains one or more FULL frames of data of that format.

The “nTimestamp” OMX buffer field refers to the first frame of data in the buffer.

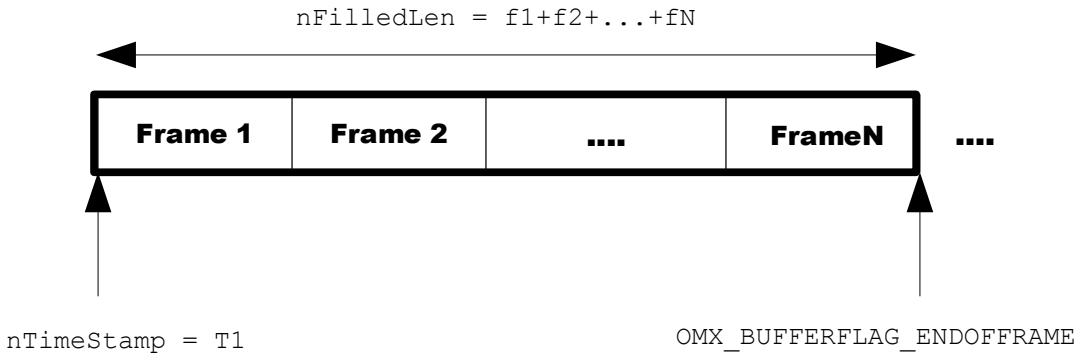

Figure 1: Packing multiple frames into one OMX buffer

5.1.4. Partial frames/NALs

In case of video formats, the PV OpenCORE framework may place either a full or a partial frame (or NAL) into an OMX input buffer. If a full frame or NAL is placed into the OMX input buffer, then the OMX_BUFFERFLAG_ENDOFFRAME flag is applied to that buffer. In case of partial frames/NALs, a frame/NAL may be split into multiple OMX input buffers and sent to the OMX component in pieces. Partial frames generally vary in size. In this case, only the buffer that contains the last piece of the frame/NAL is marked with the OMX_BUFFERFLAG_ENDOFFRAME flag. This is illustrated in Error: Reference source not found and Figure 3 below.

In the case of video formats that allow partial frames, the PV OpenCORE framework NEVER places data that belongs to more than one frame/NAL in the same OMX input buffer which contains a partial frame/NAL. In streaming cases, it is also possible that data packets contain multiple FULL frames.

In other words, in the case of video formats that allow partial frames, each OMX input buffer can contain data that belongs to either:

- multiple FULL frames
- one full frame/NAL
- portion of a single frame/NAL

OMX_BUFFERFLAG_ENDOFFRAME flag (as well as nTimestamp field) enable the component to assemble partial frames.

In the case of formats that allow partial frames – each OMX input buffer that carries the data that belongs to the same frame shall have the same “nTimestamp” field. For example, if a frame is split into 3 OMX input buffers, all 3 input buffers will have the same timestamp field – and the last buffer will also have the OMX_BUFFERFLAG_ENDOFFRAME flag set.

NOTE: If necessary (i.e., if the OMX component is not capable of assembling partial frames), the PV OpenCORE framework can perform the assembly and provide a full frame to the OMX component. In order to enable this feature, it is necessary to use the “capability” flags. In other words, the OMX component must inform the PV OpenCORE framework about its capabilities. This procedure is described in Section 3.2. of the OpenMAX Call Sequences[2] document.

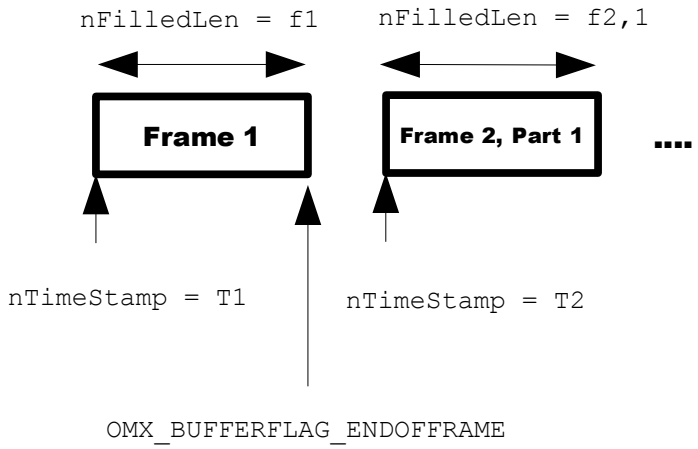

Figure 2: One full frame stored into one OMX buffer

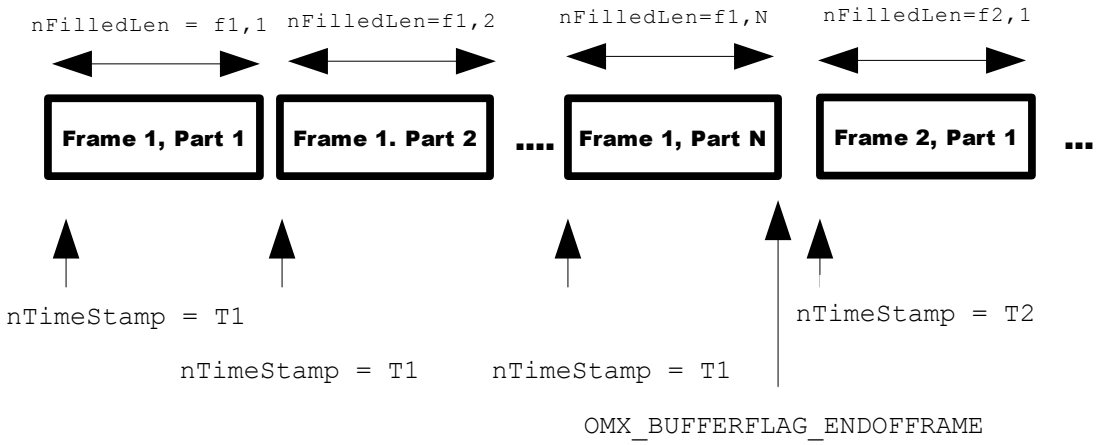

Figure 3: One frame split into N partial frames stored into N OMX buffers

5.1.5. Invalid data packing into OMX buffers

As mentioned above,

a) If PV framework uses packing of multiple frames per one OMX input buffer for a specific codec format, then PV framework will also ensure that only FULL frames will appear in such a buffer, and that partial frames WILL NOT appear in any OMX buffers for that format. This situation is illustrated in Figure 4.

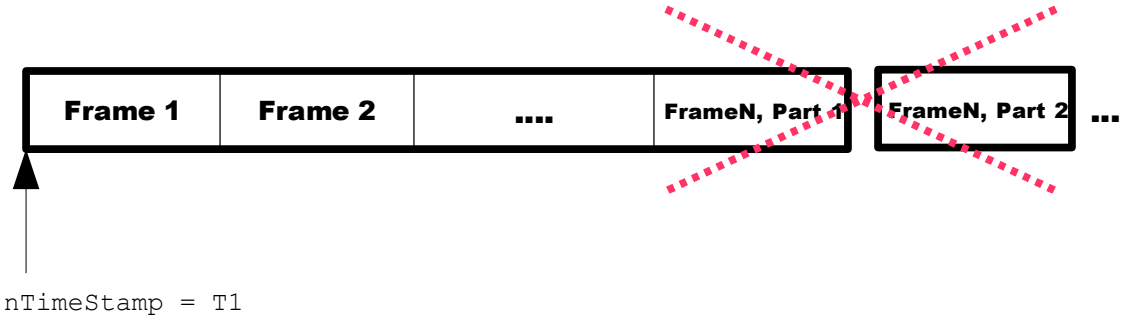

Figure 4: Invalid buffer content - If multiple frames are packed into a single OMX buffer, partial frames WILL NOT be stored in that buffer

b) If PV framework provides a frame/NAL split into partial frames/NALs for a specific codec format, then PV framework will also ensure that any data that belongs to multiple frames/NALs WILL NOT appear in the same OMX buffer. This is illustrated in Figure 5.

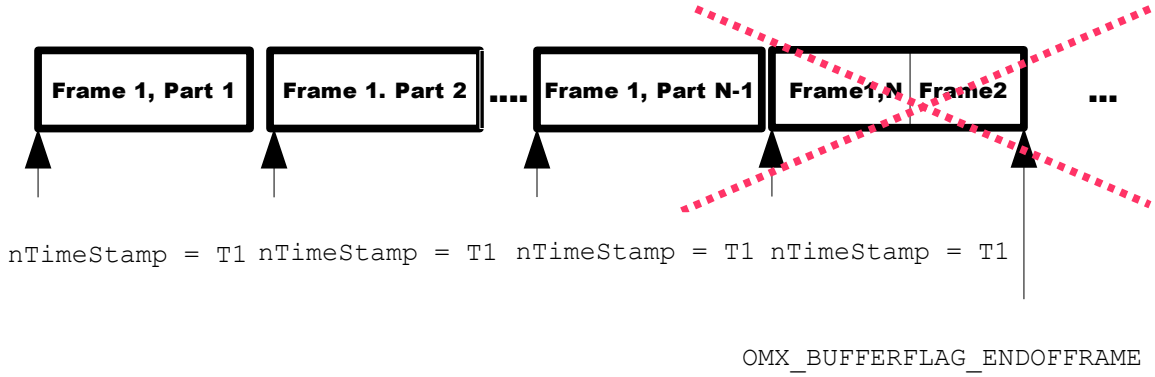

Figure 5: Invalid buffer content - If partial frames are used, data that belongs to more than one frame WILL NOT be stored in the same OMX buffer

5.1.6. Codec configuration data

If applicable, codec configuration data is sent in the first OMX input buffer. Codec configuration data buffer is marked with OMX_BUFFERFLAG_ENDOFFRAME and OMX_BUFFERFLAG_CODECCONFIG flags.

In case of H264 format, SPS and PPS NAL units are sent in separate OMX input buffers (both buffers are marked using OMX_BUFFERFLAG_ENDOFFRAME and OMX_BUFFERFLAG_CODECCONFIG flags).

5.2. AAC decoder formats

AAC decoder data can be stored in multiple formats. The data in the input OMX buffers is organized as follows:

5.2.1. ADIF AAC format

Codec Configuration Header: is sent in the first OMX input buffer. The first OMX input buffer contains only the codec configuration data and is marked with OMX_BUFFERFLAG_ENDOFFRAME and OMX_BUFFERFLAG_CODECCONFIG flag.

Data: ADIF AAC format does not provide frame boundaries. The ADIF data is placed into OMX input buffers sequentially and sent to the OMX component. The size of buffers may vary ("nFilledLen" buffer field informs the component about the quantity of data in the buffer).

Note: As a consequence of the lack of frame boundaries – except for the first OMX buffer (that contains the Codec Configuration Header) that provides the initial timestamp – all other OMX input buffer timestamps are invalid. However, the OMX component is expected to provide valid output buffer timestamps based on the initial input timestamp and the quantity of output PCM data that is produced.

5.2.2. ADTS AAC format

Codec Configuration Header is sent in the first OMX input buffer. The first OMX input buffer contains only the codec configuration data and is marked with OMX_BUFFERFLAG_ENDOFFRAME and OMX_BUFFERFLAG_CODECCONFIG flag.

Data: Each OMX input buffer contains one (or more than one) full ADTS frames. All OMX input buffers are marked with OMX_BUFFERFLAG_ENDOFFRAME. No partial frames are ever placed into OMX input buffer for this format. Since there can be multiple frames in a single OMX input buffer, the OMX component needs to keep consuming the data from the buffer until data is exhausted.

5.2.3. LATM AAC format

Codec Configuration Header is sent in the first OMX input buffer. The first OMX input buffer contains only the codec configuration data and is marked with OMX_BUFFERFLAG_ENDOFFRAME and OMX_BUFFERFLAG_CODECCONFIG flag.

Data: Each OMX input buffer contains one (or more than one) FULL LATM frames. All OMX input buffers are marked with OMX_BUFFERFLAG_ENDOFFRAME. No partial frames are ever placed into OMX input buffer for this format. Since there can be multiple frames in a single OMX input buffer, the OMX component needs to keep consuming the data from the buffer until data is exhausted.

5.2.4. MP4 audio AAC format

Codec Configuration Header is sent in the first OMX input buffer. The first OMX input buffer contains only the codec configuration data and is marked with OMX_BUFFERFLAG_ENDOFFRAME and OMX_BUFFERFLAG_CODECCONFIG flag.

Data: Each OMX input buffer contains one (or more than one) FULL MP4 Audio frames. All OMX input buffers are marked with OMX_BUFFERFLAG_ENDOFFRAME. No partial frames are ever placed into OMX input buffer for this format. Since there can be multiple frames in a single OMX input buffer, the OMX component needs to keep consuming the data from the buffer until data is exhausted.

5.3. AMR decoder formats

The following applies to both Wide-band and Narrow-band versions of AMR.

Codec Configuration Header is NOT provided separately for this format. All the necessary configuration is provided through port parameters (e.g. IF2 vs. IETF)

Data: Each OMX input buffer contains one (or more than one) FULL AMR frames. All OMX input buffers are marked with OMX_BUFFERFLAG_ENDOFFRAME. No partial frames are ever placed into OMX input buffer for this format. AMR data formatting is based on the RFC 4867 document.

Note:
In case of RTP, the RTP packet is placed into the OMX input buffer (without the header). Based on RFC-4867 (RFC-3267), RTP AMR payload data is preceded by the table of contents which lists the frame types and indicates (implicitly) the number of frames present in the buffer.

File storage format and IF2: AMR data for IF2 or File Storage Format contains the frame type field followed by frame data. The frame type field determines the size of the frame that follows. Since there can be multiple frames in a single OMX input buffer, the OMX component needs to keep consuming the data from the buffer until data is exhausted.

5.4. MP3 decoder format

Codec Configuration Header is NOT provided separately for this format. Each frame of data carries its own header.

Data: Each OMX input buffer contains one (or more than one) FULL MP3 frames. All OMX input buffers are marked with OMX_BUFFERFLAG_ENDOFFRAME. No partial frames are ever placed into OMX input buffer for this format. Since there can be multiple frames in a single OMX input buffer, the OMX component needs to keep consuming the data from the buffer until data is exhausted.

5.5. WMA decoder format

Codec Configuration Header is sent in the first OMX input buffer. The first OMX input buffer contains only the codec configuration data and is marked with OMX_BUFFERFLAG_ENDOFFRAME and OMX_BUFFERFLAG_CODECCONFIG flag.

Data: Each OMX input buffer contains one (or more than one) FULL WMA frames. All OMX input buffers are marked with OMX_BUFFERFLAG_ENDOFFRAME. No partial frames are ever placed into OMX input buffer for this format. Since there can be multiple frames in a single OMX input buffer, the OMX component needs to keep consuming the data from the buffer until data is exhausted.

5.6. MPEG4 video decoder format

Codec Configuration Header (VOL header) s sent in the first OMX input buffer. The first OMX input buffer contains only the codec configuration data and is marked with OMX_BUFFERFLAG_ENDOFFRAME and OMX_BUFFERFLAG_CODECCONFIG flag.

Data: Each OMX input buffer contains either one or more full MPEG4 frames or a portion of a single frame. In case the buffer contains one or more full frames, OMX_BUFFERFLAG_ENDOFFRAME flag is applied to the OMX input buffer. In case the buffer contains a partial frame, OMX_BUFFERFLAG_ENDOFFRAME flag is applied to the input buffer only if it contains the last piece of that frame. If an OMX buffer contains a partial frame, then data belonging to more than one frame is NEVER placed in the same OMX input buffer .

Note: If necessary – i.e. if the OMX component is not capable of assembling partial frames, the PV OpenCORE framework can perform the assembly and provide a full frame to the OMX component. In order to enable this feature, it is necessary to use the “capability” flags. In other words, the OMX component must inform the PV OpenCORE framework about its capabilities. This procedure is described in Section 3.2. of OHA document “OpenMAX call sequences”. Frame assembly in the PV OpenCORE framework may incur a performance penalty.

5.7. H263 video decoder format

Codec Configuration Header is NOT provided separately for this format. Each frame of data carries its own header.

Data: Each OMX input buffer contains either one or more full H263 frames or a portion of a single frame. In case the buffer contains one or more full frames, OMX_BUFFERFLAG_ENDOFFRAME flag is applied to the OMX input buffer. In case the buffer contains a partial frame, OMX_BUFFERFLAG_ENDOFFRAME flag is applied to the input buffer only if the buffer contains the last piece of that frame. If an OMX buffer contains a partial frame, then data belonging to more than one frame is NEVER placed in the same OMX input buffer.

Note1: H263 is the only format where the very first OMX input buffer might not be marked with OMX_BUFFERFLAG_ENDOFFRAME.

Typically, for all other PV supported formats - the first OMX input buffer either contains the codec configuration header or is guaranteed to contain a full frame. In both of these cases – the very first OMX input buffer is always marked with OMX_BUFFERFLAG_ENDOFFRAME. This may not be the case for H263 since the first OMX input buffer may contain the first piece of the first h263 frame.

Note2: If necessary – i.e. if the OMX component is not capable of assembling partial frames, the PV OpenCORE framework can perform the assembly and provide a full frame to the OMX component. In order to enable this feature, it is necessary to use the “capability” flags. In other words, the OMX component must inform the PV OpenCORE framework about its capabilities. This procedure is described in Section 3.2. of OHA document “OpenMAX call sequences”. Frame assembly in the PV OpenCORE framework may incur a performance penalty.

5.8. WMV decoder format

Codec Configuration Header is sent in the first OMX input buffer. The first OMX input buffer contains only the codec configuration data and is marked with OMX_BUFFERFLAG_ENDOFFRAME and OMX_BUFFERFLAG_CODECCONFIG flag.

Data: Each OMX input buffer contains either one or more full WMV frames or a portion of a single frame. In case the buffer contains a full frame, OMX_BUFFERFLAG_ENDOFFRAME flag is applied to the OMX input buffer. In case the buffer contains a partial frame, OMX_BUFFERFLAG_ENDOFFRAME flag is applied to the input buffer only if it contains the last piece of that frame. If an OMX buffer contains a partial frame, then data belonging to more than one frame is NEVER placed in the same OMX input buffer.

Note: If necessary – i.e. if the OMX component is not capable of assembling partial frames, the PV OpenCORE framework can perform the assembly and provide a full frame to the OMX component. In order to enable this feature, it is necessary to use the “capability” flags. In other words, the OMX component must inform the PV OpenCORE framework about its capabilities. This procedure is described in Section 3.2. of OHA document “OpenMAX call sequences”. Frame assembly in the PV OpenCORE framework may incur a performance penalty.

5.9. H264/AVC decoder format

Codec Configuration Header:
SPS and PPS NAL units are sent in the first OMX input buffers (the order in which SPS and PPS are sent is not guaranteed). SPS and PPS NALs are sent in separate buffers and these input buffers are marked with OMX_BUFFERFLAG_ENDOFFRAME and OMX_BUFFERFLAG_CODECCONFIG flag.

Data: Each OMX input buffer contains either one full AVC NAL or a portion of a NAL. In case the buffer contains a full NAL, OMX_BUFFERFLAG_ENDOFFRAME flag is applied to the OMX input buffer. In case the buffer contains a partial NAL, OMX_BUFFERFLAG_ENDOFFRAME flag is applied to the input buffer only if it contains the last piece of that NAL. Data belonging to more than one NAL is NEVER placed in the same OMX input buffer for this format.

Note 1: H264/AVC format is NAL based and applies OMX_BUFFERFLAG_ENDOFFRAME to mark the NAL (and NOT frame) boundaries. It is possible to infer the frame boundary based on the “nTimestamp” buffer fields which contain the same timestamp for all NALs and portions of NALs that belong to the same frame.

Note 2: There is no NAL start codes in the bitstream, and by default, NAL start codes are not inserted by the PV OpenCORE framework. If necessary, it is possible for the PV OpenCORE framework to insert NAL start codes at the beginning of each NAL (with a performance penalty). In order to enable this feature, it is necessary to use the “capability” flags. In other words, the OMX component must inform the PV OpenCORE framework about its capabilities. This procedure is described in Section 3.2. of OHA document “OpenMAX call sequences”.

Note3: If necessary – i.e. if the OMX component is not capable of assembling partial NALs, the PV OpenCORE framework can perform the assembly and provide a full NAL to the OMX

component. In order to enable this feature, it is necessary to use the “capability” flags. In other words, the OMX component must inform the PV OpenCORE framework about its capabilities. This procedure is described in Section 3.2. of OHA document “OpenMAX call sequences”. NAL assembly in the PV OpenCORE framework may incur a performance penalty.

5.10. YUV/RGB data format

In case of OMX video encoder components, raw video data is provided in either YUV or RGB format. The PV OpenCORE framework will provide one FULL frame of YUV or RGB data to OMX components.