SAMPLE

Pipeline Encoder

1 Introduction

The purpose of this sample is to demonstrate how to build and execute a pipeline-based video encoder using AMD Media Framework (AMF). The sample encodes raw video content to generate compressed H.264 Elementary stream.

2 Using the sample

2.1 Location

`$<installDirectory>\samples\amf\pipelineEncoder\`

2.2 Contents

Package Contents

Folder:
`$<installDirectory>\samples\amf\pipelineEncoder\src\`

File name	Description
`EncodePipeline.cpp`	Source file for Encode Pipeline class application
`EncodeMain.cpp`	Source file for Encode command line application
`EncodeConfig.cpp`	Source file for reading and parsing encoder configuration file

Folder:
`$<installDirectory>\samples\amf\pipelineEncoder\inc\`

File name	Description
`EncodePipeline.h`	Header file for Encode Pipeline class
`EncodeConfig.h`	Header file for reading and parsing encoder configuration file

Folder:
`$<installDirectory>\samples\amf\pipelineEncoder\build\windows\`

File name	Description
`PipelineEncoderVs10.sln`	Microsoft Visual Studio 10 solution file
`PipelineEncoderVs10.vcxproj`	Microsoft Visual Studio 10 project file
`PipelineEncoderVs10.vcxproj.filters`	Microsoft Visual Studio 10 project filter file
`PipelineEncoderVs12.sln`	Microsoft Visual Studio 12 project solution file
`PipelineEncoderVs12.vcxproj`	Microsoft Visual Studio 12 project file
`PipelineEncoderVs12.vcxproj.filters`	Microsoft Visual Studio 12 project filter file

Folder:
`$<installDirectory>\samples\amf\common\src\`

File name	**Description**
`CmdLogger.cpp`	Source file for Command Logging
`DeviceDX9.cpp`	Source file for DX9 Device
`DeviceDX11.cpp`	Source file for DX11 Device
`FileHelper.cpp`	Helper file for file related operations
`ParametersStorage.cpp`	Source file for Parameters Storage
`Pipeline.cpp`	Source file for the Pipeline
`PlatformWindows.cpp`	Source file for Platform Windows
`RawStreamReader.cpp`	Source file for Raw Stream Reader
`Thread.cpp`	Source file for Thread creation and handling

Folder:
`$<installDirectory>\samples\amf\common\inc\`

File name	**Description**
`AMFPlatform.h`	Header file for Platform
`ByteArray.h`	Header file for Byte Array Processing
`CmdLogger.h`	Header file for Command Logging
`DeviceDX9.h`	Header file for DX9 Device
`DeviceDX11.h`	Header file for DX11 Device
`EncoderParams.h`	Header file for Encoder Parameters
`FileHelper.h`	Helper file for file related operations
`ParametersStorage.h`	Header file for Parameters Storage
`Pipeline.h`	Header file for the Pipeline
`PipelineElement.h`	Header file for Pipeline Element
`PlatformWindows.h`	Header file for Platform Windows
`RawStreamReader.h`	Header file for Raw Stream Reader
`Thread.h`	Header file for Thread creation and handling

Folder:
`$<installDirectory>\samples\amf\pipelineEncoder\config\`

File name	**Description**
`exampleConfig.cfg`	Sample configuration file specifying generic encoder configurable parameters
`exampleConfigTranscoding.cfg`	Sample configuration file specifying encoder configurable parameters for Transcoding Usage
`exampleConfigUltraLowLatency.cfg`	Sample configuration file specifying encoder configurable parameters for Ultra Low Latency usage
`exampleConfigLowLatency.cfg`	Sample configuration file specifying encoder configurable parameters for Low Latency usage
`exampleConfigWebcam.cfg`	Sample configuration file specifying encoder configurable parameters for Webcam usage

Folder:
`$<installDirectory>\samples\amf\pipelineEncoder\docs\`

File name	**Description**
`MediaSDK_AMF_pipelineEncoder.pdf`	Sample documentation

2.3 Configurable Parameters

The encoder configurable parameters are divided into the following groups:

Common Properties

Properties such as width, height, engine type, dynamic parameter frequency, frame parameter frequency, define the various common encoder parameters and the frequency of applying the dynamic and per-frame properties to the encoder.

The `Usage` property

`Usage` values as defined in the following table must be set before the `Init()` function is called, and will apply until the end of the encoding session.

Depending on `Usage`, the encoder component enforces values of certain parameters making them read only or invisible to the user. ONLY those parameters which are configurable for `Usage` are mentioned in the respective usage specific configuration files. Also by setting `Usage` most of parameters are set implicitly. So the developer need not set all the parameters.

Usage Mode	Intended use-cases	Comments
Transcoding	Transcoding, video editing	Favor compression efficiency and throughput over latency.
Ultra-low latency	Video game streaming	Optimize for extremely low latency use-cases (e.g. cap the number of bits per frame), to enable high-interactivity applications.
Low Latency	Video collaboration, remote desktop	Optimize for low latency scenarios, but allow occasional bitrate overshoots to preserve quality.
Webcam	Video conferencing	Optimize for a low-latency video conferencing scenario, with scalable video coding (SVC) support.

Static Properties

Static properties (e.g., profile, level) must be defined before the `Init()` function is called, and will apply until the end of the encoding session.

Dynamic Properties

All dynamic properties have default values. Several properties can be changed subsequently and these changes will be flushed to encoder only before the next `Submit()` call.

The user has the flexibility to update these parameters at run time before encoding of a frame. The `setFrameParamFreq` parameter defined in the common properties, sets the rate at which these properties will be applied. For example if set to 30, then after every 30th frame, these parameters will be applied to all the frames encoded henceforth and will be used till new values are set.

Frame Per-Submission Properties

Per submission properties are applied on a per frame basis. They can be set optionally to force a certain behavior (e.g., force frame type to IDR) by updating the properties of the `AMFSurface` object that is passed through the `AMFComponent::Submit()` call.

The `setFrameParamFreq` parameter defined in the common properties, sets the rate at which these properties will be applied. For example, if set to 30, then every 30th frame the frame based parameters will be applied. The user has the flexibility to update these parameters at run time before encoding of a frame.

For a detailed description of all the encoder parameters including their supported values., see the AMD Media SDK AMF Reference Manual.

2.4 Compile

1. Ensure that the following tools and SDKs are present:
   - Microsoft Visual Studio 2010 or 2012
   - If Windows Software Development Kit (SDK) is not installed, install it from http://msdn.microsoft.com/en-us/library/windows/desktop/hh852363.aspx.
2. Open one of the following solution files:
   - `$<`*`installDirectory`*`>\samples\amf\pipelineEncoder\build\windows\pipelineEncoderVs12.sln`
   - `$<`*`installDirectory`*`>\samples\amf\pipelineEncoder\build\windows\pipelineEncoderVs10.sln`
3. Build the sample:
   - Open the `pipelineEncoderVs10.sln` solution file with Microsoft Visual Studio 2010 Professional Edition or the `pipelineEncoderVs12.sln` solution file with Microsoft Visual Studio 2012 Professional Edition.
   - To build all the solutions, select `Build > Build Solution`.
   - The executable `pipelineEncoder.exe` is created in the following folders for 32-bit builds and 64-bit builds:
     `$<`*`installDirectory`*`>\samples\amf\pipelineEncoder\bin\x86\`
     `$<`*`installDirectory`*`>\samples\amf\pipelineEncoder\bin\x86_64\`
   - Depending on the build (i.e. 32-bit or 64-bit), the custom build step copies the appropriate `.dlls` file from the `$<`*`installDirectory`*`>\dll\amf\` folder into the relevant `\bin\` directory.

3 How to Run

The sample can be executed on an AMD platform that includes the VCE hardware block.

On the command prompt, change to the directory that contains the executable, and execute the following command:

```
pipelineEncoder.exe -i <Raw Input> -o <Encoded Output> -c <config file>
-a <optional: adapterID. Default=0> -l <optional: logging level.
Default=0. Range:0 or 1>
```

The input file extensions must conform to those mentioned in the following table:

Input Format	File Extension
NV12	.nv12
YUV420P	.yuv or .420p or I420
BGRA	.bgra
RGBA	.rgba
ARGB	.argb
YV12	.yv12

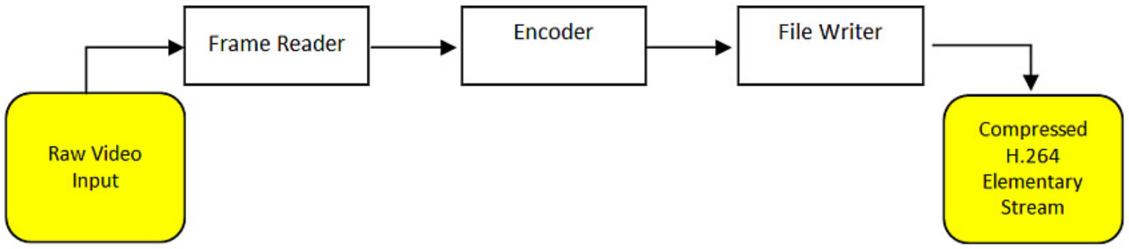
Natively (without internal conversion), the encoder supports only NV12 as input, but if the user passes any other format BGRA, AGRA, RGBA, YV12, YUV420P, it will be converted by the internal converter before submitting to the encoder block.

-a refers to the adapterID. This is an optional parameter. If the user does not specify the adapterID, then the default ID of "0" is used.

-l specifies the logging level. "0" means no logging. "1" generates the log at the API level. Enabling the log generates a log file, `PipelineEncoderErrorLog.txt`, in the sample build folder.

4 Implementation Details

The sample implements the following encode pipeline:



The Data in the Encode Pipeline flows through the following processing elements:

- Reader: Reads one frame worth of data from the input file depending on the color format and feeds the same to the H.264 Encoder.
- Encoder: HW Accelerated (VCE) H.264 Video Encoder. Encodes the input content to generate compressed H.264 Elementary stream.
- File Writer: Component which writes the encoded stream to the output file specified by the user.

5 Supported formats

The following file formats are supported:

- Input file formats: NV12, YUV420P, BGRA, ARGB, RGBA, YV12 frames
- Video encoders supported: H.264
- Output file format: H.264 Compressed Elementary Stream

6 Known Limitations

The sample is currently supported on the following platforms:

- Windows 7 (DirectX 9)
- Windows 8.1 (DirectX 9 and DirectX 11.1)

Contact

Advanced Micro Devices, Inc.
One AMD Place
P.O. Box 3453
Sunnyvale, CA, 94088-3453
Phone: +1.408.749.4000

For AMD Accelerated Parallel Processing:

URL:	**developer.amd.com/appsdk**
Developing:	**developer.amd.com/**
Forum:	**developer.amd.com/openclforum**