Joy of innovation
nuvoTon

NVTMediaSDK User Guide

www.nuvoton.com

March 2020

Outline

- **Introduction**
  - Supported Media, Codec
  - Function
  - Feature
  - Limitation
- **Usage**

Introduction

Introduce NVTMedia SDK

Introduction

- **NVTMediaSDK helps to simplify program of media player and recorder on Nuvoton's N9H26 MPU.**
- **SDK major component: Engine, Media, Codec**
  - For Playback
    - **Engine** calls **Media** to read media and de-mux data into media context by Media interface.
    - **Engine** calls **Codec** to decode context by Codec IF.
    - **Engine** flush decoded context by flush callback.
  - For Recorder
    - **Engine** calls fill callback to retrieve audio/video data into context.
    - **Engine** calls **Codec** to encode context by Codec IF.
    - **Engine** calls **Media** to write context into media by Media IF.
  - Engine controls data flow, status and playback/recording timing.

Supported Media and Codec

Media Type		Audio Codec	Video Codec
File Media	AVI	AAC-LC, G.711 alaw / ulaw	H.264(baseline)
	MP4	AAC-LC, G.711 alaw / ulaw	H.264(baseline)

Function

- **Media Playback**
  - Open / Close
  - Play
  - Pause
  - Fast-forward
  - Seek
- **Media Record**
  - Open / Close
  - Record

Feature

- Support concurrent playback / recording
- Support hardware codec as possible

Limitation

- Audio playback will be mute if audio decoded performance is not enough.
- Video playback will drop frames if video decoded performance is not enough.
- Recording storage management and text overlay are left to application handling.

Usage

How to use NVTMeida API

Sample for Player

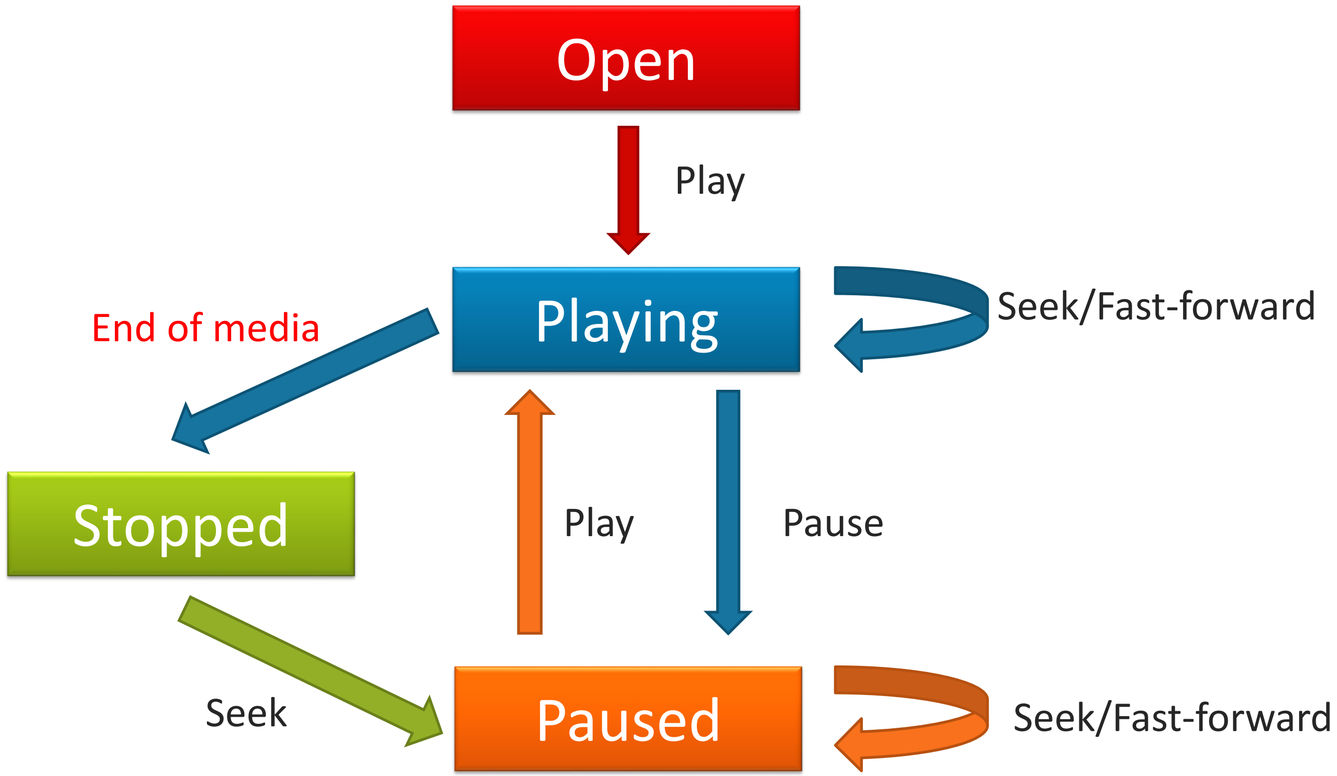
```
E_NM_ERRNO eNMRet = NMPlay_Open(szTestAVIFile, &sPlayIF, &sPlayCtx, &sPlayInfo, &pvOpenRes);
//Setup flush callback
sPlayIF.pfnVideoFlush = Render_VideoFlush;
sPlayIF.pfnAudioFlush = Render_AudioFlush;
//Setup flush audio and video context
sPlayCtx.sFlushVideoCtx.eVideoType = eNM_CTX_VIDEO_YUV422;
sPlayCtx.sFlushVideoCtx.u32Width = LCD_PANEL_WIDTH;
sPlayCtx.sFlushVideoCtx.u32Height = LCD_PANEL_HEIGHT;
sPlayCtx.sFlushAudioCtx.eAudioType = eNM_CTX_AUDIO_PCM_L16;
sPlayCtx.sFlushAudioCtx.u32SampleRate = sPlayCtx.sMediaAudioCtx.u32SampleRate;
sPlayCtx.sFlushAudioCtx.u32Channel = sPlayCtx.sMediaAudioCtx.u32Channel;
```

Sample for Player

```
sPlayCtx.sFlushAudioCtx.u32SamplePerBlock = sPlayCtx.sMediaAudioCtx.u32SamplePerBlock;
sPlayCtx.sFlushAudioCtx.pvParamSet = sPlayCtx.sMediaAudioCtx.pvParamSet;
HPLAY hPlay = (HPLAY)eNM_INVALID_HANDLE;
NMPlay_Play(&hPlay, &sPlayIF, &sPlayCtx, true);
do{
        usleep(1000);
}while(NMPlay_Status(hPlay) != eNM_PLAY_STATUS_EOM)
NMPlay_Close(hPlay, &pvOpenRes);
```

Flowchart for Player

Player Status Transition



Sample for Recorder

```
//Setup fill and Media context
sRecCtx.sFillVideoCtx.eVideoType = eNM_CTX_VIDEO_YUV420P_MB;
sRecCtx.sFillVideoCtx.u32Width = 640;
sRecCtx.sFillVideoCtx.u32Height = 480;
sRecCtx.sFillVideoCtx.u32FrameRate = 30;
sRecCtx.sMediaVideoCtx.eVideoType = eNM_CTX_VIDEO_H264;
sRecCtx.sMediaVideoCtx.u32Width = 640;
sRecCtx.sMediaVideoCtx.u32Height = 480;
sRecCtx.sMediaVideoCtx.u32FrameRate = 30;
sRecCtx.sFillAudioCtx.eAudioType = eNM_CTX_AUDIO_PCM_L16;
sRecCtx.sFillAudioCtx.u32SampleRate = 16000;
sRecCtx.sFillAudioCtx.u32Channel = 1;
```

Sample for Recorder

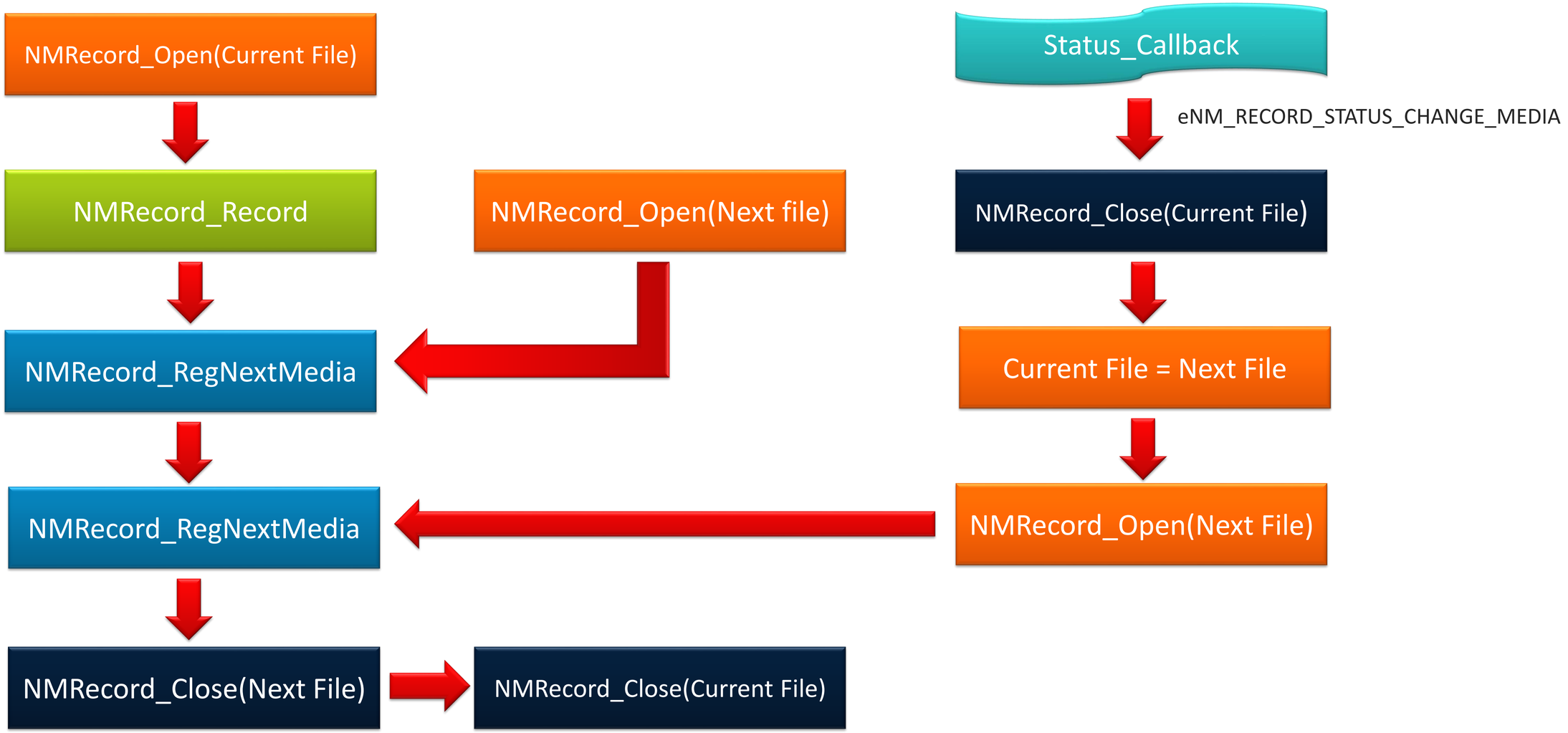
```
        sRecCtx.sMediaAudioCtx.eAudioType = eNM_CTX_AUDIO_AAC;

        sRecCtx.sMediaAudioCtx.u32SampleRate = 16000;

        sRecCtx.sMediaAudioCtx.u32Channel = 1;

        sRecCtx.sMediaAudioCtx.u32BitRate = 64000;

        //Setup fill callback

        sRecIf.pfnVideoFill = VideoIn_FillCB;

        sRecIf.pfnAudioFill = AudioIn_FillCB;

        NMRecord_Open(szFileName, eNM_MEDIA_MP4,  eNM_UMLIMIT_TIME, &sRecCtx, &sRecIf,
&pvOpenRes);

        HRECORD hRecord;

        NMRecord_Record( &hRecord, eNM_UMLIMIT_TIME, &sRecIf, &sRecCtx, Record_StatusCB,
NULL);

        NMRecord_Close(hRecord, &pvOpenRes);
```

Flowchart for Recorder

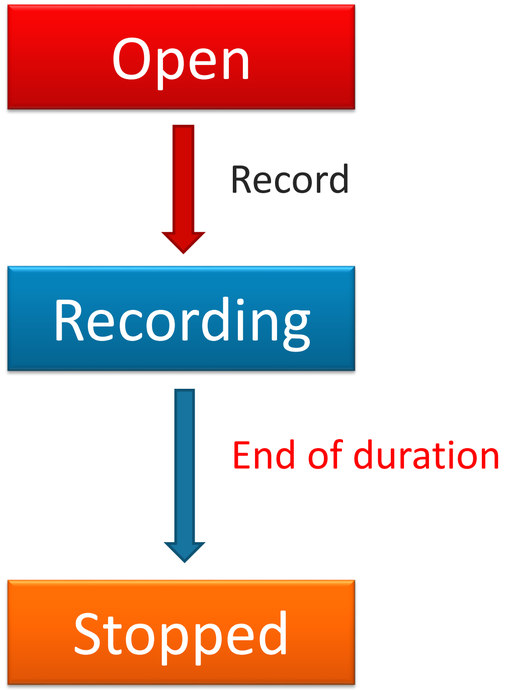
- **Each recording file duration == eNM_UNLIMIT_TIME**

Flowchart for Recorder

- **Each recording file duration != eNM_UNLIMIT_TIME**



Recorder Status Transition



API

- **Playback**
  - NMPlay_Open(*szPath, *psPlayIF, *psPlayCtx, *psPlayInfo, **ppvNMOpenRes);
  - NMPlay_Play(*phPlay, *psPlayIF, *psPlayCtx, bWait);
  - NMPlay_Pause(hPlay, bWait);
  - NMPlay_Fastforward(hPlay, eSpeed, bWait);
  - NMPlay_Seek(hPlay, u32MilliSec, u32TotalVideoChunks, u32TotalAudioChunks, bWait);
  - NMPlay_Status(hPlay);
  - NMPlay_Close(hPlay, **ppvNMOpenRes);

API

- **Record**
  - NMRecord_Open(*szPath, eMediaType, u32Duration, *psRecordCtx, *psRecordIF, **ppvNMOpenRes);
  - NMRecord_Record(*phRecord, u32Duration, *psRecordIF, *psRecordCtx, pfnStatusCB, *pvStatusCBPriv);
  - NMRecord_RegNextMedia(hRecord, *psMediaIF, *pvMediaRes, *pvStatusCBPriv);
  - NMRecord_Close(hRecord, **ppvNMOpenRes);

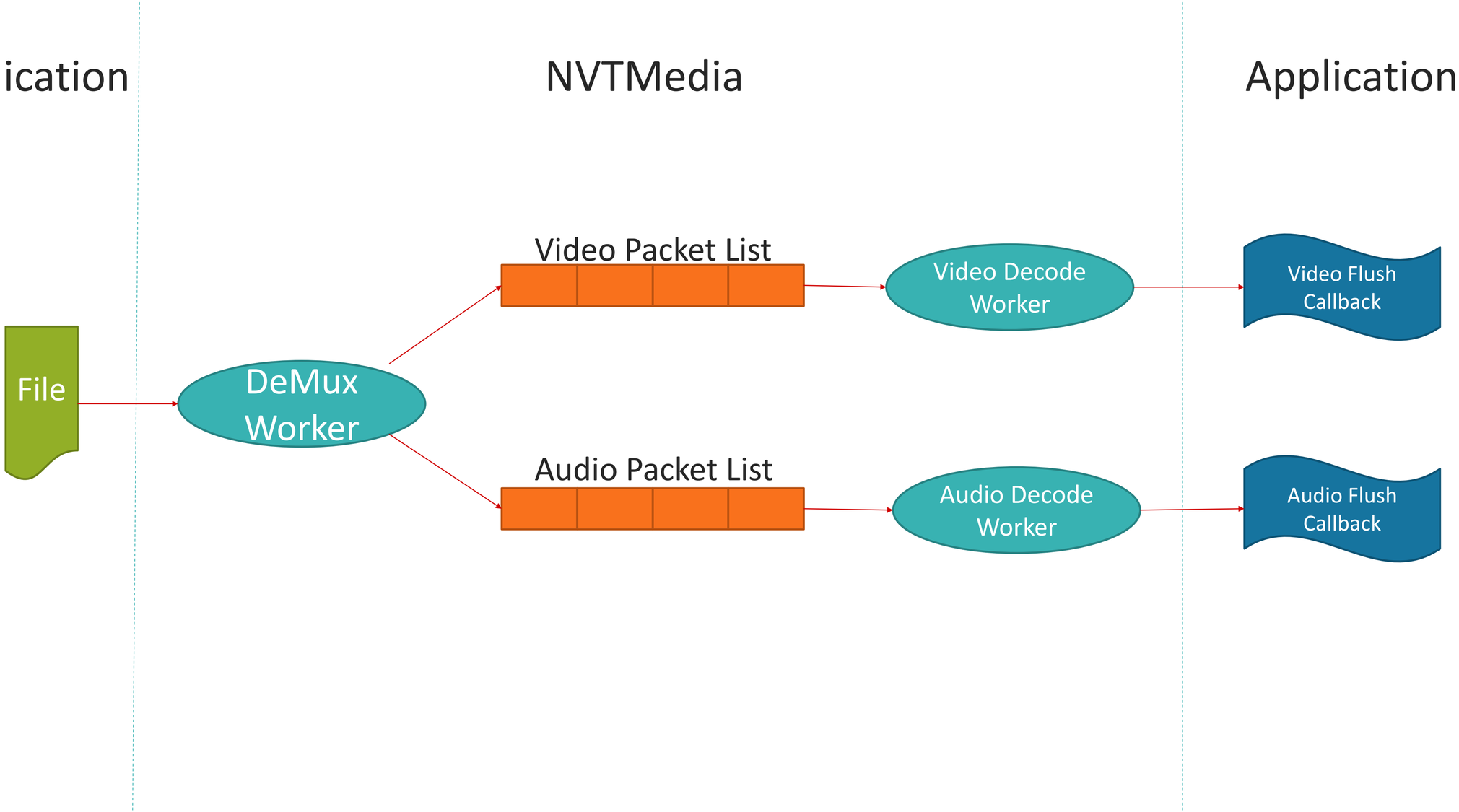
Built-in Media Interface

Media Type		Media Interface	
		Read	Write
File Media	AVI	g_sAVIReader_IF	g_sAVIWriter_IF
	MP4	g_sMP4Reader_IF	g_sMP4Writer_IF

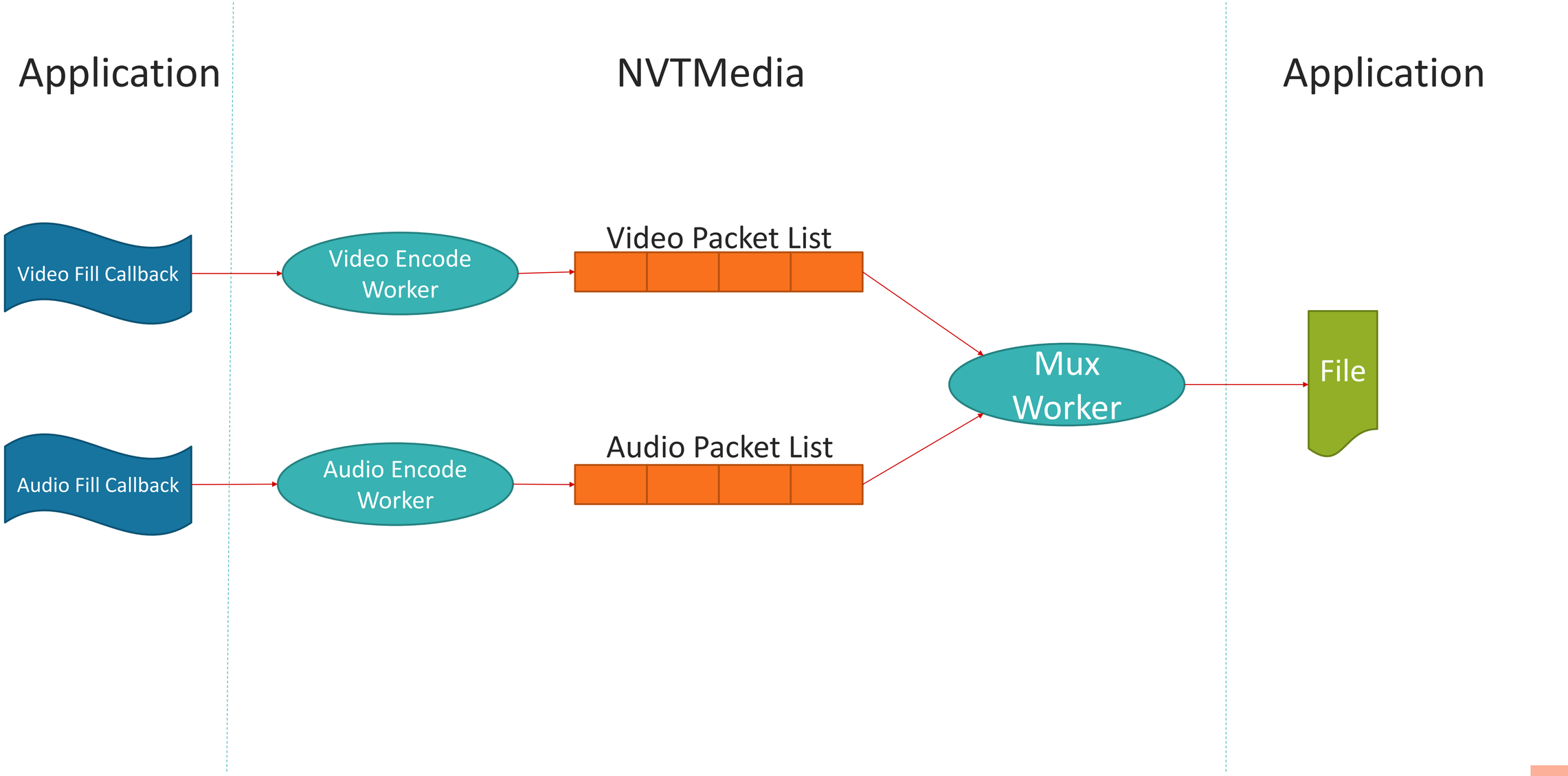
Built-in Codec Interface

	Codec Type	Codec Interface	
		Decoder	Encoder
Audio	**AAC**	g_sAACDec_IF	g_sAACEnc_IF
	G.711	g_sG711Dec_IF	g_sG711Enc_IF
Video	**H.264**	g_sH264Dec_IF	g_sH264Enc_IF

Player working flow



Recorder working flow



Troubleshooting

- **[NMRecord] If the recorded file is not smooth**
  - Checking the timestamp of each chunk on fill callback is correct or not
  - Checking the performance of storage is enough or not