SONY

■ Summary

This document describes how to implement video/audio decoding using swift on mac when executing the video playback function of RemoteSDK.
The following slides describe the specific implementation and SampleCode.

■ Video data decoding Part 1

SONY

When receiving video data after executing the video playback function, nal analysis is performed on the received video data, dividing it into “vps/sps/pps/other nal units” and extracting nal units.
Create a CMBlockBuffer from “pps/other nal units” with createBlockBuffer().

<Example>

```
class VideoDecoderH265 {
・・・(Omitted.)
   typealias AnalyseNALResult = (vps: UnsafeBufferPointer<UInt8>?, sps: UnsafeBufferPointer<UInt8>?,
                                 pps: UnsafeBufferPointer<UInt8>?, otherNalus: [UnsafeBufferPointer<UInt8>])

   public func decode(recieveData: Data, pts: Int64, frameRate: Int32) -> Void {
            var naluInfo : AnalyseNALResult
            ・・・
            nal analysis performed.
            ・・・
            // naluInfo.pps and naluInfo.otherNalus are the results obtained from the nal analysis
            guard let blockBuffer : CMBlockBuffer = self.createBlockBuffer(pps: naluInfo.pps!, otherNalus: naluInfo.otherNalus) else { return }

・・・(Omitted.)
```

■ Description of createBlockBuffer()

SONY

Allocate an area for “pps/other nal units” with CMBlockBufferCreateWithMemoryBlock().
Writes “pps/other nal units” to the area allocated by CMBlockBufferReplaceDataBytes().
Returns CMBlockBuffer.

```
<Example>
  private func createBlockBuffer (pps: UnsafeBufferPointer<UInt8>?, otherNalus: [UnsafeBufferPointer<UInt8>]) -> CMBlockBuffer? {
      var nalus: [UnsafeBufferPointer<UInt8>] = []
      nalus.append(pps!)
      for nalu in otherNalus {
          nalus.append(nalu)
      }
      let blockLength = nalus.reduce(0) { partialResult, nalu in partialResult + nalu.count + 4 }
      var blockBuffer: CMBlockBuffer? = nil
      var res = CMBlockBufferCreateWithMemoryBlock(allocator: kCFAllocatorDefault, memoryBlock: nil, blockLength: blockLength,
          blockAllocator: kCFAllocatorDefault, customBlockSource: nil, offsetToData: 0, dataLength: blockLength, flags: 0, blockBufferOut: &blockBuffer)
      guard let blockBuffer = blockBuffer else {
          return nil
      }
      var offset = 0
      for nalu in nalus {
          let naluSize = nalu.count
          var header: UInt32 = CFSwapInt32HostToBig(UInt32(naluSize))
          res = CMBlockBufferReplaceDataBytes(with: &header, blockBuffer: blockBuffer, offsetIntoDestination: offset, dataLength: 4)
          offset += 4
          res = CMBlockBufferReplaceDataBytes(with: nalu.baseAddress!, blockBuffer: blockBuffer, offsetIntoDestination: offset, dataLength: naluSize)
          offset += naluSize
      }
      return blockBuffer
  }
```

■ Video data decoding Part 2

SONY

If CMVideoFormatDescription has not been created or any of the vps/psps/pps parameters have changed, create a CMVideoFormatDescription with createFormatDescription(), createDecompSession() to create a VTDecompressionSession.

```
<Example>
 private var formatDescription: CMVideoFormatDescription? = nil
 private var curVps: [UInt8] = []
 private var curSps: [UInt8] = []
 private var curPps: [UInt8] = []
 private var session: VTDecompressionSession? = nil

func decode(recieveData: Data, pts: Int64, frameRate: Int32) {
・・・(Omitted.)
        if let vps = naluInfo.vps, let sps = naluInfo.sps, let pps = naluInfo.pps {
            if self.formatDescription == nil
                || memcmp(vps.baseAddress!, &self.curVps, vps.count) != 0
                || memcmp(sps.baseAddress!, &self.curSps, sps.count) != 0
                || memcmp(pps.baseAddress!, &self.curPps, pps.count) != 0 {
                self.formatDescription = self.createFormatDescription(vps: vps, sps: sps, pps: pps)
                self.curVps = Array(vps)
                self.curSps = Array(sps)
                self.curPps = Array(pps)
                if let formatDescription = self.formatDescription {
                    self.session = self.createDecompSession(formatDescription: formatDescription)
                }
            }
        }
・・・(Omitted.)
```

■ Description of createFormatDescription()

SONY

Using vps/sps/pps, CMVideoformatDescriptionCreateFromHEVCParameterSets() to create a CMFormatDescription.

<Example>

```
private func createFormatDescription(vps: UnsafeBufferPointer<UInt8>, sps: UnsafeBufferPointer<UInt8>, pps: UnsafeBufferPointer<UInt8>) -> CMFormatDescription? {
    var formatDescription: CMVideoFormatDescription? = nil
    var parameterSetPointers: [UnsafePointer<UInt8>] = [vps.baseAddress!, sps.baseAddress!, pps.baseAddress!]
    let parameterSetSizes: [Int] = [vps.count, sps.count, pps.count]
    let res = CMVideoFormatDescriptionCreateFromHEVCParameterSets(
        allocator: kCFAllocatorDefault,
        parameterSetCount: parameterSetPointers.count,
        parameterSetPointers: &parameterSetPointers,
        parameterSetSizes: parameterSetSizes,
        nalUnitHeaderLength: 4,
        extensions: nil,
        formatDescriptionOut: &formatDescription)
    guard res == noErr, let formatDescription = formatDescription else {
        return nil
    }
    return formatDescription
}
```

■ Description of createDecompSession() Part 1

SONY

Define a callback function to be called when a frame decoded by VTDecompressionOutputCallbackRecord() becomes available.
Set the decoder in VTDecompressionSession() and pass the callback record.

```
<Example>
 private func createDecompSession(formatDescription: CMVideoFormatDescription) -> VTDecompressionSession? {
       var ouputCallback = VTDecompressionOutputCallbackRecord(
          decompressionOutputCallback: { (decompressionOutputRefCon: UnsafeMutableRawPointer?, _: UnsafeMutableRawPointer?,
                                          status: OSStatus, _: VTDecodeInfoFlags, imageBuffer: CVImageBuffer?, presentationTimeStamp: CMTime, presentationDuration: CMTime) in
             guard let decompressionOutputRefCon = decompressionOutputRefCon else { return }
             let refcon = Unmanaged<VideoDecoderH265>.fromOpaque(decompressionOutputRefCon).takeUnretainedValue()
             refcon.decompCallback(
                status: status, imageBuffer: imageBuffer, pts: presentationTimeStamp, duration: presentationDuration
             )
          },
          decompressionOutputRefCon: Unmanaged.passUnretained(self).toOpaque()
       )

       var session: VTDecompressionSession? = nil
       let status = VTDecompressionSessionCreate(allocator: kCFAllocatorDefault, formatDescription: formatDescription, decoderSpecification: nil,
          imageBufferAttributes: nil, outputCallback: &ouputCallback, decompressionSessionOut: &session)
       return session
    }
```

■ Description of createDecompSession() Part 2

SONY

The registered callback function is executed when a decoded frame becomes available.
Generate a format descriptor (formatDescriptionOut) from the image buffer with CMVideoFormatDescriptionCreateForImageBuffer() when the callback function is executed.
Generate a sample buffer (CMSampleBuffer) from the image buffer with CMSampleBufferCreateForImageBuffer().
This converts the image buffer into a displayable format.
Display the sample buffer (CMSampleBuffer) on the screen using AVSampleBufferDisplayLayer, etc.

<Example>

```
func decompressionOutputCallback(_: UnsafeMutableRawPointer?, _: UnsafeMutableRawPointer?, status: OSStatus, _: VTDecodeInfoFlags,
                                 imageBuffer: CVImageBuffer?, presentationTimeStamp: CMTime, presentationDuration: CMTime) {
    guard let imageBuffer = imageBuffer else {
        return;
    }
    var formatDescriptionOut: CMVideoFormatDescription?
    var status = CMVideoFormatDescriptionCreateForImageBuffer(allocator: kCFAllocatorDefault,
        imageBuffer: imageBuffer, formatDescriptionOut: &formatDescriptionOut)
    guard status == noErr, let formatDescription = formatDescriptionOut  else {
        return
    }

    var sampleTiming = CMSampleTimingInfo(duration: presentationDuration, presentationTimeStamp: presentationTimeStamp, decodeTimeStamp: .invalid)
    var sampleBuffer: CMSampleBuffer? = nil
    status = CMSampleBufferCreateForImageBuffer(allocator: kCFAllocatorDefault, imageBuffer: imageBuffer, dataReady: true,
        makeDataReadyCallback: nil, refcon: nil, formatDescription: formatDescription, sampleTiming: &sampleTiming,
        sampleBufferOut: &sampleBuffer)
    guard status == noErr, let sampleBuffer = sampleBuffer else {
        return
    }

    // Display SampleBuffer on the screen using AVSampleBufferDisplayLayer, etc.
}
```

■ Video data decoding Part 3

SONY

Create a CMSampleBuffer with CMSampleBufferCreate().
Decode the generated CMSampleBuffer with VTDecompressionSessionDecodeFrame().
In addition, video and audio can be synchronized using PTS.

```
<Example>
   func decode(recieveData: Data, pts: Int64, frameRate: Int32) {
・・・(Omitted.)
        var timingInfo = CMSampleTimingInfo(
              duration: CMTimeMake(value: Int64(TIME_SCALE / Int(frameRate)), timescale: Int32(TIME_SCALE)),
              presentationTimeStamp: CMTimeMake(value: pts, timescale: Int32(TIME_SCALE)),
              decodeTimeStamp: .invalid
        )
        var sampleSize = CMBlockBufferGetDataLength(blockBuffer)
        var sampleBuffer: CMSampleBuffer? = nil
        var status = CMSampleBufferCreate(allocator: kCFAllocatorDefault, dataBuffer: blockBuffer, dataReady: true, makeDataReadyCallback: nil, refcon: nil,
                    formatDescription: formatDescription, sampleCount: 1, sampleTimingEntryCount: 1, sampleTimingArray: &timingInfo,
                    sampleSizeEntryCount: 1, sampleSizeArray: &sampleSize, sampleBufferOut: &sampleBuffer )

        if let session = self.session, let sampleBuffer = sampleBuffer {
            status = VTDecompressionSessionDecodeFrame(session, sampleBuffer: sampleBuffer, flags: [], frameRefcon: nil, infoFlagsOut: nil)
        }
```

■ Audio data decoding Part 1

SONY

Only when audio data is received for the first time after the video playback function is executed, the audio format (AudioStreamBasicDescription) is generate and the audio format descriptor (CMAudioFormatDescription) is create.

```
<Example>
 private var audioDataFirstReceived = false
 private var formatDescription: CMAudioFormatDescription? = nil

func decode(recieveData: Data, pts: Int64, sampleRate: Int32, channel: Int32) {
       if (!self.audioDataFirstReceived) {
          var audioStreamBasicDescription = AudioStreamBasicDescription(
             mSampleRate: Float64(sampleRate), mFormatID: kAudioFormatMPEG4AAC, mFormatFlags: 0,
             mBytesPerPacket: 0, mFramesPerPacket: UInt32(1024), mBytesPerFrame: 0,
             mChannelsPerFrame: UInt32(channel), mBitsPerChannel: 0, mReserved: 0 )

          let status = CMAudioFormatDescriptionCreate( allocator: kCFAllocatorDefault,
             asbd: &audioStreamBasicDescription, layoutSize: 0, layout: nil,
             magicCookieSize: 0, magicCookie: nil, extensions: nil,
             formatDescriptionOut: &self.formatDescription )
          if status == noErr {
             self.audioDataFirstReceived = true
          }
       }
・・・(Omitted.)
```

■ Audio data decoding Part 2

SONY

If the frame count is 1, perform the initialization process and initialize the timing information and data buffer.
Add information about each packet to the list as AudioStreamPacketDescription().
Add new data to the existing data buffer. If the frame count is 2 or more, proceed with the process and initialize the frame count.
In iOS, if more than 2 frames are not decoded at the same time, an error will occur, so the process will accumulate 2 frames before decoding.

```
<Example>
 private var frameCount: Int = 0
 private var frameData: Data = Data()
 private var aspdArray: [AudioStreamPacketDescription] = [AudioStreamPacketDescription]()

func decode(recieveData: Data, pts: Int64, sampleRate: Int32, channel: Int32) {
・・・(Omitted.)
        self.frameCount += 1
        if frameCount == 1 {
            self.pts = pts
            var aspdArray = [AudioStreamPacketDescription]()
            aspdArray.reserveCapacity(2)
            self.aspdArray = aspdArray
            self.frameData = Data()
        }

        let aacAdtsLen: Int = 7
        aspdArray.append(AudioStreamPacketDescription(mStartOffset: Int64(self.frameData.count + aacAdtsLen),
                                mVariableFramesInPacket: 0, mDataByteSize: UInt32(recieveData.count - aacAdtsLen)))
        self.frameData.append(recieveData)
        if self.frameCount < 2 {
            return
        }
        self.frameCount = 0
・・・(Omitted.)
```

■ Audio data decoding Part 3

SONY

Create a CMBlockBuffer corresponding to the frame data with CMBlockBufferCreateWithMemoryBlock().
Copy the data with CMBlockBufferReplaceDataBytes().

```
<Example>
func decode(recieveData: Data, pts: Int64, sampleRate: Int32, channel: Int32) {
・・・(Omitted.)
    var blockBuffer: CMBlockBuffer? = nil
    var status = CMBlockBufferCreateWithMemoryBlock(
        allocator: kCFAllocatorDefault,
        memoryBlock: nil,
        blockLength: frameData.count,
        blockAllocator: nil,
        customBlockSource: nil,
        offsetToData: 0,
        dataLength: frameData.count,
        flags: 0,
        blockBufferOut: &blockBuffer)
    guard let blockBuffer = blockBuffer else {
        return
    }

    status = frameData.withUnsafeBytes( { (vp: UnsafeRawBufferPointer) -> OSStatus in
        return CMBlockBufferReplaceDataBytes(with: vp.baseAddress!, blockBuffer: blockBuffer, offsetIntoDestination: 0, dataLength: frameData.count)
    })
・・・(Omitted.)
```

■ Audio data decoding Part 4

SONY

Create a CMSampleBuffer with CMAudioSampleBufferCreateReadyWithPacketDescriptions() based on frame data and packet information.
Play the created CMSampleBuffer using AVSampleBufferAudioRenderer(), etc.

```
<Example>
func decode(recieveData: Data, pts: Int64, sampleRate: Int32, channel: Int32) {
・・・(Omitted.)
 var sampleBuffer: CMSampleBuffer? = nil
       status = CMAudioSampleBufferCreateReadyWithPacketDescriptions(
           allocator: kCFAllocatorDefault,
           dataBuffer: blockBuffer,
           formatDescription: formatDescription,
           sampleCount: frameCount,
           presentationTimeStamp: CMTimeMake(value: self.pts, timescale: Int32(TIME_SCALE)),
           packetDescriptions: aspdArray,
           sampleBufferOut: &sampleBuffer
       )
       guard status == noErr, let sampleBuffer = sampleBuffer else {
           return
       }
       // Play sampleBuffer using AVSampleBufferAudioRenderer(), etc.
・・・(Omitted.)
```