

INTO THE new
BREW 2007 CONFERENCE

Using IMedia and ICamera

Srinivas Patwari, Staff Engineer
QUALCOMM Incorporated





Overview

- BREW® IMedia
- BREW ICamera
- Q & A



BREW IMedia

- BREW IMedia Overview
- IMedia APIs
- IMedia Scenarios
- Custom IMedia Extension



IMedia Overview

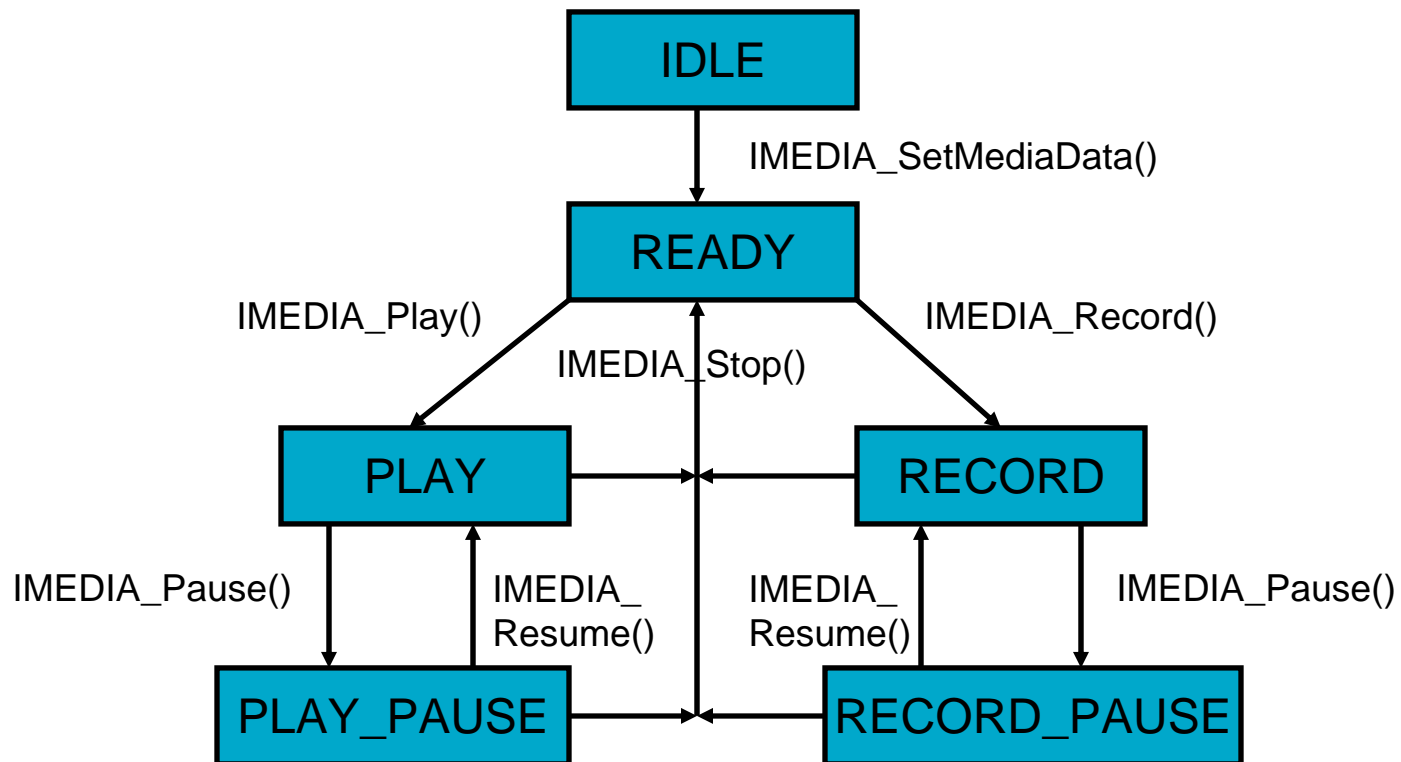
- IMedia is the abstract base class to all audio/video media formats
- IMedia allows media data source/sink to be file, memory or stream
- IMedia-based derived classes handle specific media format
- BREW provides a set of concrete IMedia class implementations
- IMediaUtil interface enables media type detection and creates the media object



IMedia APIs

- **IMedia** is an abstract interface that provides APIs to
 - Render media (audio/video)
 - Control playback like seek, pause, resume
 - Record media
 - Set/Get media control parameters
 - Handle asynchronous events from IMedia object
 - Get media state

IMedia State Machine





IMedia API: Scenarios Media Setup



Media Setup

- Two methods to create IMedia object
 - Method 1: Media type known
 - > Use ISHELL_CreateInstance()
 - > Set media data
 - > Use IMedia object
 - Method 2: Media type unknown
 - > Call IMEDIAUTIL_CreateMedia() with media data
 - > Use IMedia object
- Register with IMedia object to receive media events

Media Setup: Media type known

- Create IMedia object

```
// Create an IMedia object to render MP3 content.  
// Returned IMedia object state: IDLE  
ISHELL_CreateInstance(pme->a.m_pIShell, AEECLSID_MEDIAMP3,  
                      (void **)&pme->m_pIMedia);
```

- Specify media data in **AEEMediaData**
 - **AEEMediaData::clsData** is set to the source type
 - > **MMD_FILE_NAME** specifies file name of the media
 - > **MMD_BUFFER** specifies memory buffer
 - > **MMD_ISOURCE** specifies ISource, which fetches media data on demand
 - **AEEMediaData::pData** is set based on **clsData**
 - **AEEMediaData::dwSize** is size in bytes of the buffer (**MMD_BUFFER**)



Media Setup: Media type known

```
// Set sample.mp3, located in app dir, as media data
{
    AEEMediaData    md;
    md.clsData = MMD_FILE_NAME;    // pData is file name
    md.pData = "sample.mp3";    // Media file name
    md.dwSize = 0;
    IMEDIA_SetMediaData(pme->pIMedia, &md); //Set media data
}
```



Media Setup: Media type unknown

- Media type not known. Feed media data to `IMEDIAUTIL_CreateMedia()` API, which
 - Detects media type
 - Creates relevant `IMedia` object
 - Sets media data
- Returned `IMedia` object will be in `READY` state and ready to go
- See code snippet in next slide

Media Setup: Media type unknown

```
// Play sample.mp3 file located in the app directory.
static void CApp_PlayMedia(CApp *pme)
{
    IMediaUtil      *pIMediaUtil;
    AEEMediaData    md;

    // Create IMediaUtil instance.
    ISHELL_CreateInstance(pme->a.m_pIShell, AEECLSID_MEDIAUTIL,
                          (void **) &pIMediaUtil);

    // Set sample.mp3 file as source.
    md.clsData = MMD_FILE_NAME;
    md.pData = "sample.mp3";
    md.dwSize = 0;

    // Create IMedia object, set media data, put IMedia object
    // in READY state and retrieve IMedia object pointer.
    IMEDIAUTIL_CreateMedia(pIMediaUtil, &md, &pme->pIMedia);
    // Start playback
    :
}
}
```



Media Setup: Media Events

- IMedia sends asynchronous media events to caller
- Register a callback function to receive media events

```
// Register CApp_MediaNotify() function as  
//callback.
```

```
IMEDIA_RegisterNotify(pme->pIMedia,  
CApp_MediaNotify, pme);
```



IMedia API: Scenarios Media Playback

Media Playback

- Start media playback

```
IMEDIA_Play(pme->m_pIMedia);
```

- Allow the playback to complete by itself or stop it using

```
IMEDIA_Stop(pme->m_pIMedia);
```

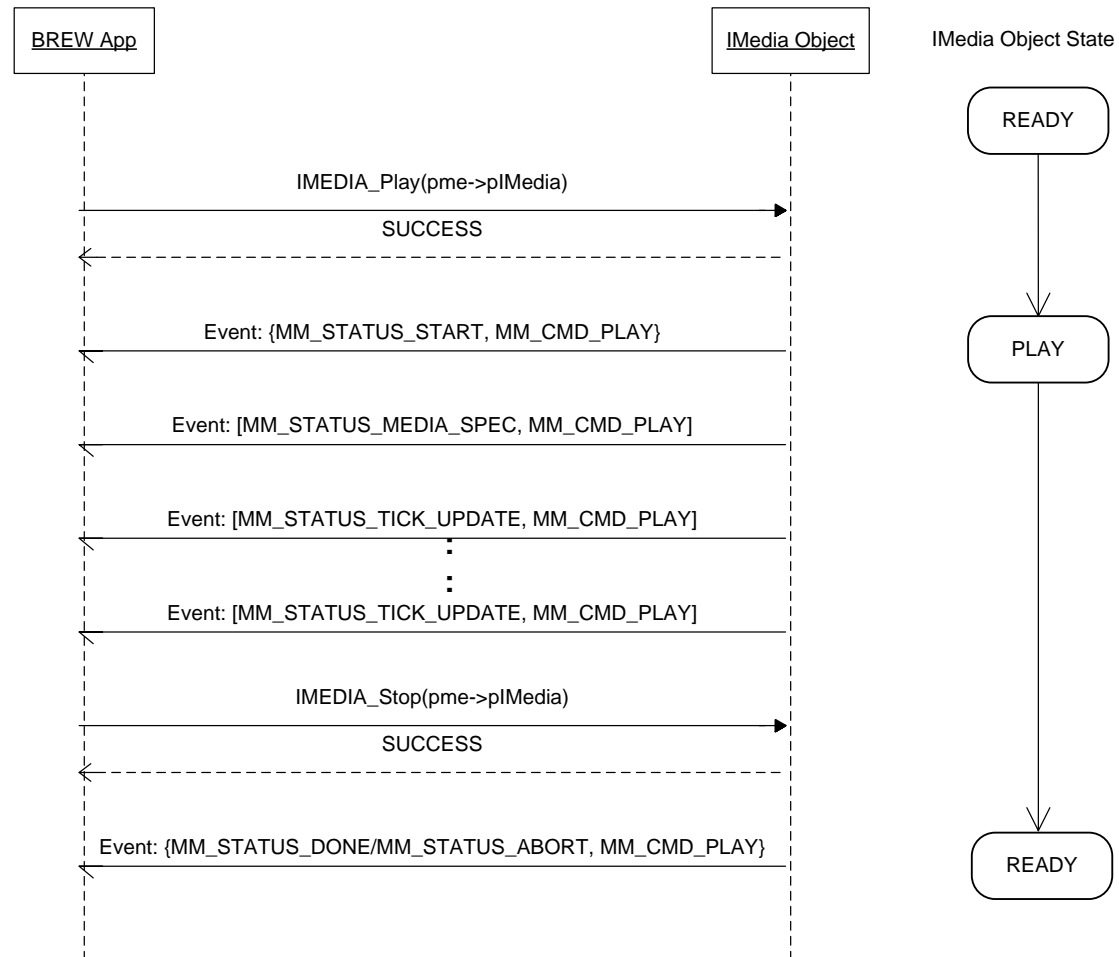
- Handle asynchronous playback related events
 - IMedia object delivers events via registered function
 - Event information is contained in **AEEMediaCmdNotify**
 - > **nStatus** specifies status code (MM_STATUS_XXX)
 - > **nCmd** specifies command code (MM_CMD_PLAY)
 - > **nSubCmd** contains command specific code. For example, for MM_CMD_SETMEDIAPARM, it contains parm ID (MM_PARM_XXX)
 - > **pData** contains context-based data for the event
 - > **dwSize** contains size, in bytes, of *pData



Media Playback: Process Events

- Important status codes sent during playback
 - **MM_STATUS_START** indicates that the media playback operation is about to begin. IMedia object transitions from READY state to PLAY state
 - [**MM_STATUS_MEDIA_SPEC**] gives the media specifications. AEMediaNotify::pData points to the media specifications. E.g. for MP3, it points to AEMediaMP3Spec
 - [**MM_STATUS_TICK_UPDATE**] gives periodic tick updates.
Note: This is not a *real-time* clock feed rather it is a *coarse* progress update
 - **MM_STATUS_PAUSE/MM_STATUS_PAUSE_FAIL** indicates result of pause operation
 - **MM_STATUS_RESUME/MM_STATUS_RESUME_FAIL** indicates result of resume operation
 - **MM_STATUS_SEEK/MM_STATUS_SEEK_FAIL** indicates result of seek operation
 - **MM_STATUS_DONE/MM_STATUS_ABORT** indicates that the media playback has run to its completion or stopped. IMedia object transitions from PLAY state to READY state

Media Playback: Sequence Diagram



Media Playback: Buffer Source

- During setup, set AEEMediaData as follows

```
AEEMediaData md;  
md.clsData = MMD_BUFFER;  
md.pData = pme->m_pMediaBuffer;  
md.dwSize = pme->m_dwMediaBufferSizeInBytes;  
IMEDIA_SetMediaData(pme->m_pIMedia, &md);
```

- Media data memory (pme->m_pMediaBuffer) is
 - Allocated and owned by app
 - Kept valid by the app during lifetime of the IMedia object, i.e., until this IMedia object is completely released
 - Loaded with entire (NOT a fragment of) media content
 - Freed *after* IMedia object is deleted
- Notes:
 - Never specify stack memory as media buffer
 - Buffer once set, using IMEDIA_SetMediaData(), cannot be replaced by another buffer. IMedia doesn't go back to IDLE state



IMedia API: Scenarios Media Streaming



Media Streaming

- IMedia streams media data from ISource object
- Media data can be streamed from network, buffer or file using a concrete ISource object
- Media streams can be of two types
 - *Formatted*: Media data in a well defined format that contains header, encode specs and position of raw data. E.g. Media files with .mp3 or .wav extensions
 - *Raw*: Media data is raw. Encode specs provided by the user separately. E.g. an audio codec or a game engine first specifies encode specs, dynamically generates raw PCM data at various points in time, and streams the data to IMedia object via ISource



Media Streaming

- Provide a concrete ISource implementation or use BREW supplied one
- Create the ISource object, which is
 - Owned by app
 - Kept valid by the app during lifetime of the IMedia object, i.e., until this IMedia object is completely released
 - Ready to feed data when IMedia object makes buffer
 - Released *after* IMedia object is deleted
- Initialize AEEMediaDataEx structure and set media data. See code snippet in the next slide
- Start playback. IMedia object starts asynchronous buffer requests by calling ISOURCE_Read()

Media Streaming: Setup

```
static void CApp_SetupSource(CApp * pme)
{
    AEEMediaDataEx md;
    IFileMgr *pfm; ISourceUtil *psu;

    // STEP #1: Create IMedia PCM object. It will be in IDLE state.
    ISHELLL_CreateInstance(pme->e.m_pIShell, AEECLSID_MEDIAPCM, (void **)&pme->m_pIMedia);

    // STEP #2: Create the concrete ISource object which is the IFile object
    // referring to "sample.wav".
    ISHELLL_CreateInstance(pme->e.m_pIShell, AEECLSID_FILEMGR, (void **)&pfm)
    pme->m_pFile = IFILEMGR_OpenFile(pfm, "sample.wav", _OFM_READ);
    IFILEMGR_Release(pfm);
    ISHELLL_CreateInstance(pme->e.m_pIShell, AEECLSID_SOURCEUTIL, (void **)&psu)
    ISOURCEUTIL_SourceFromAStream(psu, (IAStream *)pme->m_pFile, &pme->m_pISource);
    ISOURCEUTIL_Release(psu);

    // STEP #3: Initialize AEEMediaDataEx with ISource object
    md.clsData = MMD_ISOURCE;// pData is ISource
    md.pData = (void *)pme->m_pISource; // ISource object
    md.dwSize = 0;
    md.dwStructSize = sizeof(md);           // Size of AEEMediaDataEx structure
    md.dwCaps = 0;                          // What capabilities to enable. 0 means all.
    md.bRaw = FALSE;                        // Is this Raw data? Set it to no (FALSE)
    md.dwBufferSize = 0;                   // Internal buffer size. 0 means use default.
    md.pSpec = NULL;                       // Valid only for raw data
    md.dwSpecSize = 0;                     // Valid only for raw data

    // STEP #4: Set the media data. IMedia object transitions to READY state.
    IMEDIA_SetMediaDataEx(pme->m_pIMedia, &md, 1);
}
```



Media Streaming: Raw Data

- Encoding specifications need to be supplied by caller
- There is no end to media play
 - IMEDIA_Stop() needs to be explicitly called to stop playback
- In AEEMediaDataEx structure,
 - Set bRaw to TRUE
 - Set pSpec to raw media data encoding specifications. For example, for PCM, point it to AEEMediaWaveSpec
- Same steps as the above streaming code snippet



IMedia API: Scenarios Video Playback



Video Playback

- Video frames can be displayed in two ways
 - Let BREW draw the frames. Use **IMEDIA_SetRect()** to specify the area on the screen
 - Enable *frame callback mechanism* which delivers video frames to app
- Frame callback mechanism
 - Check if frame callback is enabled
 - **nErr = IMEDIA_IsFrameCallback(pme->m_pIMedia, &bFrameCB);**
 - Enable frame callback
 - **nErr = IMEDIA_EnableFrameCallback(pme->m_pIMedia, TRUE);**
 - Start media playback
 - Handle **MM_STATUS_FRAME** event delivered for each frame. See code snippet in the next slide

Video Playback: Frame processing

```
// Process MM_STATUS_FRAME event, retrieve the frame and display it.
static void CApp_MediaEventNotify(CApp *pme, AEEMediaCmdNotify *pcn)
{
    :
    switch (pcn->nStatus)
    {
        :
        case MM_STATUS_FRAME:
        {
            IBitmap *      pFrame;
            AEEBitmapInfo  bi;

            IMEDIA_GetFrame(pme->m_pIMedia, &pFrame);
            IBITMAP_GetInfo(pFrame, &bi, sizeof(bi));
            IDISPLAY_BitBlt(pme->e.m_pIDisplay, 0, 0, bi.cx, bi.cy,
                           pFrame, 0, 0, AEE_RO_COPY);
            IDISPLAY_Update(pme->e.m_pIDisplay);
            IBITMAP_Release(pFrame);
            break;
        }
        :
    }
}
```



IMedia API: Scenarios

Audio-Video Synchronization



Audio-Video Synchronization

- Provides continuous audio render feedback
- Enables apps or extensions to synchronize audio-video rendering
- `MM_PARM_AUDIO_SYNC` enables AV sync

```
IMEDIA_SetParm(pIMedia, MM_PARM_AUDIO_SYNC, MM_AUDIO_SYNC_MODE_TIME,  
                250); // Feedback every 250ms
```
- `MM_STATUS_AUDIO_SYNC` event arrives at specified periodicity with `AEEMediaAudioSync`
- `AEEMediaAudioSync` contains AV sync info
 - `ullTimeStampMS`: Time stamp in ms of when samples are rendered
 - `ullSamples`: Total number of samples rendered so far

Audio-Video Sync: AudioSync Event

```
// Process MM_STATUS_FRAME event, retrieve the frame and display it.
static void CApp_MediaEventNotify(CApp *pme, AEEMediaCmdNotify *pcn)
{
    :
    switch (pcn->nStatus)
    {
        :
        case MM_STATUS_AUDIO_SYNC:
        {
            AEEMediaAudioSync *pas = pcn->pData;
            // Save the current render time stamp
            pme->m_ullTimeStampMS = pas->ullTimeStampMS;
            // Save the total number of rendered samples
            pme->m_ullSamples = pas->ullSamples;

            // [OPTIONAL] To account for latency in delivery of
            // the callback, app can do GETUPTIMEMS() to get the
            // current time and compare it against pas->ullTimeStampMS
        }
    }
    :
}
```



Media Recording

- Create QCP or PCM IMedia instance. Setup media data. See [Media Setup: Type known](#)

- Start media recording

```
IMEDIA_Record(pme->m_pIMedia);
```

- Stop recording

```
IMEDIA_Stop(pme->m_pIMedia);
```

- Handle asynchronous recording related events

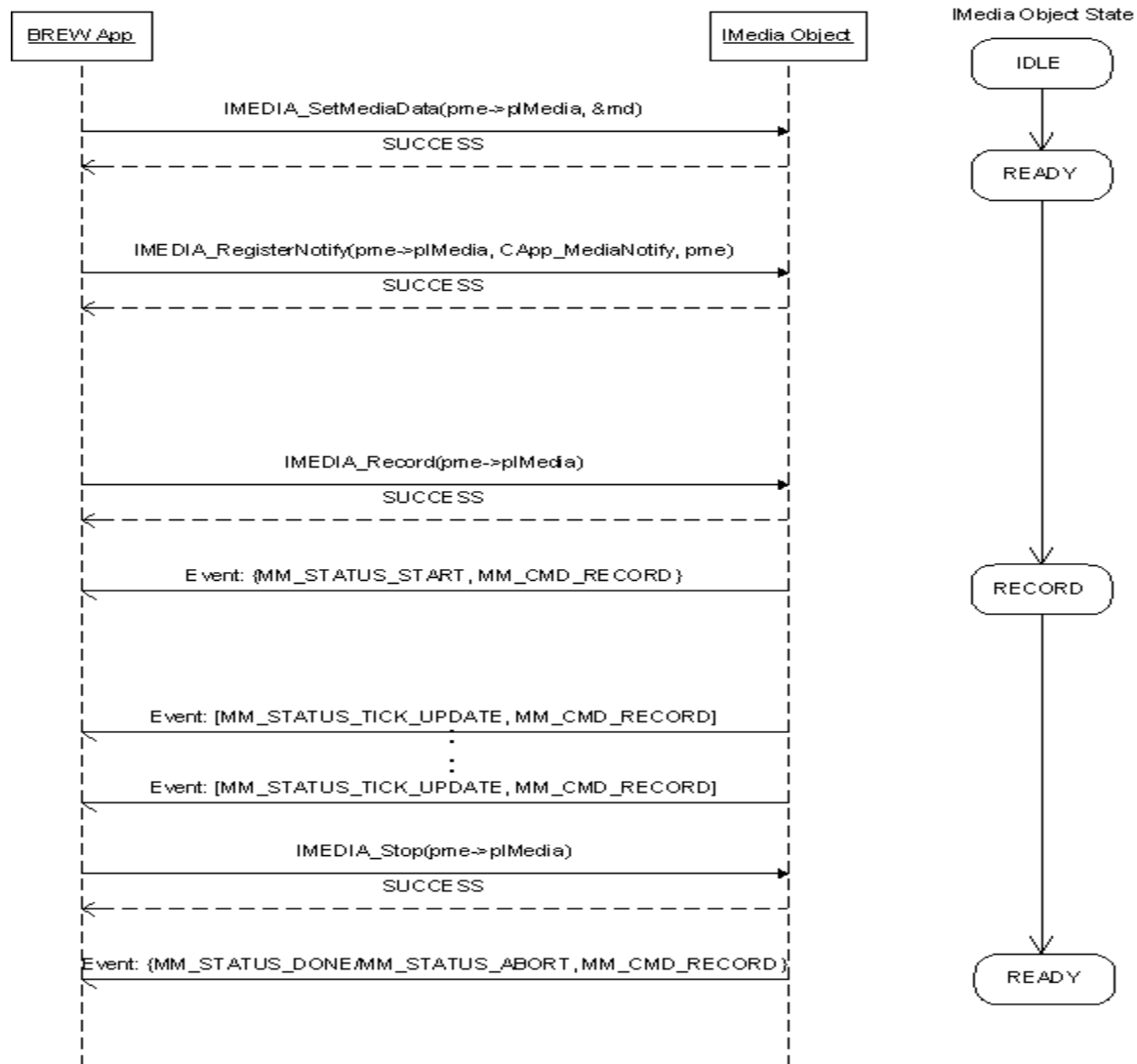
- IMedia object delivers events via registered function
- Event information is contained in **AEEMediaCmdNotify**
 - > **nStatus** specifies status code (MM_STATUS_XXX)
 - > **nCmd** specifies command code (MM_CMD_PLAY)
 - > **nSubCmd** contains command specific code. For example, for MM_CMD_SETMEDIAPARM, it contains parm ID (MM_PARM_XXX)
 - > **pData** contains context-based data for the event
 - > **dwSize** contains size, in bytes, of *pData



Media Recording: Process Events

- Important status codes sent during recording
 - **MM_STATUS_START** indicates that the media record operation is about to begin. IMedia object transitions from READY state to RECORD state
 - [**MM_STATUS_TICK_UPDATE**] gives periodic tick updates.
 - **MM_STATUS_PAUSE/MM_STATUS_PAUSE_FAIL** indicates result of pause operation
 - **MM_STATUS_RESUME/MM_STATUS_RESUME_FAIL** indicates result of resume operation
 - **MM_STATUS_DONE/MM_STATUS_ABORT** indicates that the media playback has run to its completion or stopped. IMedia object transitions from RECORD state to READY state

Media Record: Sequence Diagram





Custom IMedia Extension

- Developers/OEMs can create custom IMedia extension that handles new media format(s)
- IMedia extension can be packaged as one module (a Mod and a MIF)
- To create new IMedia extension,
 - Using MIF Editor, in *Extensions* tab, add following entry in *Exported MIME Types* box
 - > Register handler for media MIME type in the MIF.
 - > *MIME Type*: Specify the media MIME type
 - > *Base Class*: Specify AEECLSID_MEDIA (0x01005500)
 - > *Handler Class*: Specify the CLSID of the new IMedia class
 - > Enter the CLSID of the new IMedia class under *Exported Classes*



Custom IMedia Extension

- To create new IMedia extension, (contd...)
 - Implement new IMedia class based on IMedia interface specifications. Important areas are
 - > Play and control APIs & parameters
 - > State machine
 - > Media decoding
 - > Event generation and delivery
 - Media decoding: audio portion
 - > Decode the audio portion of media as raw linear PCM data
 - > Create IMedia object for PCM using AEECLSID_MEDIAPCM
 - > Set up the IMedia object for raw PCM data streaming
 - > Implement ISource to stream raw PCM data
 - > Stream the raw PCM data to IMedia object to render the audio
 - > Take advantage of AV Sync feature, if needed



Custom IMedia Extension

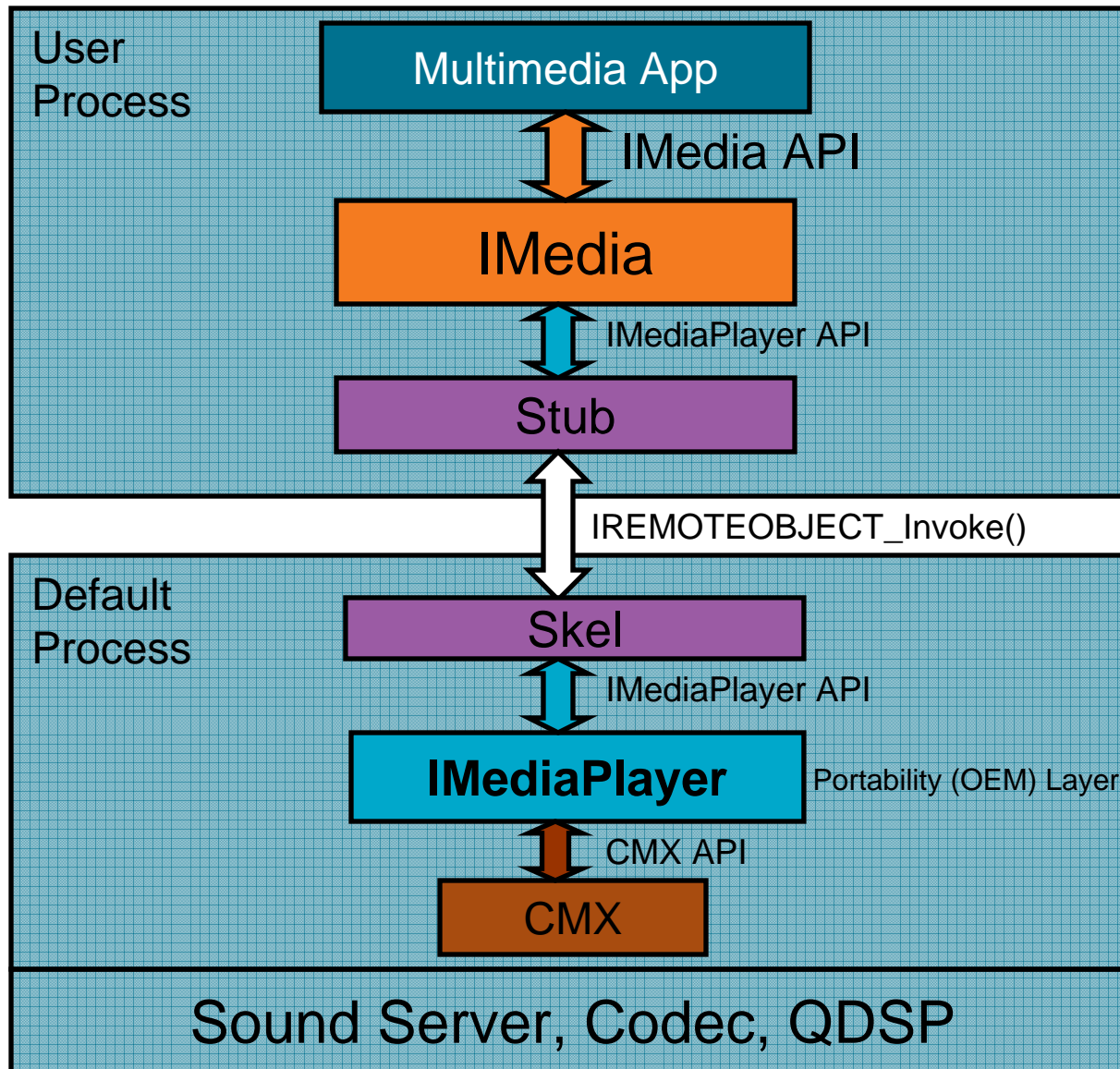
- To create new IMedia extension, (contd...)
 - Media decoding: video portion
 - > Decode the video portion of media to be represented as IBitmap (IDIB) object
 - > Implement frame display by extension (default behavior)
 - > Implement frame callback mechanism (highly recommended)
 - > Deliver frames to app (MM_STATUS_FRAME)
 - > Use AV Sync feature of PCM IMedia object to perform AV sync
 - Testing
 - > Use BREW[®] MediaPlayer app for basic testing
 - > [OEMs only] Use PEK OATMedia module to extensively test the IMedia extension



IMedia in BREW 4.x

- To deploy media service in a separate process
- To enable development of multimedia apps to access secure media service
- To allow usage of IMedia APIs from BREW or non-BREW apps
- In essence, IMedia API can be used from
 - Any process
 - Any BREW or non-BREW app
- Note: IMedia in 4.x is backward compatible. Existing BREW apps using IMedia should continue to work

IMedia in BREW 4.x





BREW ICamera

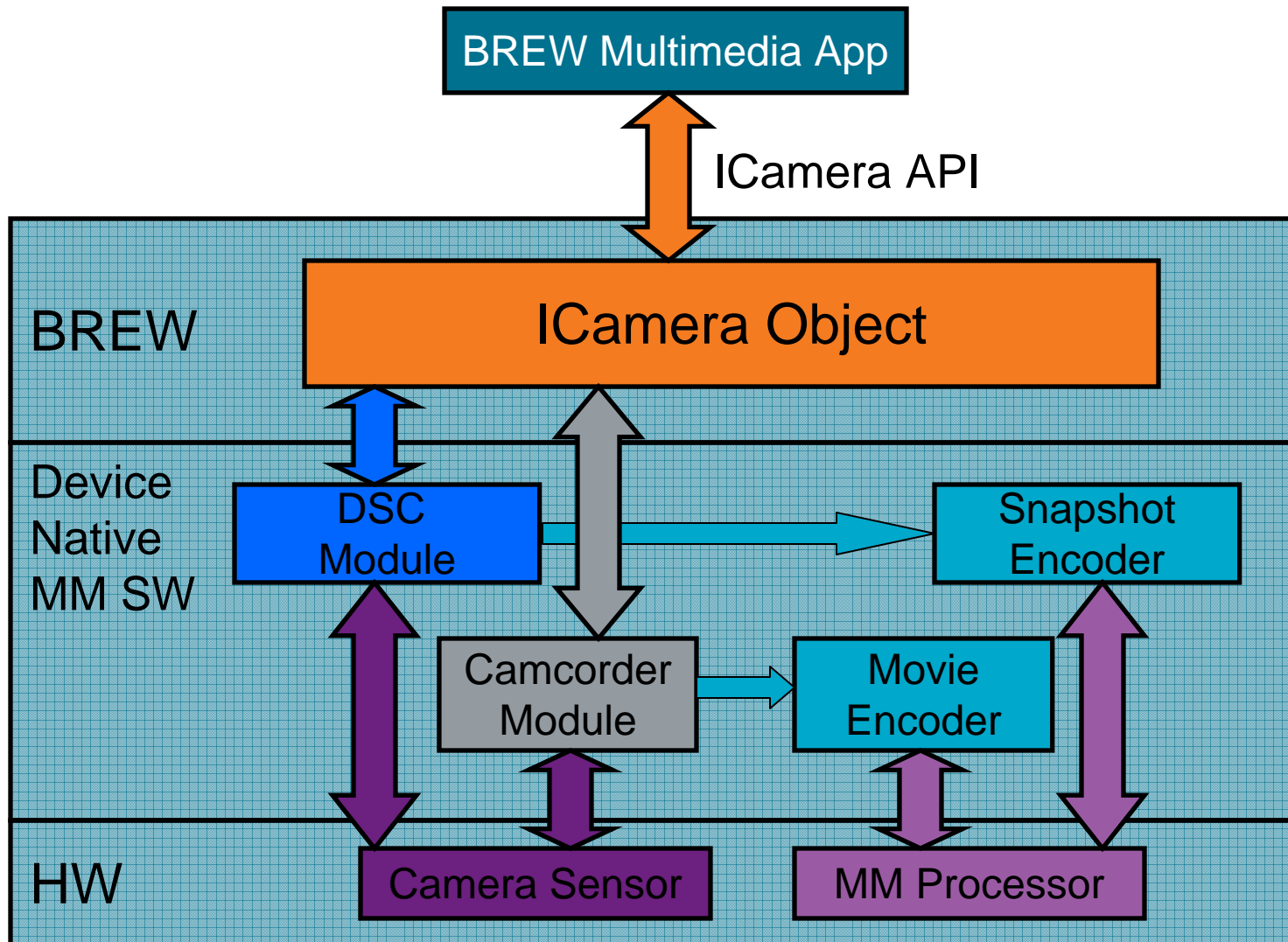
- BREW ICamera Overview
- ICamera API
- ICamera Scenarios
- New Features



ICamera Overview

- Enables access to device camera sensor
- Configures camera in snapshot (DSC) or movie (Camcorder) mode
- Abstracts camera control and hides device complexities
- Delivers frames to app
- Provides easy APIs to record and encode snapshots
- Provides easy APIs to record movies
- Allows access to multiple camera sensors

ICamera: Device View

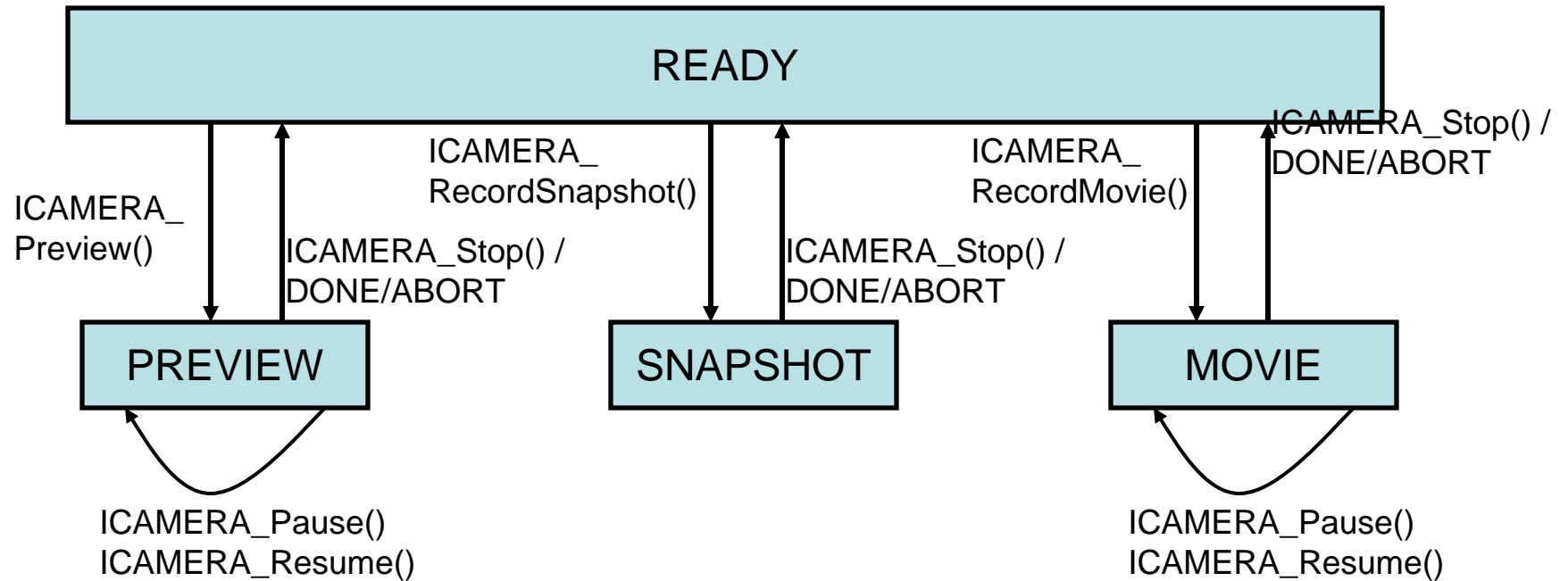




ICamera APIs

- Using **ICamera** APIs, BREW app can
 - Preview camera frames (view finder)
 - Record snapshots and encode them in specified format like JPEG
 - Record movies and encode them in specified format like MPEG4
 - Control camera settings like brightness, zoom, white balance, focus,...
 - Receive camera frames and manipulate them
 - Listen to asynchronous camera events

Camera Modes





ICamera: Scenarios

Camera Setup

Camera Setup

- Create ICamera object

```
// Create an ICamera object to access camera
// Returned ICamera object mode: READY
ISHELL_CreateInstance(pme->a.m_pIShell, AEECLSID_CAMERA,
                      (void **)&pme->m_pICamera);
```

- Register a callback function to receive ICamera events

```
// Register CApp_CameraNotify() function as callback.
ICAMERA_RegisterNotify(pme->pICamera, CApp_CameraNotify,
pme);
```

- Specify intended usage mode for previewing frames

```
// To preview for snapshot, specify CAM_PREVIEW_SNAPSHOT.
ICAMERA_SetParm(pme->pICamera, CAM_PARM_PREVIEW_TYPE,
CAM_PREVIEW_SNAPSHOT, 0);
```

Camera Setup

- Specify display and encode frame sizes

```
ICAMERA_SetDisplaySize(pme->m_pICamera, &pme->m_sizeDisplay);
```

```
ICAMERA_SetSize(pme->m_pICamera, &pme->m_sizeFrame);
```

- Set video/image and audio encoding format

```
// For snapshot, specify JPEG image
```

```
ICAMERA_SetVideoEncode(pme->m_pICamera,  
AEECLSID_JPEG, 0);
```

```
// For movie, specify MPEG4 video and AAC audio
```

```
ICAMERA_SetVideoEncode(pme->m_pICamera,  
AEECLSID_MPEG4, 0);
```

```
ICAMERA_SetAudioEncode(pme->m_pICamera,  
AEECLSID_AAC, 0);
```



ICamera: Scenarios Preview

Preview (View Finder)

- Start the preview

```
ICAMERA_Preview(pme->m_pICamera);
```
- Enter next mode (snapshot or movie) by stopping preview

```
ICamera_Stop(pme->m_pICamera);
```
- Handle asynchronous preview related events
 - ICamera object delivers events via registered function
 - Event information is contained in **AEECameraNotify**
 - > **nStatus** specifies status code (**CAM_STATUS_XXX**)
 - > **nCmd** specifies command code (**CAM_CMD_START**)
 - > **nSubCmd** contains command specific code. For example, for **CAM_CMD_START**, it contains mode (**CAM_MODE_XXX**)
 - > **pData** contains context-based data for the event
 - > **dwSize** contains size, in bytes, of *pData

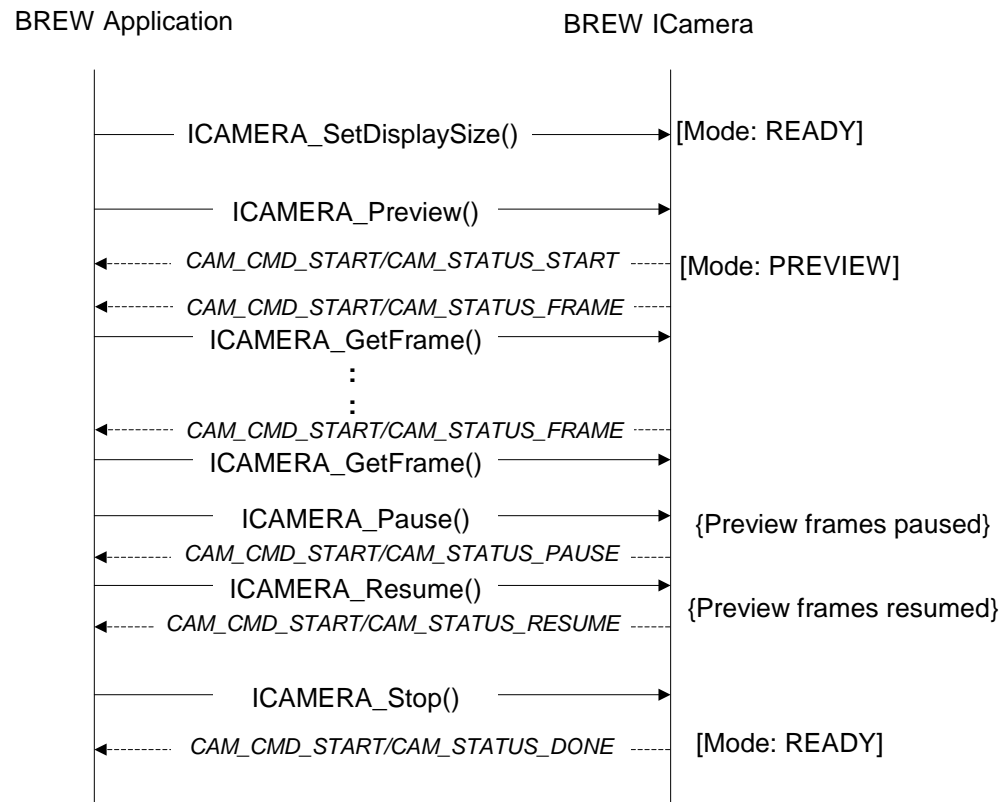
Note: Following slides represent an event as
{nCmd, nSubCmd, nStatus [, pData][, dwSize]}



Preview: Preview Events

- {CAM_CMD_START, CAM_MODE_PREVIEW, CAM_STATUS_START} indicates start of the preview. ICamera object transitions from READY to PREVIEW mode
- {CAM_CMD_START, CAM_MODE_PREVIEW, CAM_STATUS_FRAME} is a frame event received for each frame
 - Retrieve the frame using **ICAMERA_GetFrame()** API
 - Display it at the desired location on the screen
(See code snippet in next slide)
- {CAM_CMD_START, CAM_MODE_PREVIEW, CAM_STATUS_DONE} indicates preview has stopped. ICamera object transitions from PREVIEW to READY mode
- {CAM_CMD_START, CAM_MODE_PREVIEW, CAM_STATUS_ABORT, CAM_EXXX_} can be received at any time if the preview aborts for some reason

Preview: Sequence Diagram



Preview: Frame Event

```
static void CApp_CameraNotify(CApp *pme, AEECameraNotify * pn)
{
    :
    switch (pn->nStatus)
    {
        case CAM_STATUS_START:
            // Preview has begun...
            break;
        case CAM_STATUS_FRAME:
            {
                IBitmap *      pFrame;
                AEEBitmapInfo bi;
                // Get the frame.
                ICAMERA_GetFrame(pme->m_pICamera, &pFrame));
                // Get the bitmap info.
                IBITMAP_GetInfo(pFrame, &bi, sizeof(bi));
                // Display the frame at (0, 0) location of the screen
                IDISPLAY_BitBlt(pme, 0, 0, bi.cx, bi.cy, pFrame, 0, 0, AEE_RO_COPY);
                IBITMAP_Release(pFrame);
                break;
            }
        case CAM_STATUS_DONE:
            // ICAMERA_Stop() stopprf preview operation stopped.
            break;
        case CAM_STATUS_ABORT:
            // Preview got aborted.
            break;
    }
    :
}
```



ICamera: Scenarios Record Snapshot



Record Snapshot

- Two snapshot modes: normal and defer encode
- Set media destination using

```
ICAMERA_SetMediaData(pme->m_pICamera, &md);
```

- Record the snapshot (normal)

```
ICAMERA_RecordSnapshot(pme->m_pICamera);
```

- Raw image is captured
- Raw image is encoded in specified format (JPEG)

- Defer encoding and record snapshot

```
ICAMERA_DeferEncode(pme->m_pICamera, TRUE);
```

```
ICAMERA_RecordSnapshot(pme->m_pICamera);
```

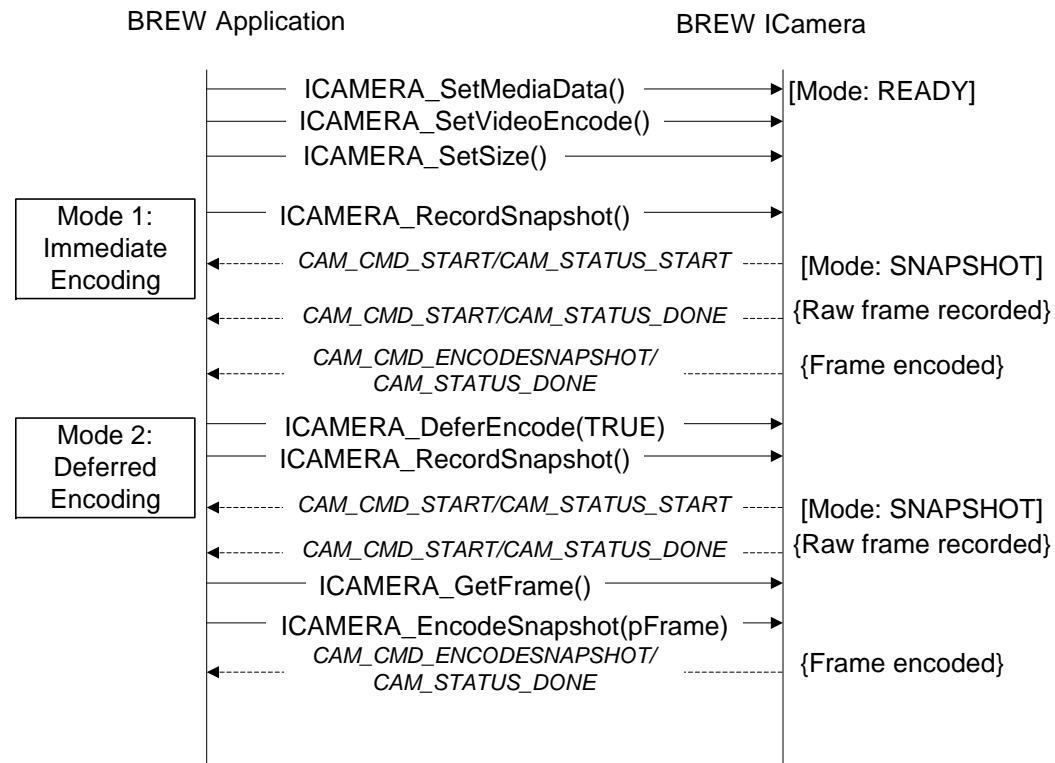
- Encode snapshot explicitly

```
ICAMERA_EncodeSnapshot(pme->m_pICamera);
```

Record Snapshot: Snapshot Events

- {CAM_CMD_START, CAM_MODE_SNAPSHOT, CAM_STATUS_START} indicates start of snapshot recording. ICamera object transitions from READY to SNAPSHOT mode
- {CAM_CMD_START, CAM_MODE_SNAPSHOT, CAM_STATUS_FRAME} is recorded frame
 - Retrieve the frame using **ICAMERA_GetFrame()** API
 - Display it at the desired location on the screen
- {CAM_CMD_START, CAM_MODE_SNAPSHOT, CAM_STATUS_DONE} indicates recording is done. ICamera object transitions from SNAPSHOT to READY mode
- {CAM_CMD_ENCODESNAPSHOT, 0, CAM_STATUS_DONE} indicates encoding is done
- {CAM_CMD_START, CAM_MODE_SNAPSHOT, CAM_STATUS_ABORT, CAM_EXXX_} can be received at any time if the recording aborts for some reason

Record Snapshot: Two Modes





ICamera: Scenarios Record Movie



Record Movie

- Recording a movie triggers following operations
 - Raw frames and audio captured
 - Frames sent to app
 - Simultaneously, frames and audio are encoded in specified format (MPEG4/AAC)

- Set media destination using

```
ICAMERA_SetMediaData(pme->m_pICamera, &md);
```

- Start movie recording

```
ICAMERA_RecordMovie(pme->m_pICamera);
```

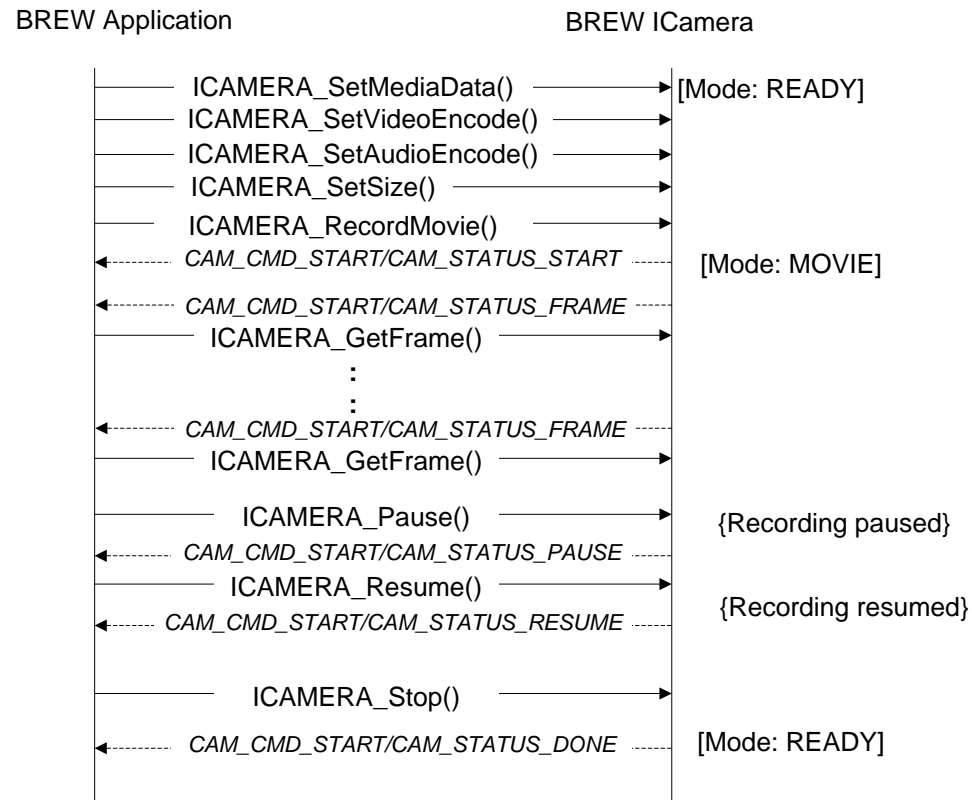
- Stop recording

```
ICAMERA_Stop(pme->m_pICamera);
```

Record Movie: Movie Events

- {CAM_CMD_START, CAM_MODE_MOVIE, CAM_STATUS_START} indicates start of snapshot recording. ICamera object transitions from READY to SNAPSHOT mode
- {CAM_CMD_START, CAM_MODE_MOVIE, CAM_STATUS_FRAME} is recorded frame
 - Retrieve the frame using `ICAMERA_GetFrame()` API
 - Display it at the desired location on the screen
- {CAM_CMD_START, CAM_MODE_MOVIE, CAM_STATUS_DONE} indicates recording is done. ICamera object transitions from SNAPSHOT to READY mode
- {CAM_CMD_START, CAM_MODE_MOVIE, CAM_STATUS_ABORT, CAM_EXXX_} can be received at any time if the recording aborts for some reason

Record Movie: Sequence Diagram





Thank You!