

# I Media

Srinivas Patwari, Staff Engineer

*QUALCOMM Incorporated*



# Overview

- **BREW<sup>®</sup> Multimedia Overview**
- **IMedia API: Basic Operations**
- **IMedia API: Advanced Operations**
- **New and Upcoming Features**
- **Q & A**

# BREW Multimedia Overview

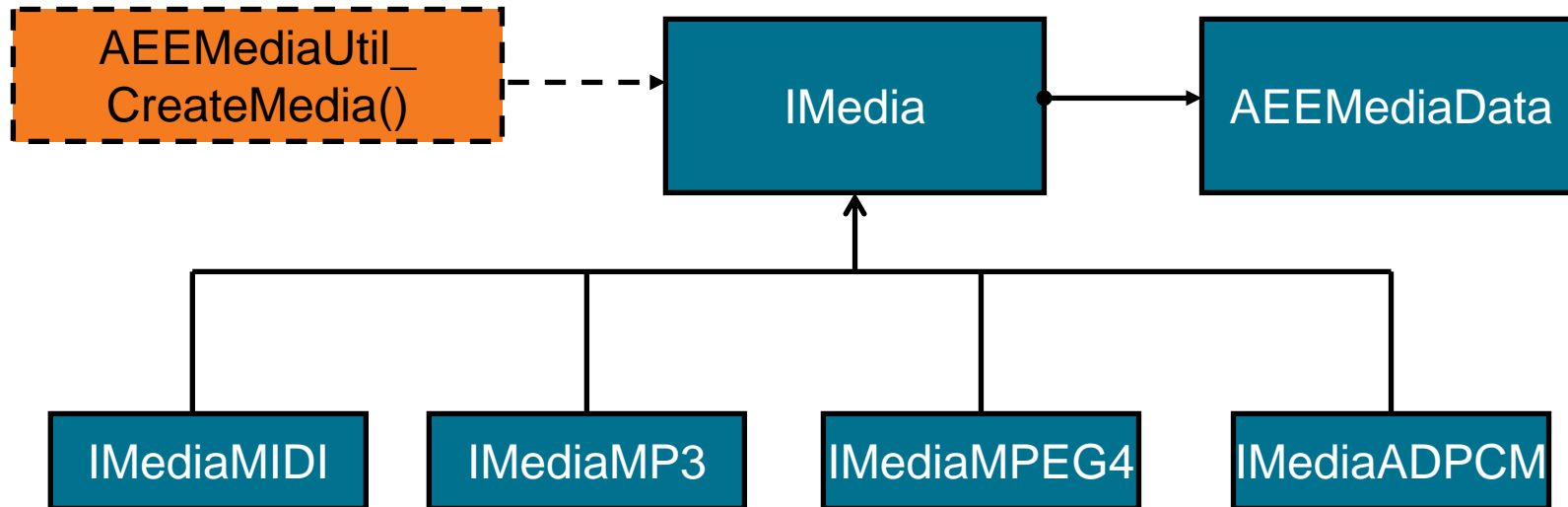
# Multimedia Framework

- **Defines an architectural framework for multimedia apps and extensions**
- **Abstracts multimedia content handling**
- **Hides device complexities in rendering media content**
- **Provides easy interface to audio/video playback/recording and control**
- **Provides building blocks and advanced features for apps like games, mediaplayers**
- **Enables development of new media extensions as plug-ins**
- **Allows dynamic, automatic media type detection by apps**

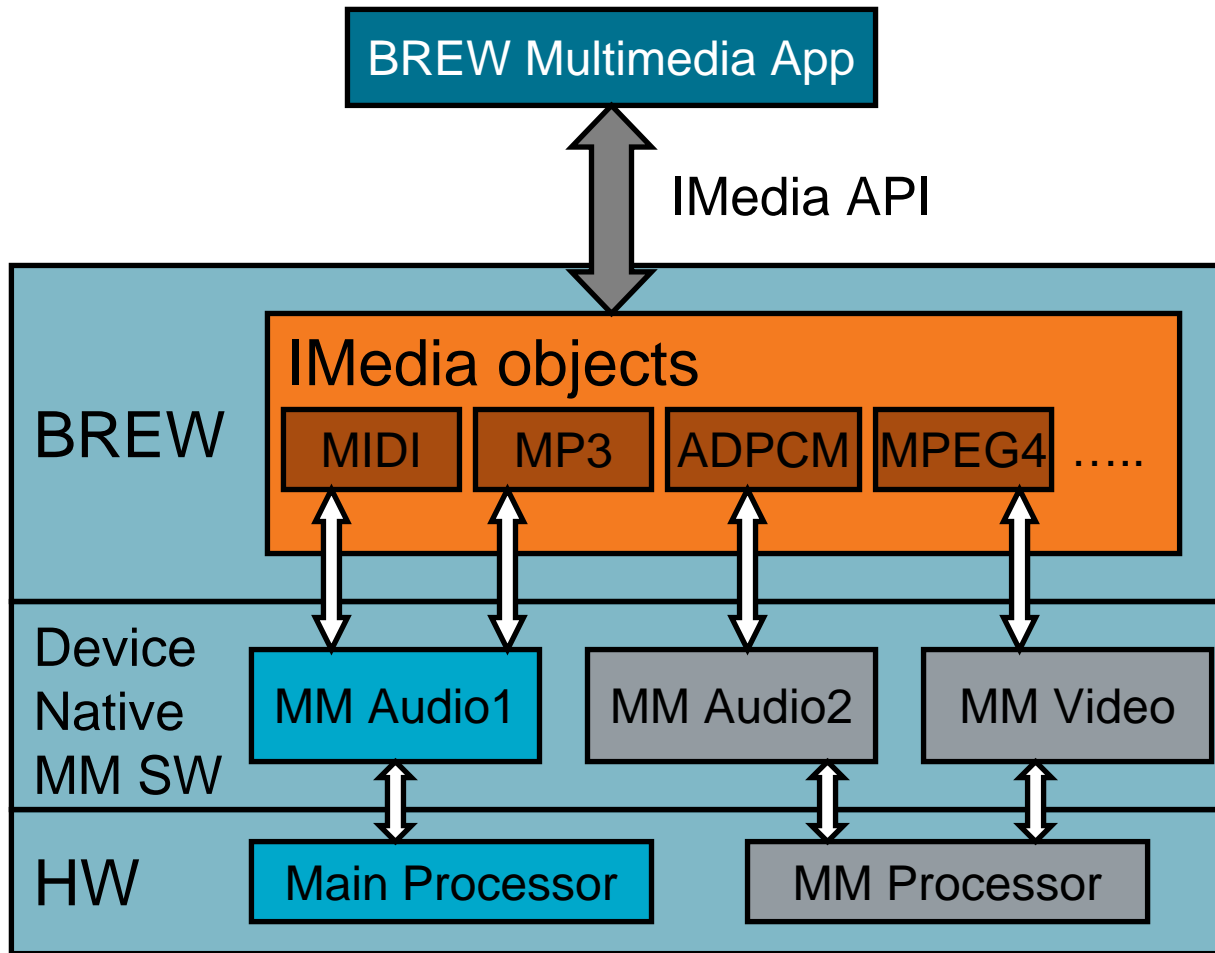
# IMedia Interface

- **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 Class Hierarchy



# IMedia Objects Device View



# IMedia API Overview

- **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 API Overview

- **Media Setup**

- `IMEDIA_SetMediaData()`: Sets media source/sink
- `IMEDIA_RegisterNotify()`: Registers callback that gets asynchronous IMedia object events

- **Playback/Record**

- `IMEDIA_Play()`: Start playback
- `IMEDIA_Record()`: Start record operation

- **Playback/Record Controls**

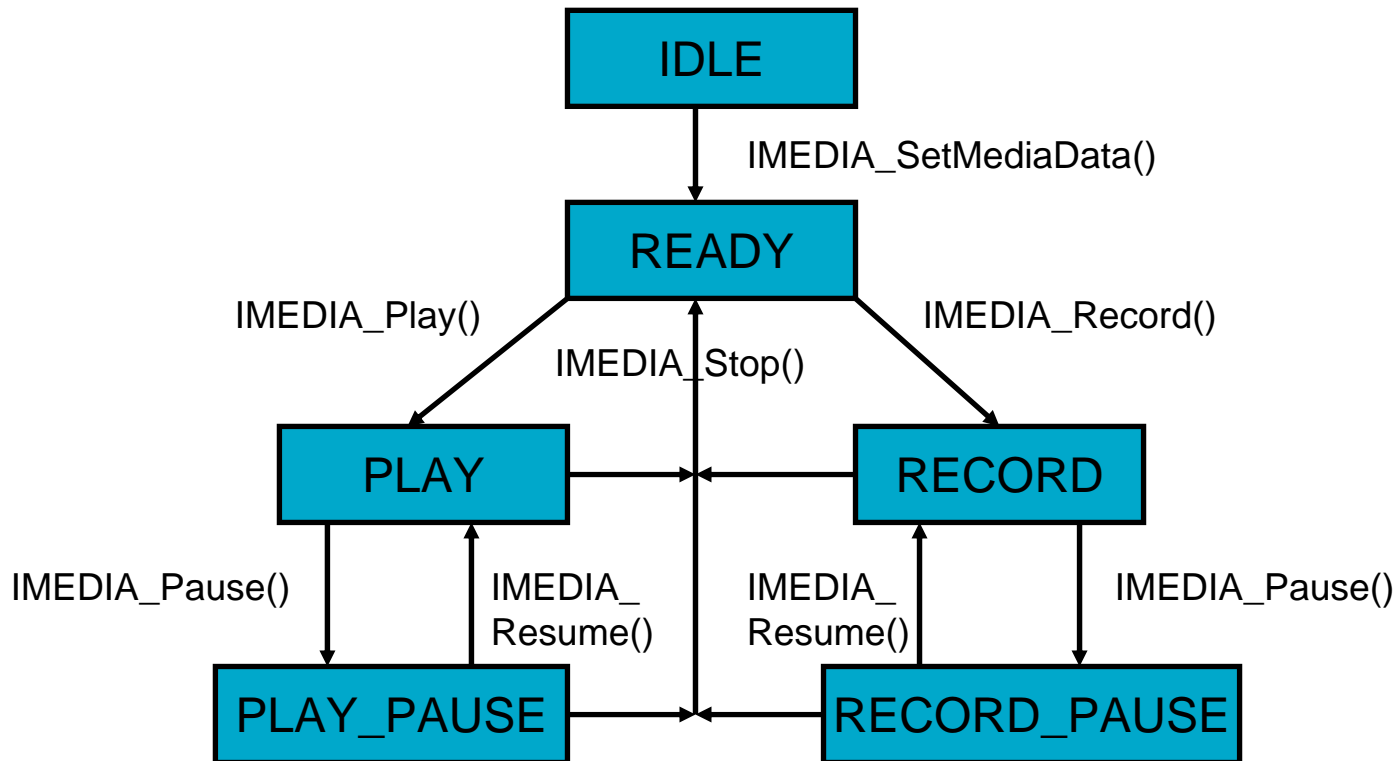
- `IMEDIA_Stop()`: Stop playback/recording
- `IMEDIA_Pause()`: Pause playback/recording
- `IMEDIA_Resume()`: Resume playback/recording
- `IMEDIA_Seek()`: Seek media. Rewind or fast forward

# IMedia API Overview

- **Media Parameters**

- `IMEDIA_SetMediaParm()`: Set media parameter like volume, pan, ...
- `IMEDIA_GetMediaParm()`: Get media parameter
- `IMEDIA_GetState()`: Get media state

# IMedia State Machine



# Reference MSM IMedia Implementations

Format	Class ID	Class Name	MIME	File Ext	Comments
<b>MIDI</b>	AEECLSID_MEDIAMIDI	IMediaMIDI	audio/mid	mid	-
<b>MP3</b>	AEECLSID_MEDIAMP3	IMediaMP3	audio/mp3	mp3	-
<b>QCP</b>	AEECLSID_MEDIAQCP	IMediaQCP	audio/qcp	qcp	Supports recording and simultaneous playback
<b>MIDImsg</b>	AEECLSID_MEDIAMIDI OUTMSG	IMediaMIDIOutMsg	-	-	Sends MIDI message in buffer to the MIDIOut device
<b>PMD (also called CMX)</b>	AEECLSID_MEDIAPMD	IMediaPMD	video/pmd	pmd	-
<b>MPEG4</b>	AEECLSID_MEDIAMPEG4	IMediaMPEG4	video/mp4	mp4	-
<b>MMF (SMAF)</b>	AEECLSID_MEDIAMMF	IMediaMMF	audio/mmf	mmf	-
<b>Phrase</b>	AEECLSID_MEDIAPAPHR	IMediaPhr	audio/spf	spf	Supports simultaneous playback
<b>IMA-ADPCM</b>	AEECLSID_MEDIAADPCM	IMediaADPCM	audio/wav	wav	Supports simultaneous playback
<b>AAC</b>	AEECLSID_MEDIAAAC	IMediaAAC	audio/aac	aac	-
<b>IMelody</b>	AEECLSID_MEDIAIMELODY	-	audio/imy	imy	-

# IMedia API: Basic Operations

# Media Setup

- **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: Contd...

- **Specify media data in AEEMediaData.**  
Contd..

```
// 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
}
```

- **Register a callback function to receive IMedia events**

```
// Register CApp_MediaNotify() function as callback.
IMEDIA_RegisterNotify(pme->pIMedia, CApp_MediaNotify,
pme);
```

# Media Setup: Another Method

- **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: Another Method

```
// 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 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);
```

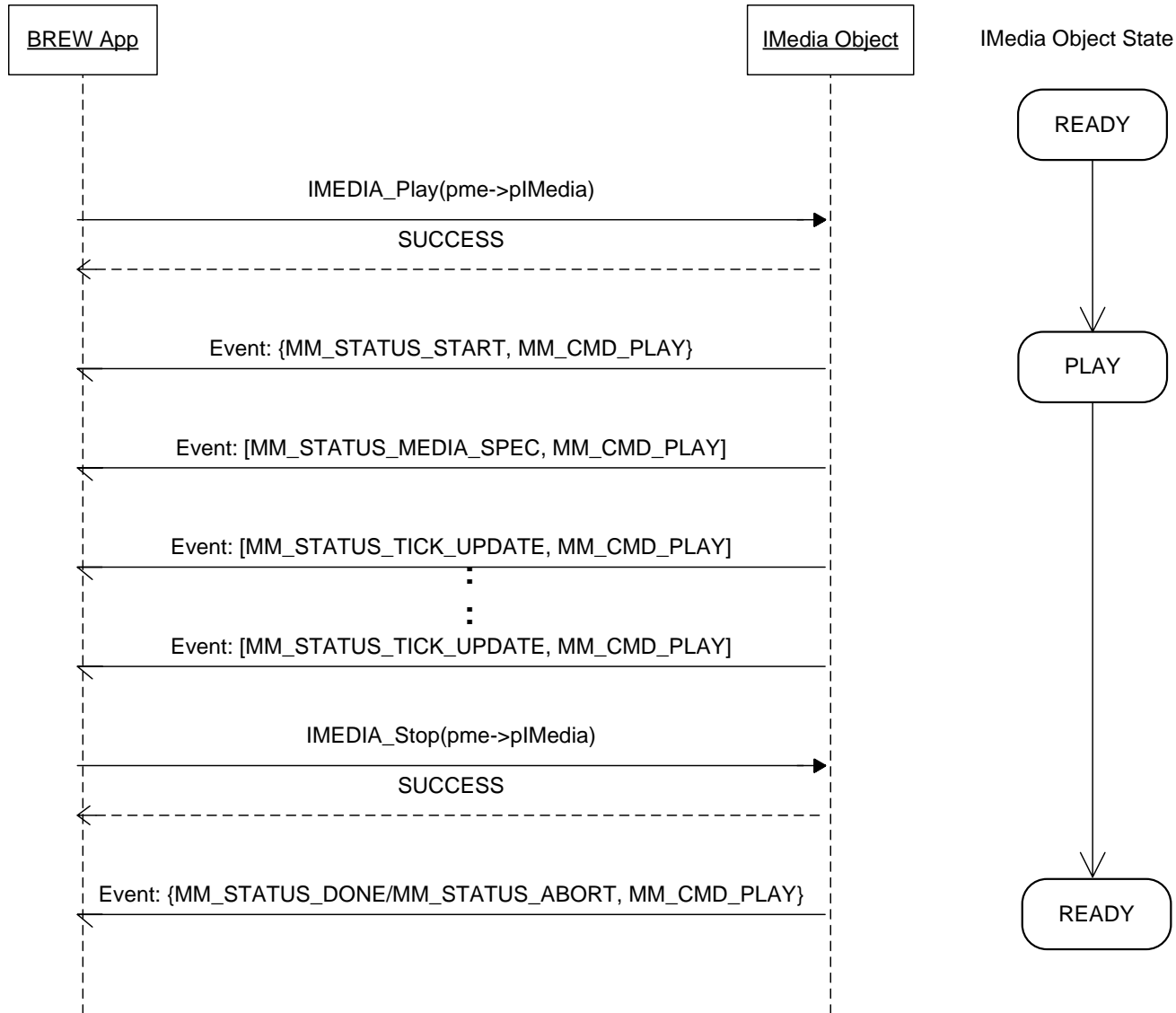
- **Handle asynchronous playback related events**

- **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 Events Processing

- **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. AEEMediaNotify::pData points to the media specifications. E.g. for MP3, it points to AEEMediaMP3Spec
  - **[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 Controls

- **Pause the play**

```
IMEDIA_Pause(pme->m_pIMedia); // MM_STATUS_PAUSE event triggered
```

- **Resume the play**

```
IMEDIA_Resume(pme->m_pIMedia); // MM_STATUS_RESUME event triggered
```

- **Rewind or fast forward the play**

```
IMEDIA_Seek(pme->m_pIMedia, MM_SEEK_CURRENT, pme->lTimeMS);  
// MM_STATUS_SEEK event triggered
```

- **Change the play tempo**

```
IMEDIA_SetTempo(pme->m_pIMedia, pme->nTempoFactor);  
// (MM_STATUS_SETMEDIAPARM, MM_PARM_TEMPO) event triggered
```

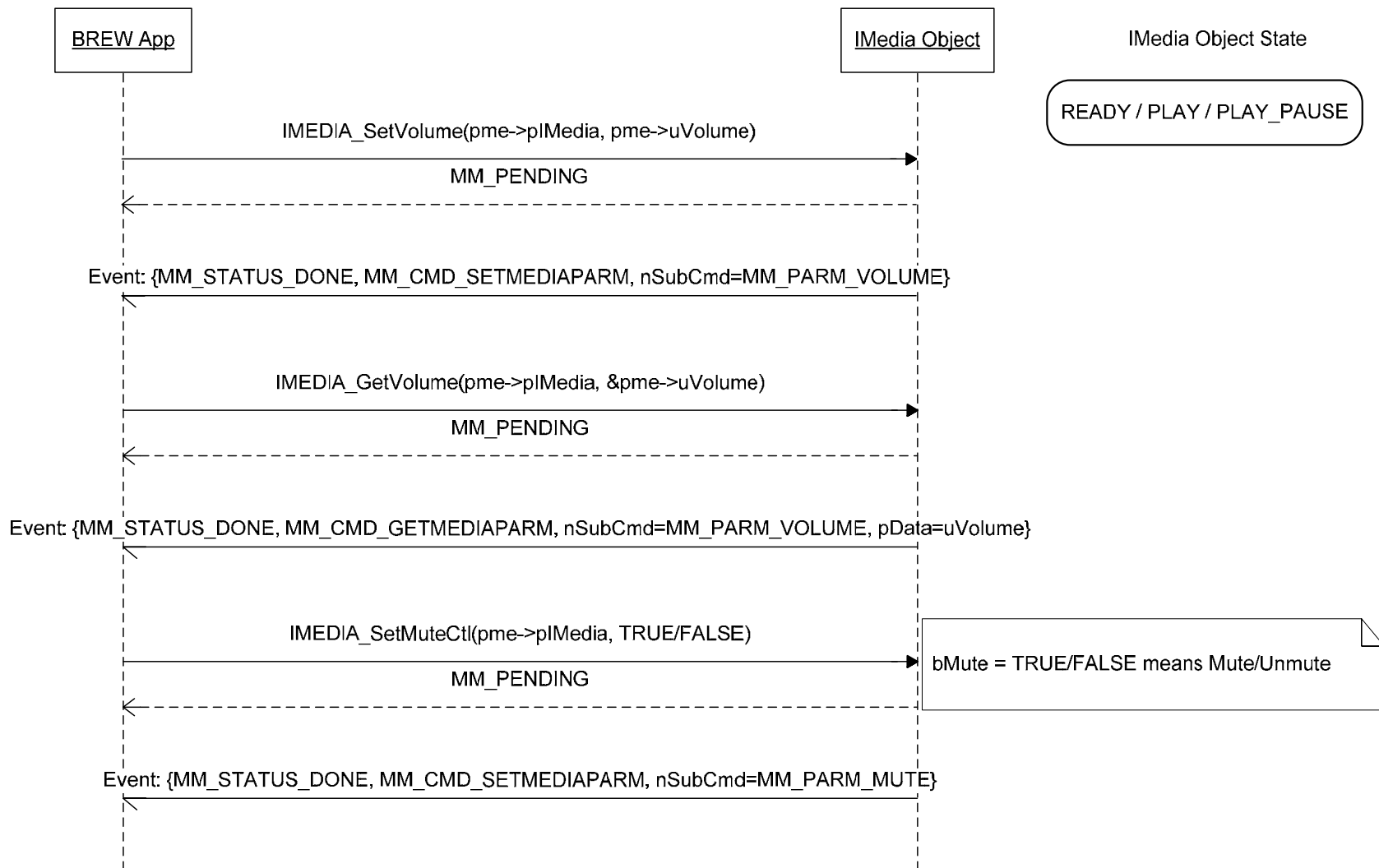
- **Pan the play across stereo speakers/headset**

```
IMEDIA_SetPan(pme->m_pIMedia, pme->nPanFactor);  
// (MM_STATUS_SETMEDIAPARM, MM_PARM_PAN) event triggered
```

- **Change the volume**

```
IMEDIA_SetVolume(pme->m_pIMedia, pme->uVolume);  
// (MM_STATUS_SETMEDIAPARM, MM_PARM_VOLUME) event triggered
```

# Media Control Sequence Diagram



# Media Info

- To get media class ID, use

```
IMEDIA_GetClassID(pme->m_pIMedia, &clsMedia);
```

- To get state of media anytime, use

```
IMEDIA_GetState(pme->m_pIMedia, &nState, &pbStateChanging);
```

- To get total playback time of media, use

```
IMEDIA_GetTotalTime(pme->m_pIMedia, &nState, &pbStateChanging);
```

- To change the play tempo

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

```
// MM_STATUS_GETTOTALTIME event triggered. pData contains
```

```
// total time in milliseconds
```

- To get media capabilities, which tells if it has audio/video/text, use  
**MM\_PARM\_CAPS**

```
uint32    dwCaps;
```

```
IMEDIA_GetMediaParm(pme->m_pIMedia, MM_PARM_CAPS,
```

```
                (int32 *)&dwCaps, 0);
```

```
if (dwCaps & MM_CAPS_VIDEO)
```

```
{
```

```
    // Do video related processing.
```

```
}
```

# Media Playback From Buffer

- 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: Advanced Operations**

# 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: Contd...

- 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: Contd...

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

    // STEP #1: Create IMedia PCM object. It will be in IDLE state.
    ISHELL_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".
    ISHELL_CreateInstance(pme->e.m_pIShell, AEECLSID_FILEMGR, (void **)&pfm)
    pme->m_pFile = IFILEMGR_OpenFile(pfm, "sample.wav", _OFM_READ);
    IFILEMGR_Release(pfm);
    ISHELL_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);
}
```

# 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: Contd...

```
// 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;
        }
        :
    }
}
```

# AV Sync

- 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

# AV Sync: Contd...

```
// 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
        }
        :
    }
}
```

# Audio Mixing (Simultaneous Playback)

- IMedia allows audio mixing when *Channel Share* feature is enabled
- Simultaneous plays of channel-share enabled IMedia objects are mixed and output to sound device
- To perform audio mixing,
  - Enable channel share on each IMedia object  
`IMEDIA_EnableChannelShare(pme->m_pIMedia1, TRUE);`
  - Start playback on each media
  - Process events from each IMedia object
  - Issue usual play control commands on each IMedia object

# New IMedia Implementation

- **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*

# New IMedia Implementation: Contd...

- **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

# New IMedia Implementation: Contd...

- **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

# New and Upcoming Features

- **MIDI control**

- [BREW 3.1.4] Multi-sequencer support: simultaneous playback of more than one MIDI (Channel share feature for MIDI)
- [BREW 3.1.5] Adjust multi-sequence playback quality: specify number of notes to be voiced simultaneously (MM\_PARM\_NOTES)
- [BREW 3.1.4] DLS support. IDLS and IDLSLinker

- **SVG**

- [BREW 3.1.4] SVG Tiny 1.1

- **AEEMediaVideo**

- To be released by Q4 2006
- New extension to handle video formats: MPEG4, WM, Real
- Uses QTV engine
- Simulator extension available

**Thank You!**