

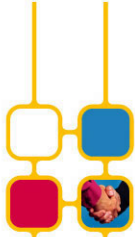
# Game Audio Programming in BREW

**Samir Gupta**

*Principal Engineer*

*QUALCOMM CDMA Technologies*

**June 10, 2004**

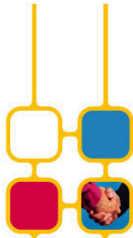


## Presentation Agenda

**Overview of BREW IMEDIA APIs for real-time audio rendering**

**Brief overview of planned 3D audio API**

**Audio synchronization concepts**



# Game Audio Requirements

## Ability to play background music

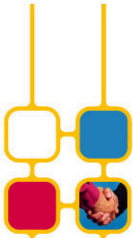
- > Polyphonic MIDI engine is ideally suited
- > Downloadable Sounds (DLS) support for ability to customize musical sounds
- > Multisequencer API for multiple MIDI file playback
- > MIDI API to provide individual sound control

## Simultaneous Audio Objects

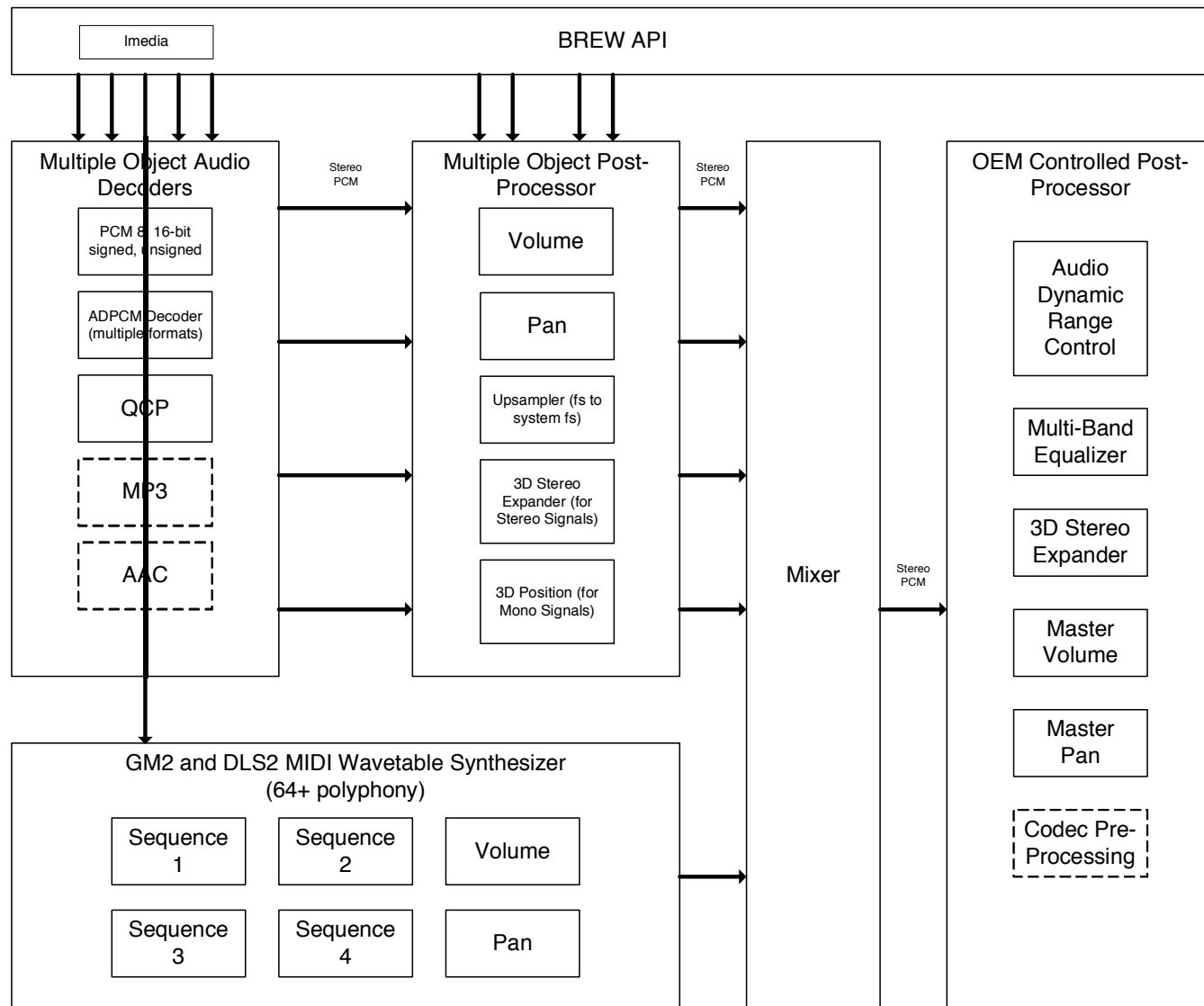
- > Automatic queuing support does not require developer to manage channel IDs
- > Multiple format and sampling rate support
- > Ability to control attributes of the sound object in real-time
- > Allows optimization of quality, bandwidth, storage tradeoff

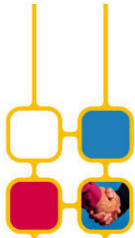
## Individual control of each Audio Object

## Resource manager controls resource overflow



# Game Audio Engine Architecture

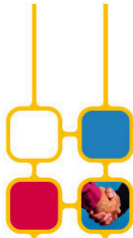




## **IMedia Interface**

**IMedia is an abstract interface that provides APIs for:**

- > Rendering media (audio/video)**
- > Controlling playback like seek, pause, resume**
- > Recording media**
- > Setting/getting media control parameters**
- > Handling asynchronous events from IMedia object**
- > Controlled real-time attributes of IMedia object**
- > Getting media state**



# IMedia API Overview

## Media Setup

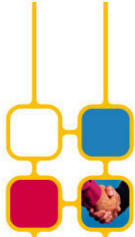
- > **IMEDIA\_SetMediaData():** Sets media source/sink
- > **IMEDIA\_RegisterNotify():** Registers callback that gets asynchronous IMedia object events

## Playback/Record

- > **IMEDIA\_Play():** Starts playback
- > **IMEDIA\_Record():** Starts record operation

## Playback/Record Controls

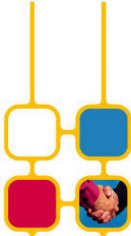
- > **IMEDIA\_Stop():** Stops playback/recording
- > **IMEDIA\_Pause():** Pauses playback/recording
- > **IMEDIA\_Resume():** Resumes playback/recording
- > **IMEDIA\_Seek():** Seeks media. Rewinds or fast forwards



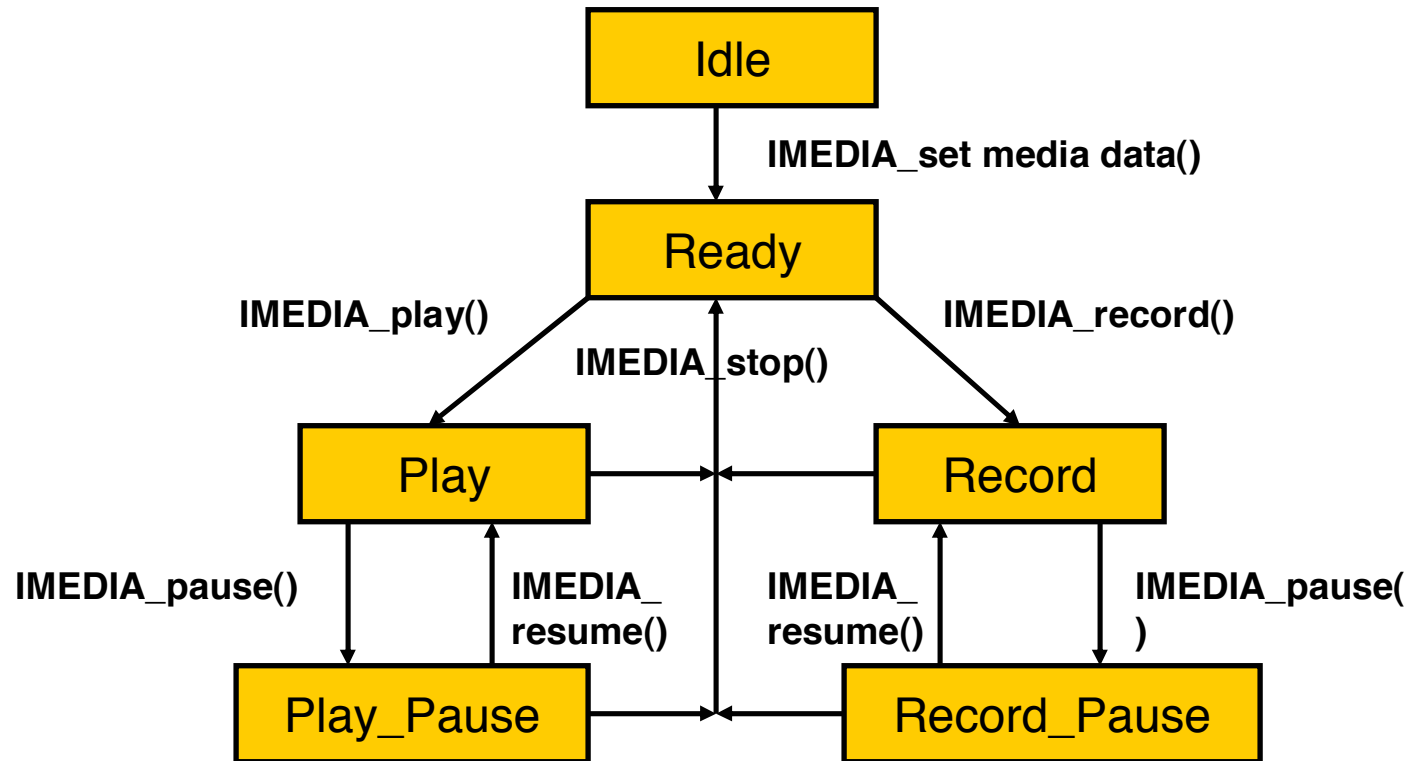
# IMedia API Overview

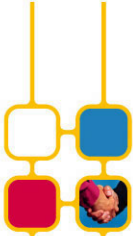
## Media parameters

- > **IMEDIA\_SetMediaData(): Sets media parameter like volume, panÖ**
- > **IMEDIA\_GetMediaParm(): Gets media parameter**
- > **IMEDIA\_GetState(): Gets media state**

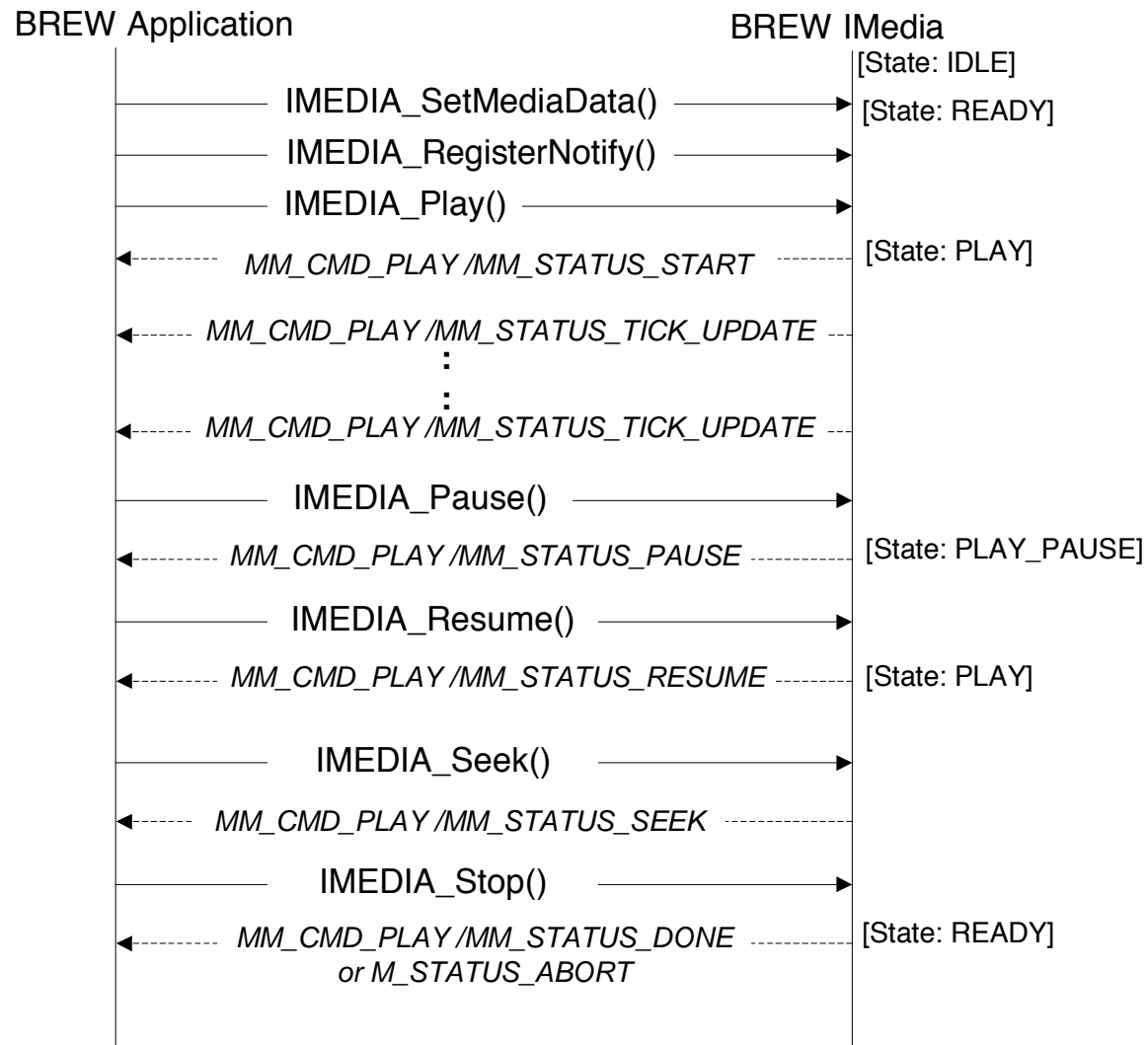


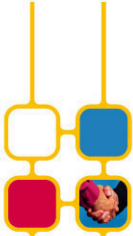
# IMedia State Machine



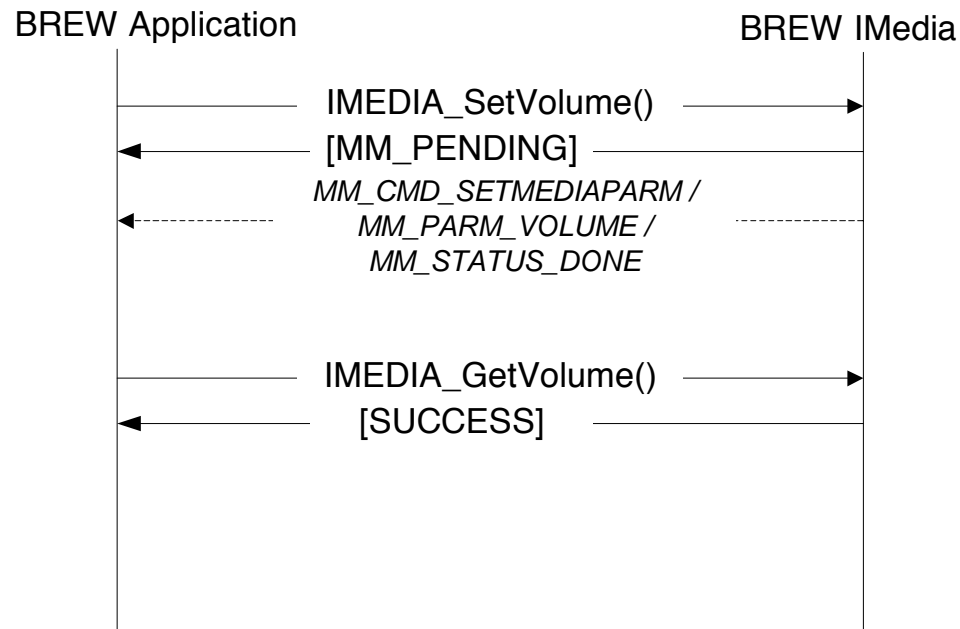


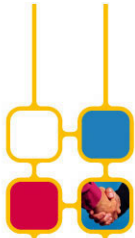
# Media Playback





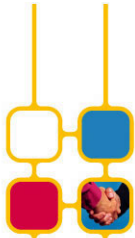
# Media Control Parameters





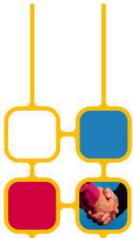
## Reference IMedia Sound Implementations (1)

Format	Class ID	Class Name	MIME	File ext	Comments
MIDI	AEECLSID_MEDIAMIDI	IMediaMIDI	audio/mid	mid	-
MP3	AEECLSID_MEDIAMP3	IMediaMP3	audio/mp3	mp3	-
QCP	AEECLSID_MEDIAQCP	IMediaQCP	audio/qcp	qcp	Supports recording and simultaneous playback
MIDImsg	AEECLSID_MEDIAMIDI OUTMSG	IMediaMIDIOutMsg	-	-	Sends MIDI msg in buffer to the MIDIOut device
CMX/PD	AEECLSID_MEDIAPMD	IMediaPMD	video/pmd video/cmf	pmd	-
MPEG4	AEECLSID_MEDIA MPEG4	IMediaMPEG4	video/mp4	mp4	-



## Reference IMedia Sound Implementations (2)

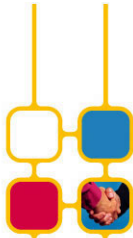
<b>Format</b>	<b>Class ID</b>	<b>Class Name</b>	<b>MIME</b>	<b>File ext</b>	<b>Comments</b>
MMF (SMAF)	AEECLSID_MEDIAMMF	IMediaMMF	audio/mmf	mmf	-
Phrase	AEECLSID_MEDIAPAPHR	IMediaPhr	audio/spf	spf	Supports simultaneous playback
IMA_ADPCM	AEECLSID_ADPCM	IMediaADPCM	audio/wav	wav	Supports simultaneous playback
AAC	AEECLSID_MEDIAAAC	IMediaAAC	audio/aac	aac	-
IMelody	AEECLSID_MEDIAIMELODY	-	audio/imy	imy	-



## IMediaUtil

### Provides media utility services

- > **IMEDIAUTIL\_CreateMedia(): Creates IMedia object based on media data Same as AEMediaUtil\_CreateMedia()**
- > **IMEDIAUTIL\_EncodeMedia(): Simple API to encode new media**
  - **Example: IMEDIAUTIL\_EncodePMD() combines JPEG and QCP to encode a PMD**
- > **New IMEDIAUTILs are planned for various features**



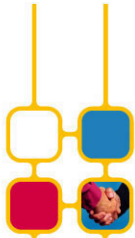
## IMedia Audio Attributes

The following IMedia attributes for an audio object can be set:

- > **IMedia\_SetVolume()** : Sets the volume for given object
- > **IMedia\_SetPolarPosition()** : Set the localized object position, in polar coordinates
- > *IMedia\_SetFrequency()* : Sets the playback frequency for the sound

IMedia attributes can be controlled by the following:

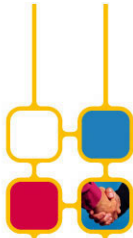
- > **IMedia\_SetPanRate()** : Sets the amount of time in milliseconds, in which a change of attribute will be completed



## IMedia Audio Object Status Functions

The following IMedia attributes can be retrieved:

- > **Imedia\_GetVolume():** ñ Gets the volume for given object
- > **Imedia\_GetPolarPosition():** ñ Gets the object position in polar coordinates
- > *Imedia\_GetFrequency():* – Gets the playback frequency of the object
- > **Imedia\_GetPanRate():** ñ Sets the amount of time in milliseconds, in which a change of attribute will be completed



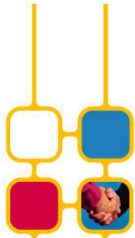
## Scheduling of Audio Objects

### Dynamic

- > The API automatically selects the most important channels for playback
- > API automatically handles the details for the application as well as resource management in HW and DSP

### >Static

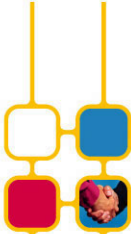
- > API gives application more control over priorities by getting access to information
- > Application Developer has to know details about CPU DSP limitations



## Polar vs Cartesian Coordinates

**Both polar and cartesian coordinates can be used at the same time:**

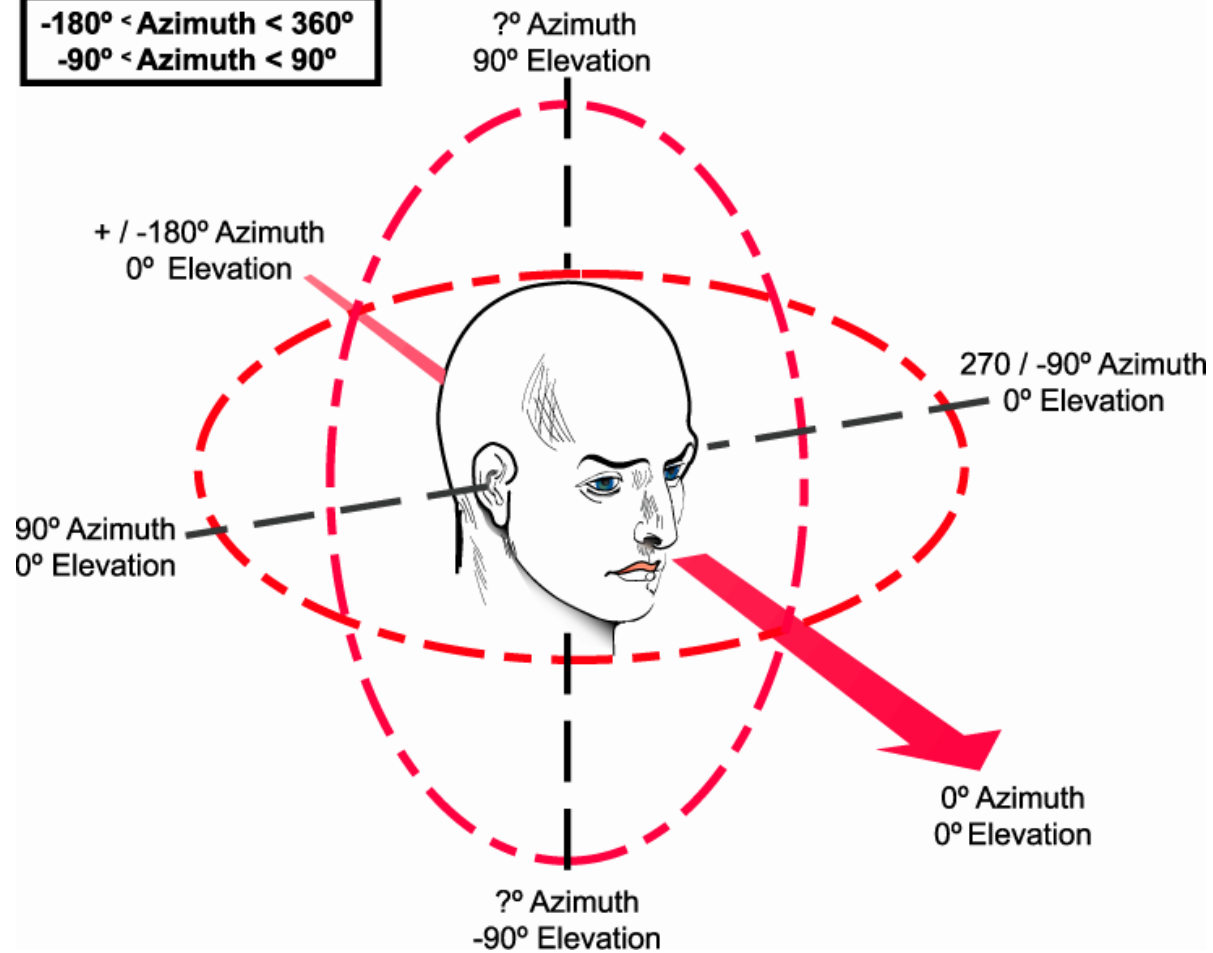
- > Polar coordinates are ideal for audio events that are positioned relative to the listener and tend to move together with the listener**
- > Cartesian coordinates are ideal for independent audio objects (such as a game opponent) that move of their own volition**
- > Normal stereo positioning is also supported to reduce complexity for sounds without strict positioning requirements**

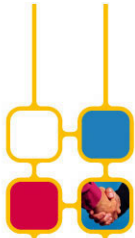


# Polar Coordinates

## Polar Coordinates: Azimuth and Elevation

$-180^\circ < \text{Azimuth} < 360^\circ$   
 $-90^\circ < \text{Azimuth} < 90^\circ$

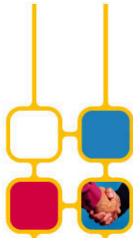




## Listener Position

### These functions control the listener's attributes:

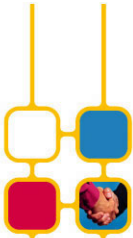
- > **`Imedia_SetListenerPosition()`** : ñ Sets the listener's position. Positions of sound sources are automatically updated when the listener is moved
- > **`Imedia_SetListenerOrientation()`** : ñ Sets the direction of the listener's head
- > *`Imedia_SetListenerVelocity()`* : – Sets the listener's velocity for Doppler effects



## Reverb/Room Control

### These functions control the listener's attributes:

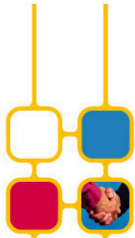
- > **Imedia\_SetReverbEnvironment()**: ñ Sets the room type for reverberation control
- > **Imedia\_SetReverbDecayTime()**: ñ Set the amount of time, in seconds, that the reverberation will decay until it reaches ñ60 dB of the original impulse.
- > **Imedia\_SetReverbDampingFactor()**: ñ Sets the relative decay time of high frequency vs. low frequency content
- > **Imedia\_SetReverbMix()**: - Sets the amount the audio object contributes to the reverb
- > **Imedia\_SetReverbVolume()**: - Sets the overall reverberation volume
- > **Imedia\_SetRoomSize()**: ñ For range effects
- > **Imedia\_SetSpeedOfSound()**: ñ Speed of Sound used for Doppler Effects



## Sound Source Control

### These functions control the listener's attributes:

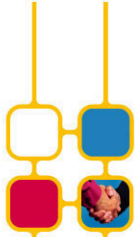
- > `Imedia_SetSourcePosition()`:- `Ö`
- > `Imedia_SetSourceCone()`:- `Ö`
- > `Imedia_SetSourceVelocity()`:- Sets position, orientation, and velocity attributes
- > `Imedia_SetDistanceMapping()`:- Sets the controls for how the volume of a source changes with distance.
- > `Imedia_SetPolarPosition()`:- described earlier



# Audio/Graphics Synchronization

## Synchronization Tools

- > Applications will need feedback from the audio engine to control synchronization of graphics
- > Audio is generally considered to be the static stream while graphics and video adjusts its timing to sync with it
- > Additional APIs are being considered to provide synchronization capabilities to the application
- > Your feedback is appreciated!



## Summary

- > Overview of BREW IMEDIA APIs for real-time audio rendering
- > Brief Overview of planned 3D Positional APIs
- > Audio Synchronization Concepts