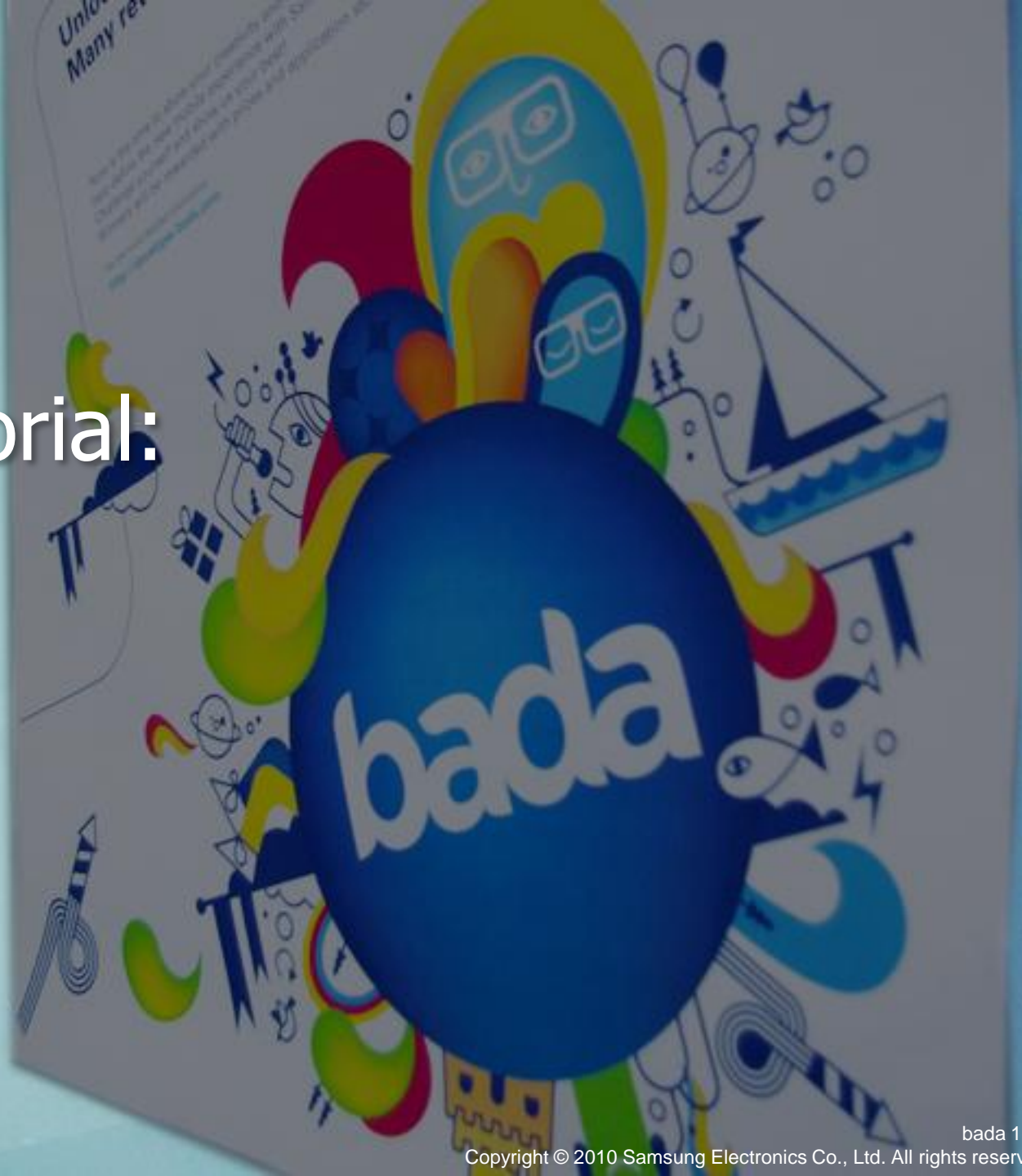


bada Tutorial: Media



Contents (1/2)

- Essential Classes
- Relationships between Classes
- Overview
- Images
 - Example: Decode and Display an Image
- Player
 - Example: Play a Video
 - Sample Application: “MediaPlayer”
- AudioIn
 - Example: Record Audio
- AudioOut
 - Example: Play Audio
- Camera
 - Example: Preview and Take a Picture
 - Sample Application: “CameraCapture”



Contents (2/2)

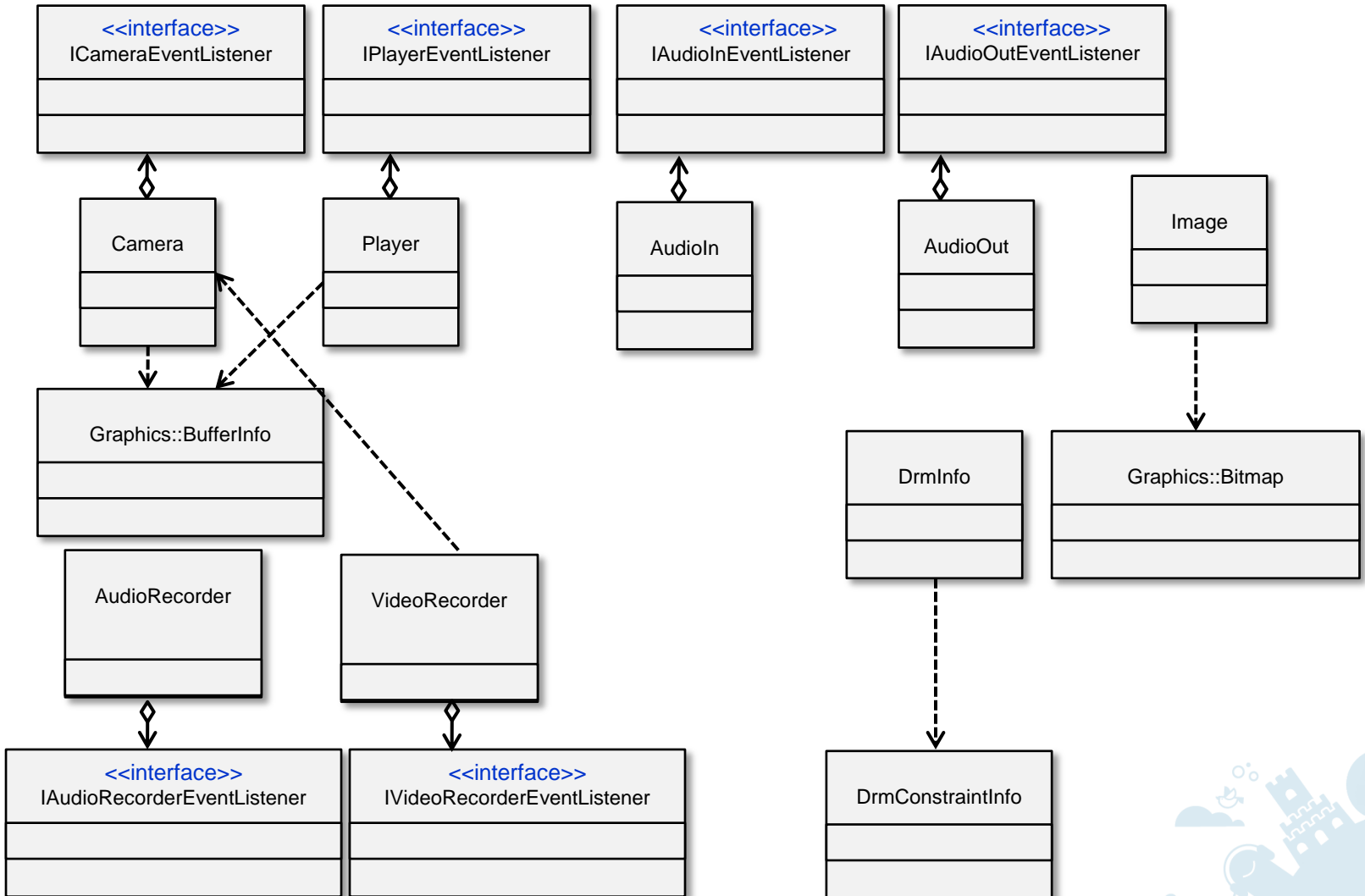
- Recorder
 - Example: Record a Video
 - Sample Application: “VoiceRecorder”
 - Sample Application: “CameraCapture” for Video Recorder
- DRM Information
 - Right ConstraintInfo
 - OMA DRM Format
 - Example: Get DRM Information
- FAQ
- Review
- Answers



Essential Classes

Device feature	Provided by
Plays audio and video content from local storage.	Player
Controls cameras to preview live images or capture images.	Camera
Records audio from a microphone.	AudioRecorder
Records video from the camera sensor.	VideoRecorder
Encodes, decodes, and compresses images.	Image
Provides a container for DRM information for a DRM-protected file.	DrmInfo
Provides DRM usage information for a DRM-protected file.	DrmConstraintInfo
Records audio from the audio input device.	AudioIn
Plays a single audio resource.	AudioOut
Provides a listener for <code>Player</code> .	<code>IPlayerEventListener</code>
Provides a listener for <code>Camera</code> .	<code>ICameraEventListener</code>
Provides a listener for <code>AudioRecorder</code> .	<code>IAudioRecorderEventListener</code>
Provides a listener for <code>VideoRecorder</code> .	<code>IVideoRecorderEventListener</code>
Provides a listener for <code>AudioIn</code> .	<code>IAudioInEventListener</code>
Provides a listener for <code>AudioOut</code> .	<code>IAudioOutEventListener</code>

Relationships between Classes



Overview

- The Media namespace contains classes and interfaces that let developers easily integrate audio, video, and image processing into their applications.
- Key features include:
 - Encoding, decoding, and processing images.
 - Playing audio and video.
 - Recording audio and video.
 - Previewing camera images in real time.
 - Capturing still images from camera sensors.
- For information on bada file directories, and their usage and permissions, see the Io namespace in the Fundamentals tutorial.



Images (1/2)

- `Media::Image` provides methods to encode, decode, and convert images.
- Images can be handled from a file in storage or a buffer in memory.
- The application can encode and decode images with `Media::Image`.
- Optionally, the application can automatically scale images when decoding them.
- The application can compress JPEG images into smaller JPEG images with barely noticeable quality loss.



Images (2/2)

Supported formats:

- Bitmap formats (See the Graphics namespace for details):
 - 16-bit RGB565
 - 32-bit ARGB8888
 - 32-bit R8G8B8A8
- Input formats (decoding):
 - BMP
 - GIF
 - JPEG
 - PNG
 - TIFF
 - WBMP
- Output formats (encoding):
 - BMP
 - JPEG
 - PNG



Example: Decode and Display an Image

Decode a JPEG image and display it.

- Open `<BADA_SDK_HOME>\Examples\MediaContent\Media\src\ImageDecodeExample.cpp`

1. Construct an image object:

```
Image::Construct()
```

2. Decode an image file and get the resulting bitmap:

```
Image::DecodeN()
```

3. Create a rectangle to draw the bitmap on:

```
Graphics::Rectangle()
```

4. Get a Canvas instance:

```
App::IAppFrame::GetCanvasN()
```

5. Draw a bitmap onto the rectangle:

```
Graphics::Canvas::DrawBitmap()
```

6. Show the canvas on the screen:

```
Graphics::Canvas::Show()
```

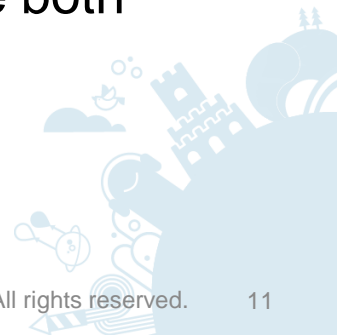


Player (1/4)

- Player can play audio and video from:
 - Local files.
 - Memory buffers.
 - Streaming servers.
- DRM (Digital Rights Management) file can be played automatically.
 - Player chooses the proper methods.
 - Supported DRM specifications are:
 - OMA (Open Mobile Alliance), WMDRM (Windows Media DRM).
- Playback controls:
 - Play, pause, and stop
 - Seek
 - Loop (audio only)
 - Volume and mute
- The application can display video on the background buffer that is embedded in the `Ui::Controls::OverlayPanel` control.

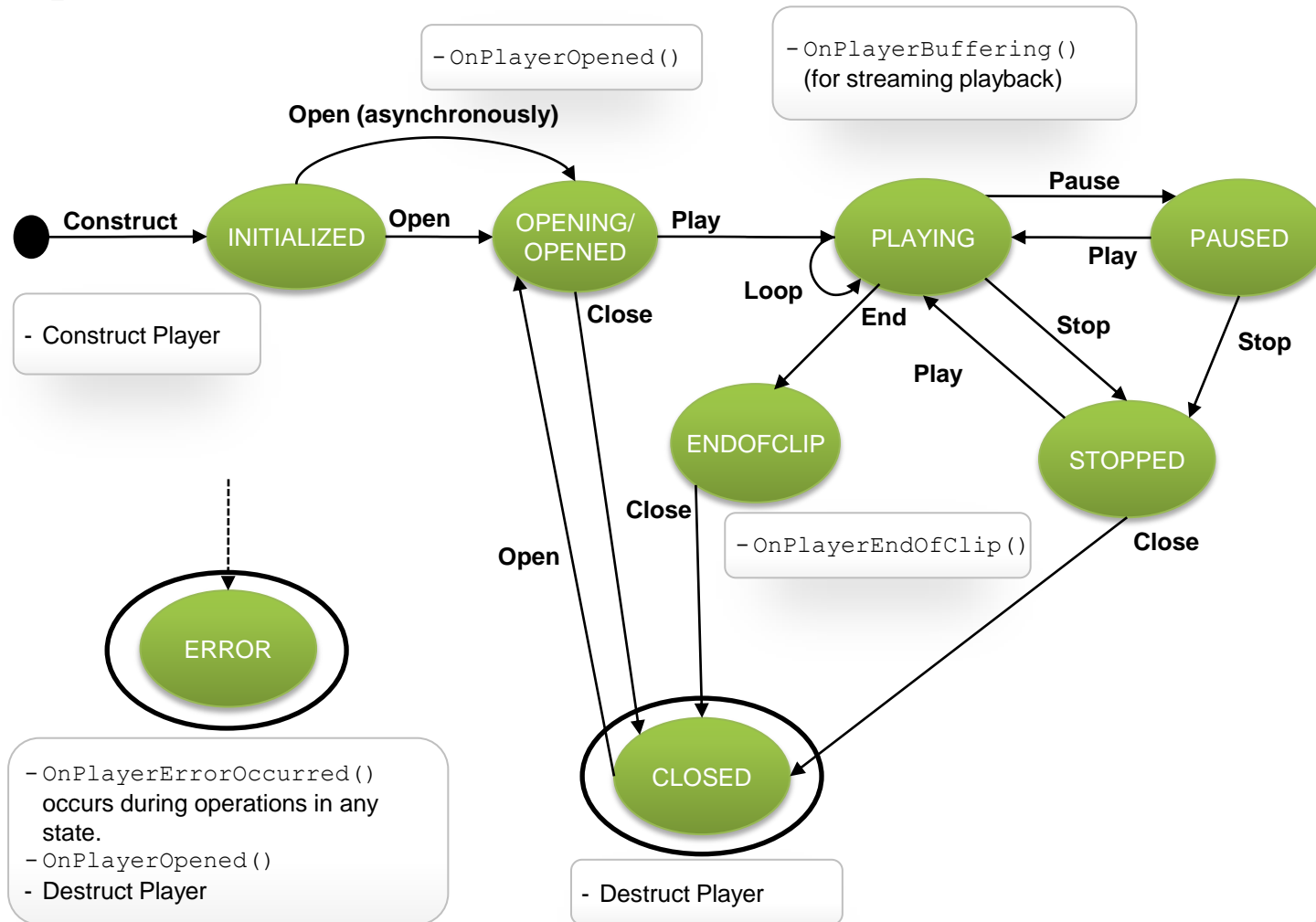
Player (2/4)

- Supported formats:
 - Audio: MP3, AAC, WMA, M4A, XMF, 3GA, MMF, MIDI, WAV, AMR
 - Video: WMV, ASF, MP4, 3GP, AVI
- Multiple audio stream playback is supported:
 - Multiple uncompressed audio files, such as WAV, can be played concurrently.
 - Only one instance of an audio file can be played concurrently for each audio codec.
 - Multiple compressed audio files can be played concurrently if each file is encoded in a different codec.
 - For example: 1 MP3 file + 1 AAC file + 3 WAV files
- All Player operations are synchronous except for the `OpenFile()`, `OpenBuffer()`, and `OpenUrl()` methods, which operate both synchronously and asynchronously.



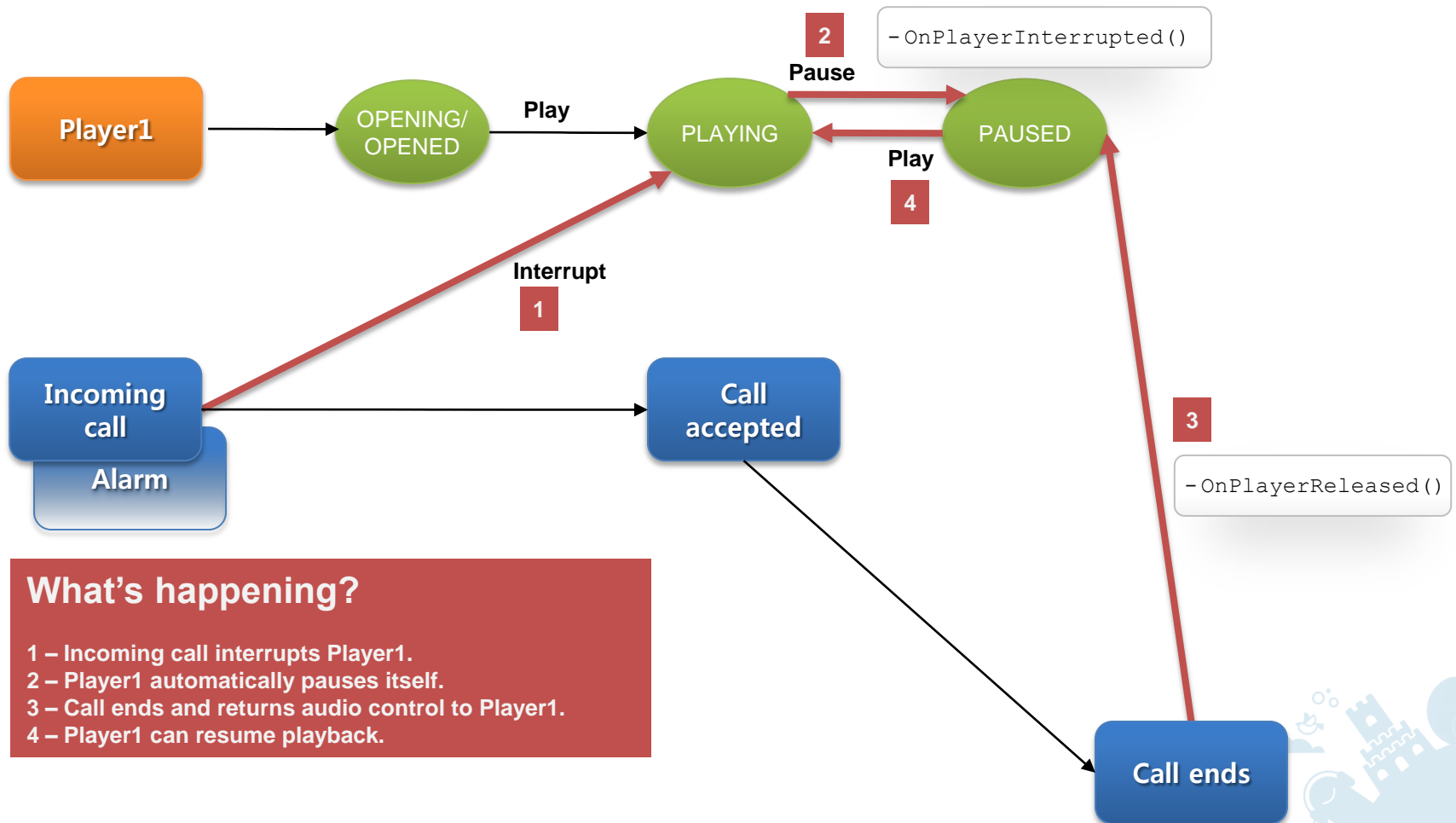
Player (3/4)

Player states:



Player (4/4)

Player states when interrupted by system player:



Example: Play a Video (1/2)

Prepare a listener, overlay panel, and canvas, and create a player.

- Open `\<BADA_SDK_HOME>\Examples\MediaContent\Media\src\PlayerExample.cpp`

1. Create a class and implement listeners of `IPlayerEventListener`.

2. Create an overlay panel:

```
Ui::Controls::OverlayPanel
```

3. Add the overlay panel to a parent container:

```
Ui::Container::AddControl(overlayPanel)
```

4. Get a buffer information from the overlay panel:

```
Ui::Controls::OverlayPanel::  
GetBackgroundBufferInfo()
```

5. Construct a player using the listener and the buffer information:

```
Player::Construct(listener, bufferInfo)
```



Example: Play a Video (2/2)

Open a video file, play, pause, and resume playback.

- Open `\<BADA_SDK_HOME>\Examples\MediaContent\Media\src\PlayerExample.cpp`

6. Open a video media file asynchronously:

```
Player::OpenFile(filePath, true)
```

7. Play the media in the `OnPlayerOpened()` event handler:

```
Player::Play()
```

8. Pause playback:

```
Player::Pause()
```

9. Resume playback:

```
Player::Play()
```

10. Stop playback:

```
Player::Stop()
```

11. Close the file:

```
Player::Close()
```



Sample Application: "MediaPlayer"

- Open the project in `\<BADA_SDK_HOME>\Samples\MediaPlayer\src`.
- For audio and video, `MediaPlayer` shows how to:
 - Play an audio or video file synchronously.
 - Pause and resume playback.
 - Stop playback.
- The `MediaPlayer` sample application can play MP3 audio files as well as MP4 video files.



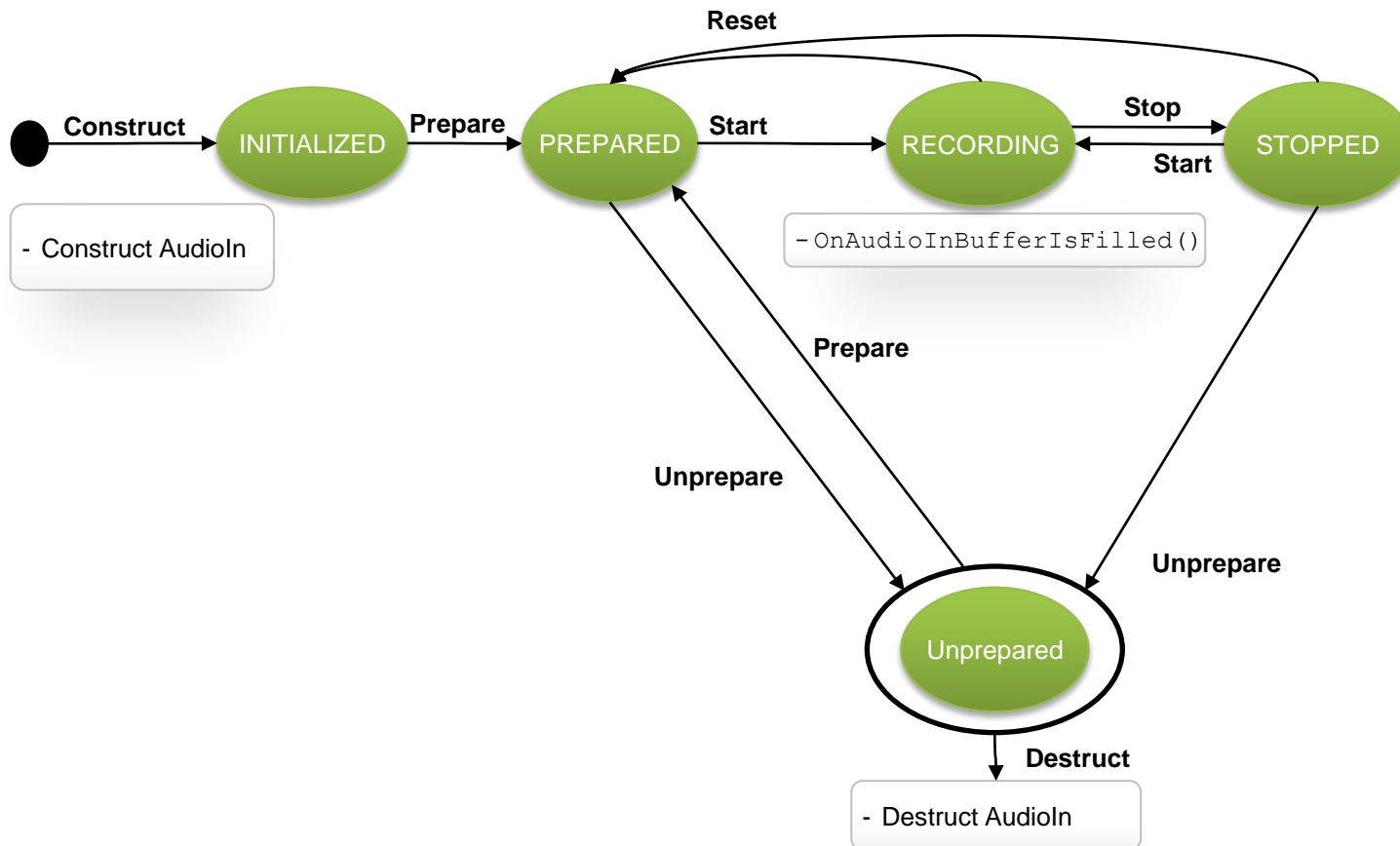
AudioIn (1/3)

- `AudioIn` provides a mechanism for recording raw uncompressed pulse code modulated (PCM) data from the user-defined input device.
- The application is responsible for implementing `IAudioInEventListener` to receive audio PCM data from the input device.
- You must define PCM data settings before recording:
 - Input device type
 - Microphone
 - Audio channels
 - Mono (1 channel)
 - Stereo (2 channels)
 - Audio sample type
 - Unsigned 8-bit PCM
 - Signed little-endian 16-bit PCM
 - Audio sample rate
 - 8000 Hz ~ 48000 Hz



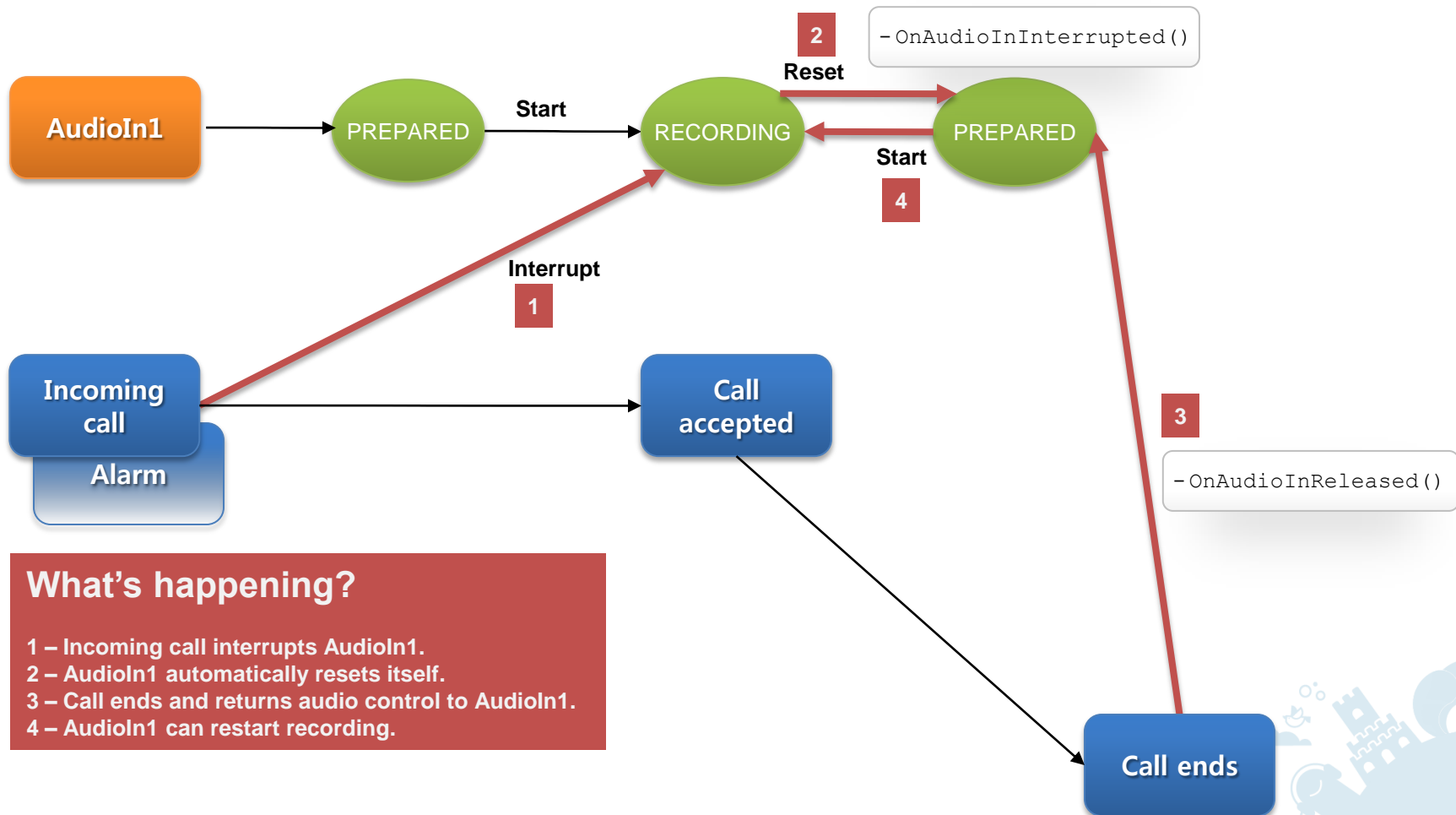
AudioIn (2/3)

AudioIn states:



AudioIn (3/3)

AudioIn states when interrupted by system player:



What's happening?

- 1 – Incoming call interrupts AudioIn1.
- 2 – AudioIn1 automatically resets itself.
- 3 – Call ends and returns audio control to AudioIn1.
- 4 – AudioIn1 can restart recording.

Example: Record Audio (1/2)

Prepare a listener and create an `AudioIn`.

- Open `\<BADA_SDK_HOME>\Examples\MediaContent\Media\src\AudioInExample.cpp`

1. Create the `AudioIn`:

```
__pAudioInInstance = new AudioIn();
```

2. Create the listener:

```
__pMyAudioInListener = new AudioInExampleListener;
```

3. Construct an `AudioIn` instance:

```
__pAudioInInstance->Construct(*__pMyAudioInListener);
```

4. Prepare an `AudioIn` instance:

```
__pAudioInInstance->Prepare(AUDIO_INPUT_DEVICE_MIC,  
AUDIO_TYPE_PCM_U8, AUDIO_CHANNEL_TYPE_STEREO, 8000);
```



Example: Record Audio (2/2)

Prepare buffers to store PCM data and start recording.

- Open `<BADA_SDK_HOME>\Examples\MediaContent\Media\src\AudioInExample.cpp`

5. Prepare buffers to store PCM data:

```
__pByteBuffer1 = new ByteBuffer();  
__pByteBuffer1->Construct(MAX_BUFFER_SIZE);  
__pByteBuffer2 = new ByteBuffer();  
__pByteBuffer2->Construct(MAX_BUFFER_SIZE);  
__pByteBuffer3 = new ByteBuffer();  
__pByteBuffer3->Construct(MAX_BUFFER_SIZE);
```

6. Add buffers:

```
__pAudioInInstance->AddBuffer(__pByteBuffer1);  
__pAudioInInstance->AddBuffer(__pByteBuffer2);  
__pAudioInInstance->AddBuffer(__pByteBuffer3);
```

7. Start recording:

```
__pAudioInInstance->Start();
```



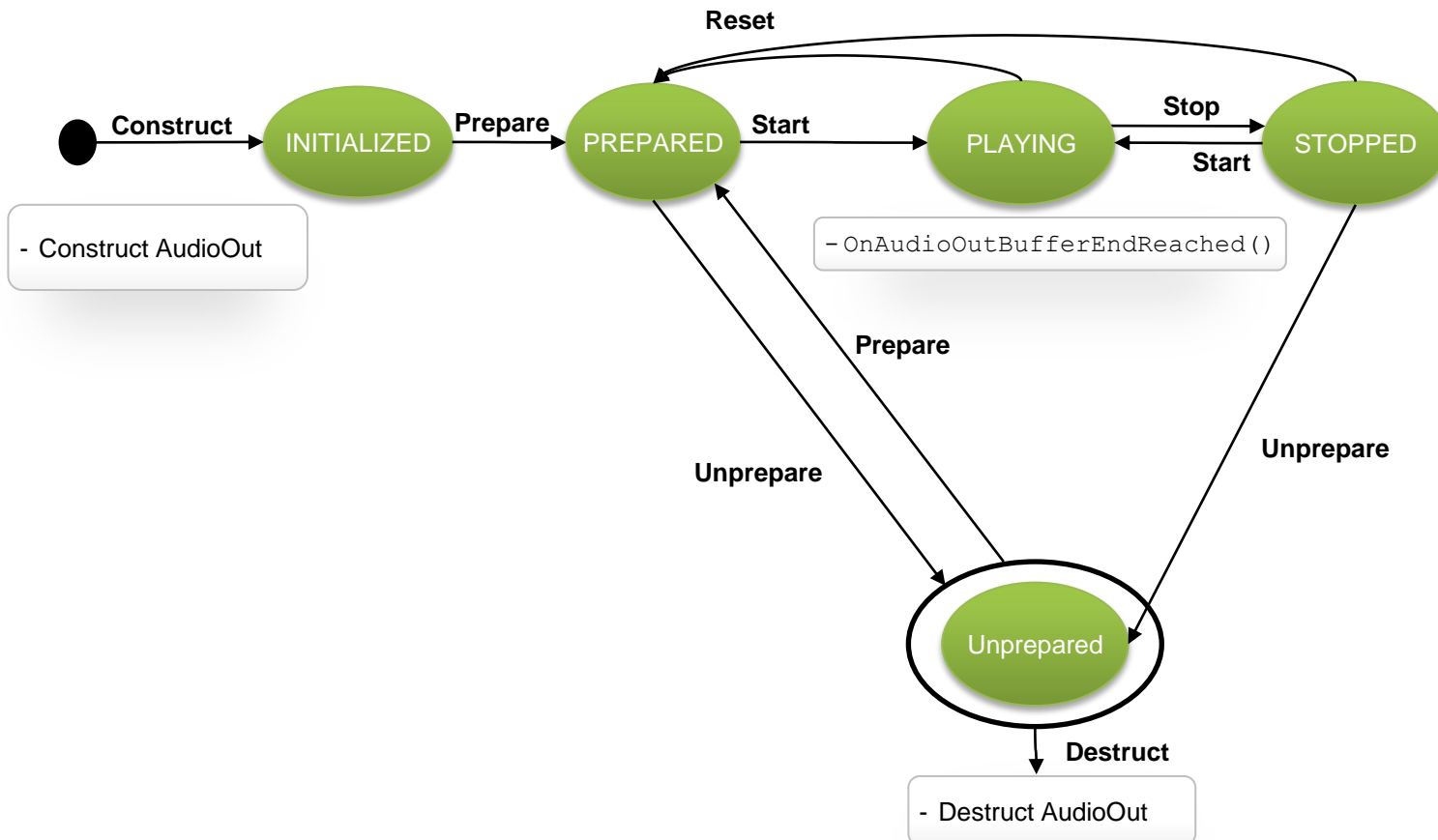
AudioOut (1/3)

- `AudioOut` provides a mechanism for playing raw uncompressed PCM data.
- The application is responsible for implementing `IAudioOutEventListener` to receive notifications through a user buffer.
- You must define PCM data settings before playing:
 - Audio channels
 - Mono (1 channel)
 - Stereo (2 channels)
 - Audio sample type
 - Unsigned 8-bit PCM
 - Signed little-endian 16-bit PCM
 - Audio sample rate
 - 8000 Hz ~ 48000 Hz



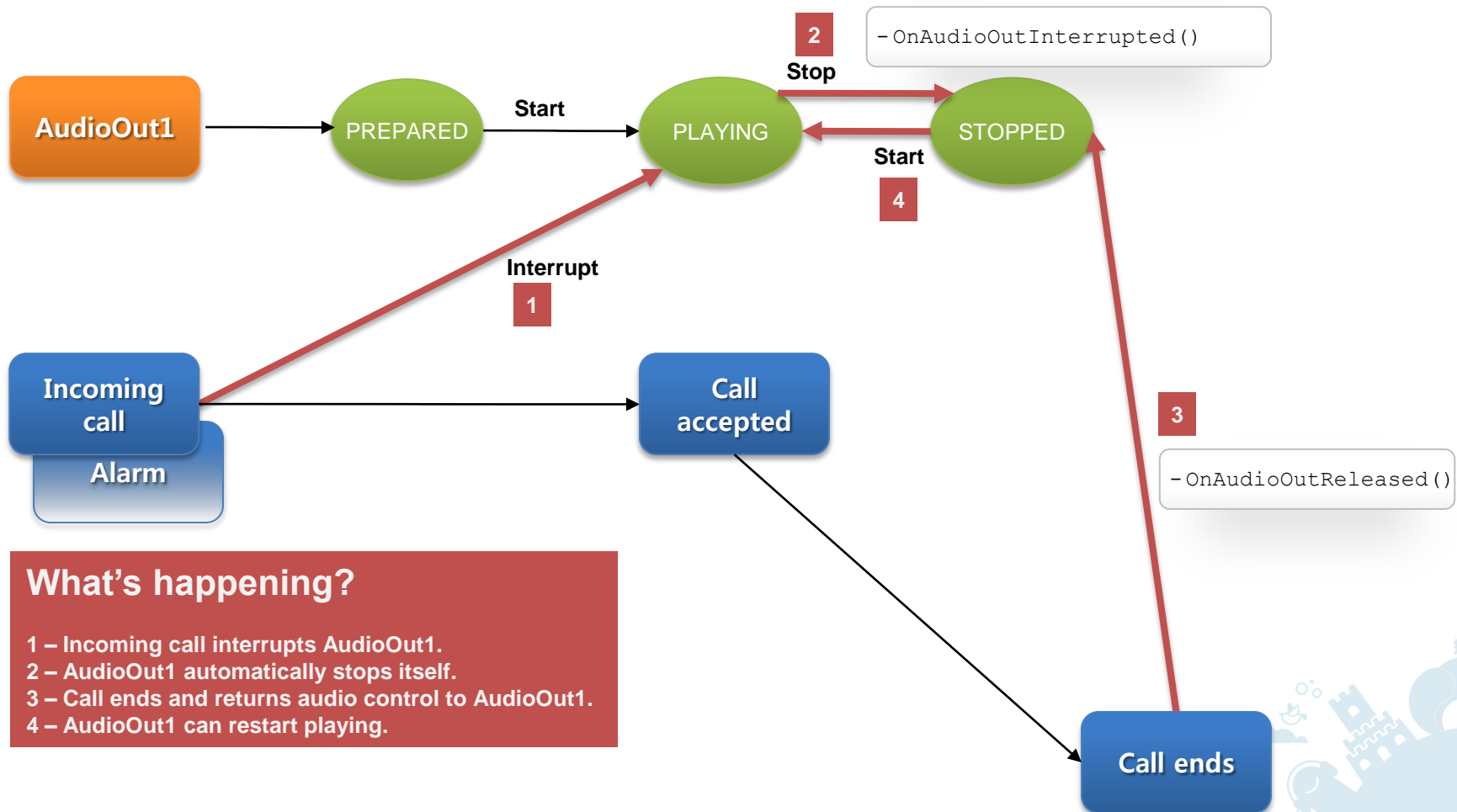
AudioOut (2/3)

AudioOut states:



AudioOut (3/3)

AudioOut states when interrupted by a system player:



What's happening?

- 1 – Incoming call interrupts AudioOut1.
- 2 – AudioOut1 automatically stops itself.
- 3 – Call ends and returns audio control to AudioOut1.
- 4 – AudioOut1 can restart playing.

Example: Play Audio (1/2)

Prepare a PCM data array and a listener, and create an `AudioOut`.

- Open `\<BADA_SDK_HOME>\Examples\MediaContent\Media\src\AudioOutExample.cpp`

1. Prepare PCM data in an array list:

```
__pDataArray;
```

2. Create a listener:

```
__pMyAudioOutListener = new AudioOutExampleListener;
```

3. Create an `AudioOut`:

```
__pAudioOutInstance = new AudioOut();
```

4. Construct an `AudioOut` instance with a listener:

```
__pAudioOutInstance->Construct(*__pMyAudioOutListener);
```

5. Prepare an `AudioOut` instance:

```
__pAudioOutInstance->Prepare(AUDIO_TYPE_PCM_U8,  
AUDIO_CHANNEL_TYPE_STEREO, 8000);
```

Example: Play Audio (2/2)

Write buffers and start playback.

- Open `\<BADA_SDK_HOME>\Examples\MediaContent\Media\src\AudioOutExample.cpp`

6. Write several buffers in a queue of the device:

```
pWriteBuffer = static_cast<ByteBuffer*>  
(__pDataArray->GetAt(__playCount++));  
__pAudioOutInstance->WriteBuffer(*pWriteBuffer);  
pWriteBuffer = static_cast<ByteBuffer*>  
(__pDataArray->GetAt(__playCount++));  
__pAudioOutInstance->WriteBuffer(*pWriteBuffer);  
pWriteBuffer = static_cast<ByteBuffer*>  
(__pDataArray->GetAt(__playCount++));  
__pAudioOutInstance->WriteBuffer(*pWriteBuffer);
```

7. Start playing until the end of the array:

```
__pAudioOutInstance->Start();
```



Camera (1/12)

- The application can preview images in real time:
 - Preview in RGB or YCbCr pixel formats.
 - Previews can be done at the frame rate.
- The application can capture image stills, and then save as JPEG or RGB565 bitmap formats.
- The application can control camera settings for contrast, effects, ISO, white balance, zoom, flash, and focus.
- The application has independent control of primary and secondary camera sensors.
- Camera previews are drawn on the background buffer that is embedded in the `Ui::Controls::OverlayPanel`.



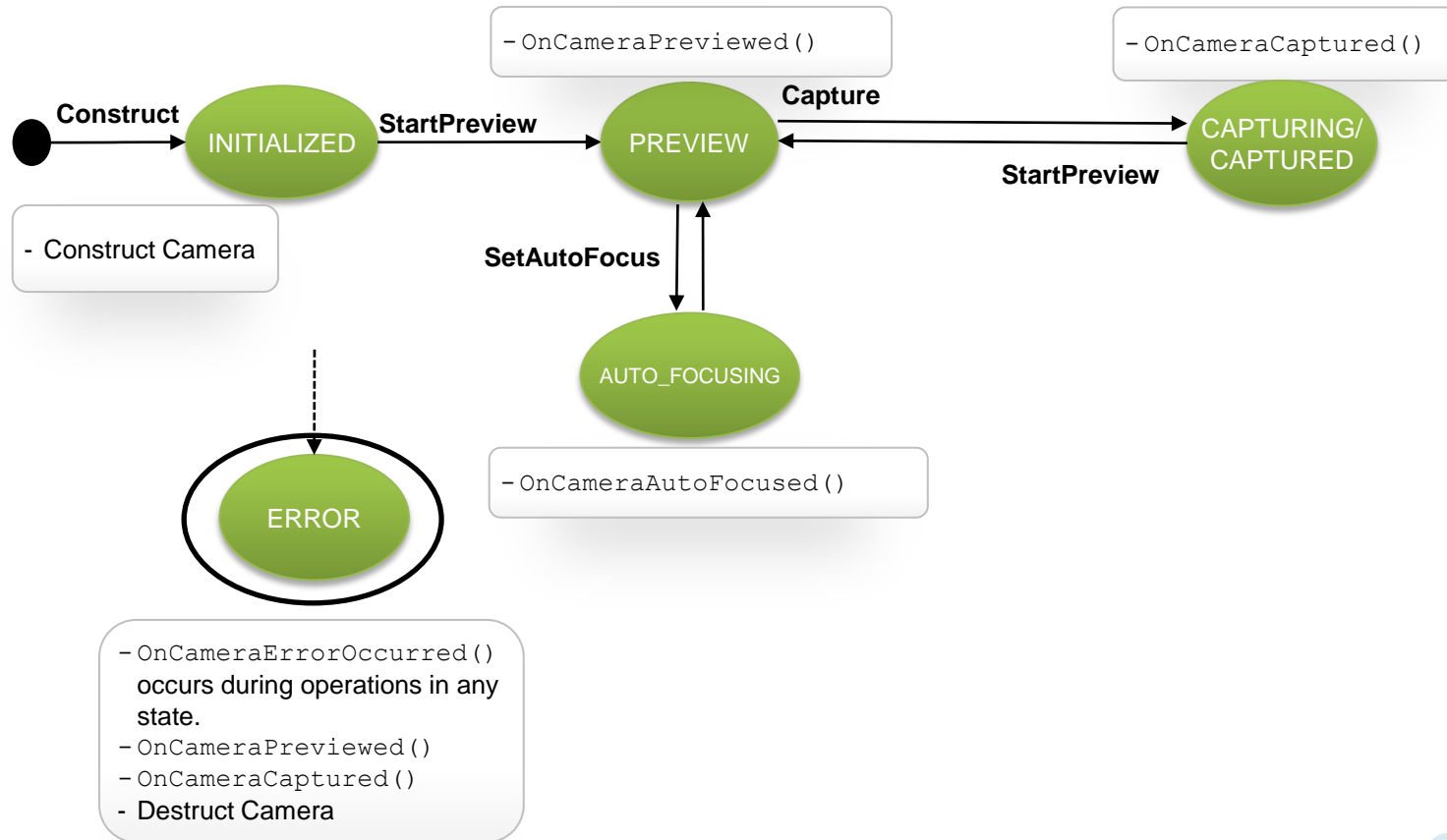
Camera (2/12)

- Supported preview pixel formats:
 - RGB565
 - YCbCr
- Supported pixel formats are device-dependent. You can obtain this information from the device using the camera's `GetSupportedPreviewFormatListN()` method.
- When developing a camera application in the Simulator environment with a webcam, you must rotate the webcam manually. The webcam rotation must correspond to the physical orientation of the camera module on the target device.



Camera (3/12)

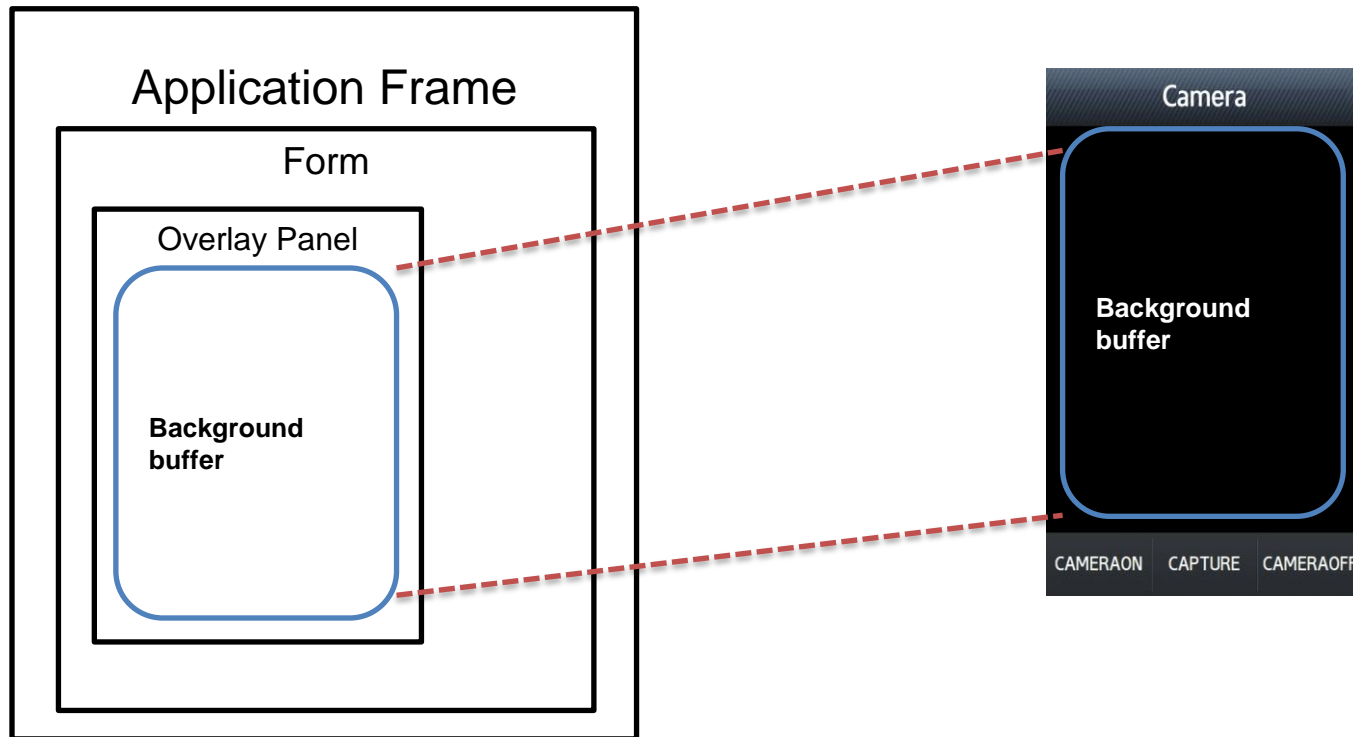
Camera states:



The `OnCameraPreviewed()` event handler is only called if specified in the `StartPreview()` method, for example: `StartPreview(canvas, true)`.

Camera (4/12)

To display previews, use the `Ui::Controls::OverlayPanel::GetBackgroundBufferInfo()` method to place a hardware-accelerated buffer inside of an overlay panel.



Camera (5/12)

To display a camera preview:

1. Create a form with `Ui::Controls::Form`.
2. Add the form to the application frame with the `AddControl()` method.
3. Set the orientation:
`Form::SetOrientation(ORIENTATION_LANDSCAPE)`
4. Create an overlay panel using `Ui::Controls::OverlayPanel`.
5. Add the overlay panel to your form with its `AddControl()` method.
6. Set the current form using `Ui::Controls::Frame::SetCurrentForm`.
7. Draw and Show the frame is essential for displaying the camera preview.
`Ui::Control::Draw()`, `Ui::Control::Show()`
8. Get the background buffer information through the overlay panel's `GetBackgroundBufferInfo()` method. (The buffer information object is a `Graphics::BufferInfo` object.)
9. Use the buffer information through the camera object's `StartPreview()` method:
`Media::Camera::StartPreview(bufferInfo)`

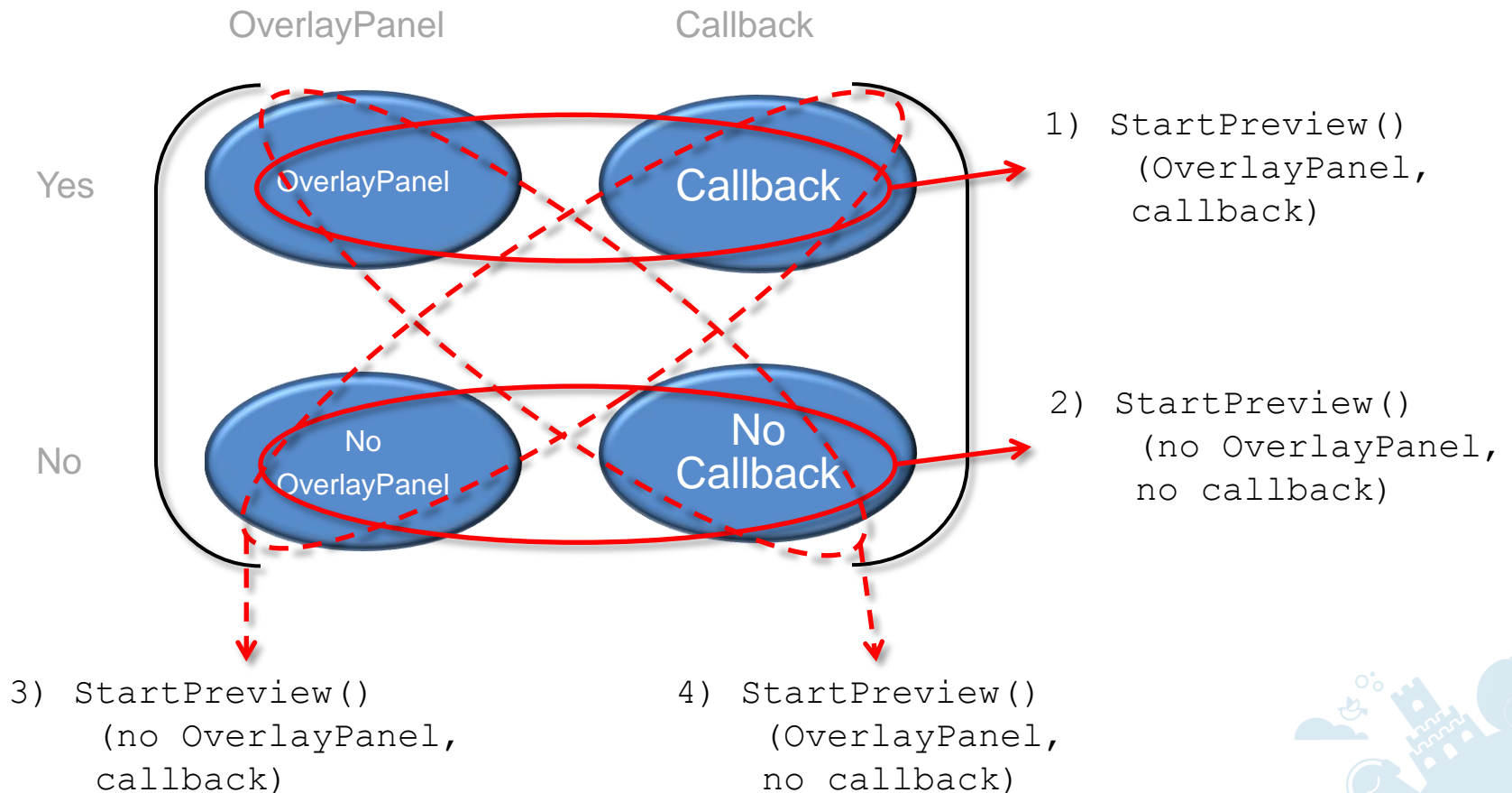
Camera (6/12)

- Supported preview resolutions:
 - Camera preview resolutions are device-dependent.
 - Device preview resolutions are available from the device through the `GetSupportedPreviewResolutionListN()` method.
- Supported preview formats:
 - Camera's preview formats are device-dependent.
 - Device preview formats are available from the device through the `GetSupportedPreviewFormatListN()` method.



Camera (7/12)

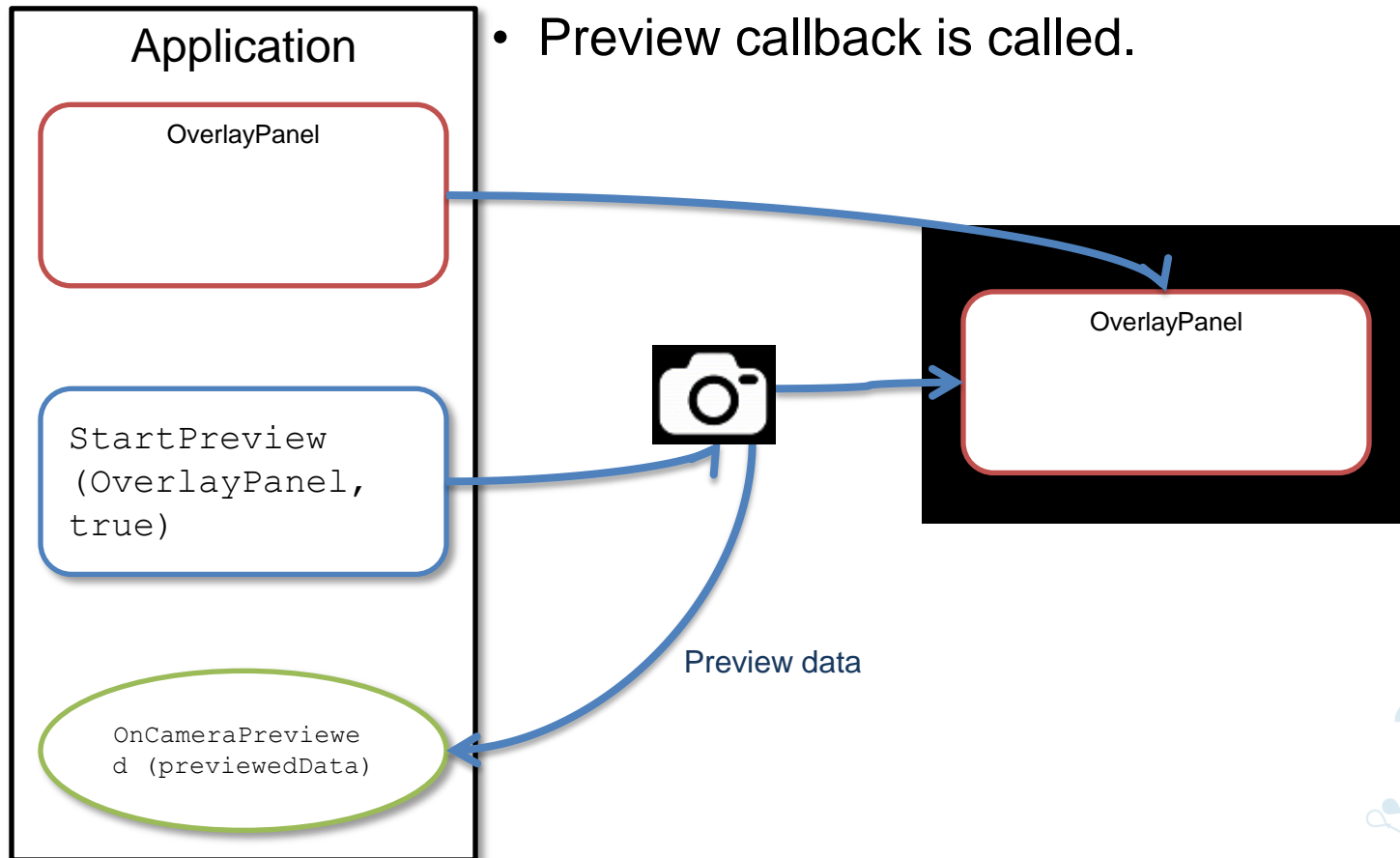
There are 4 cases for the `StartPreview()` method signature:



Camera (8/12)

OverlayPanel with callback

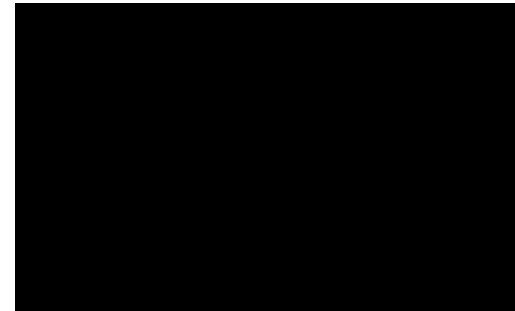
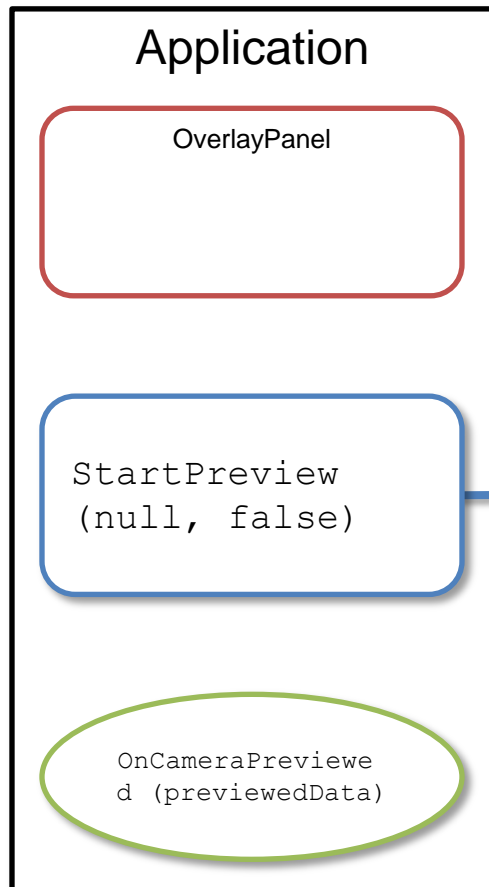
- Camera is working.
- Previews are displayed automatically.
- Preview callback is called.



Camera (9/12)

No `OverlayPanel` without callback

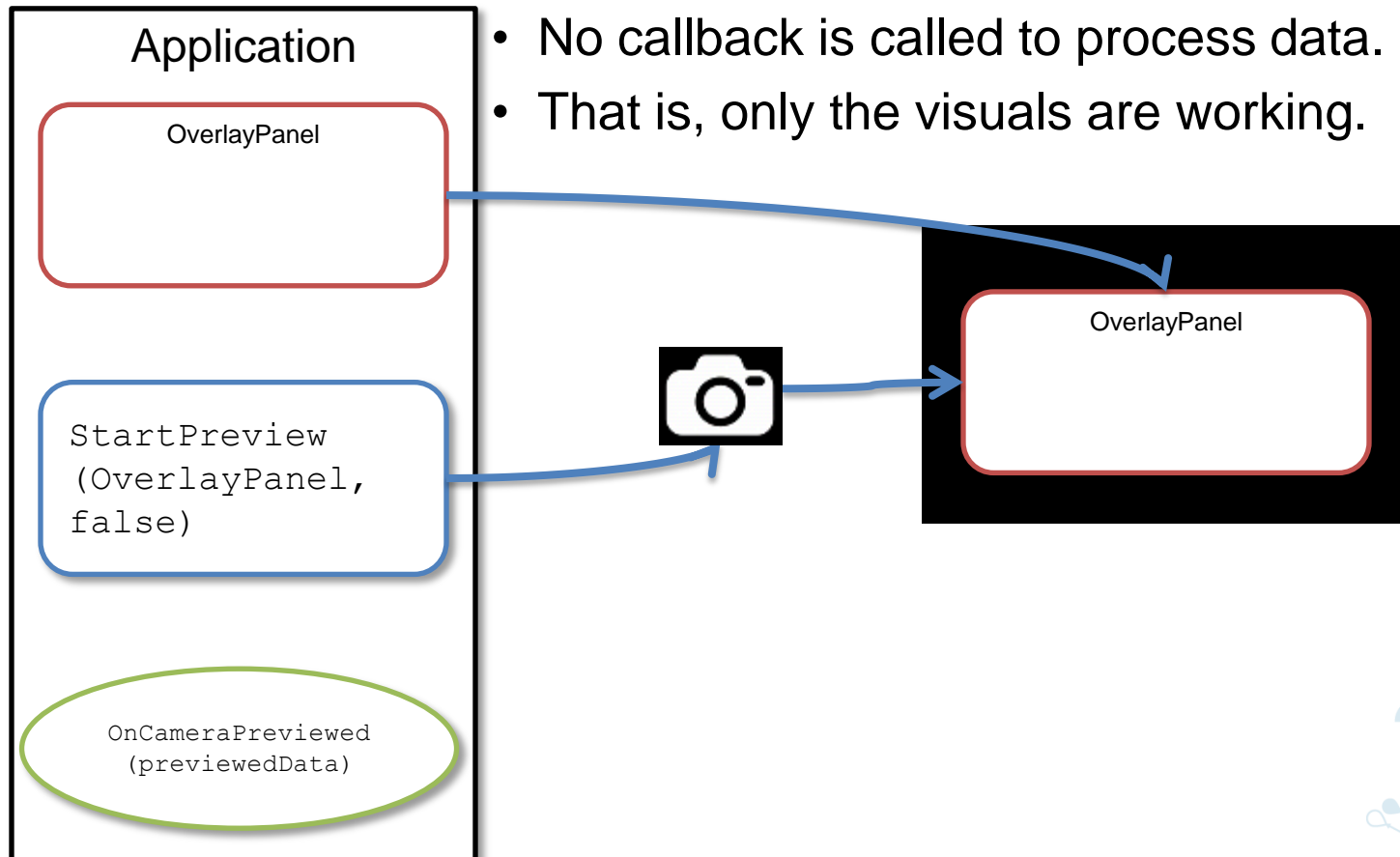
- Camera is working.
- Previews are not displayed.
- No callback is called to process data.
- The application can use this case as a power-saving mode to take pictures.



Camera (10/12)

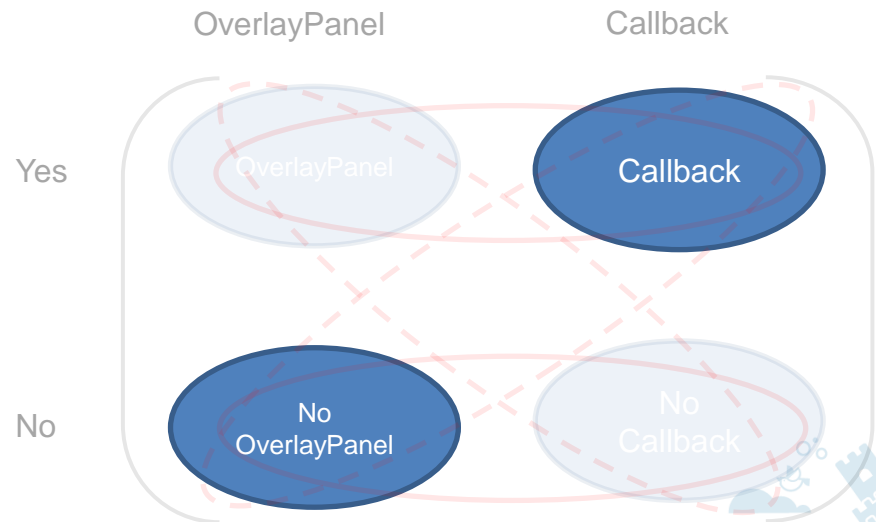
OverlayPanel without callback

- Camera is working.
- Previews are displayed automatically.
- No callback is called to process data.
- That is, only the visuals are working.



Camera (11/12)

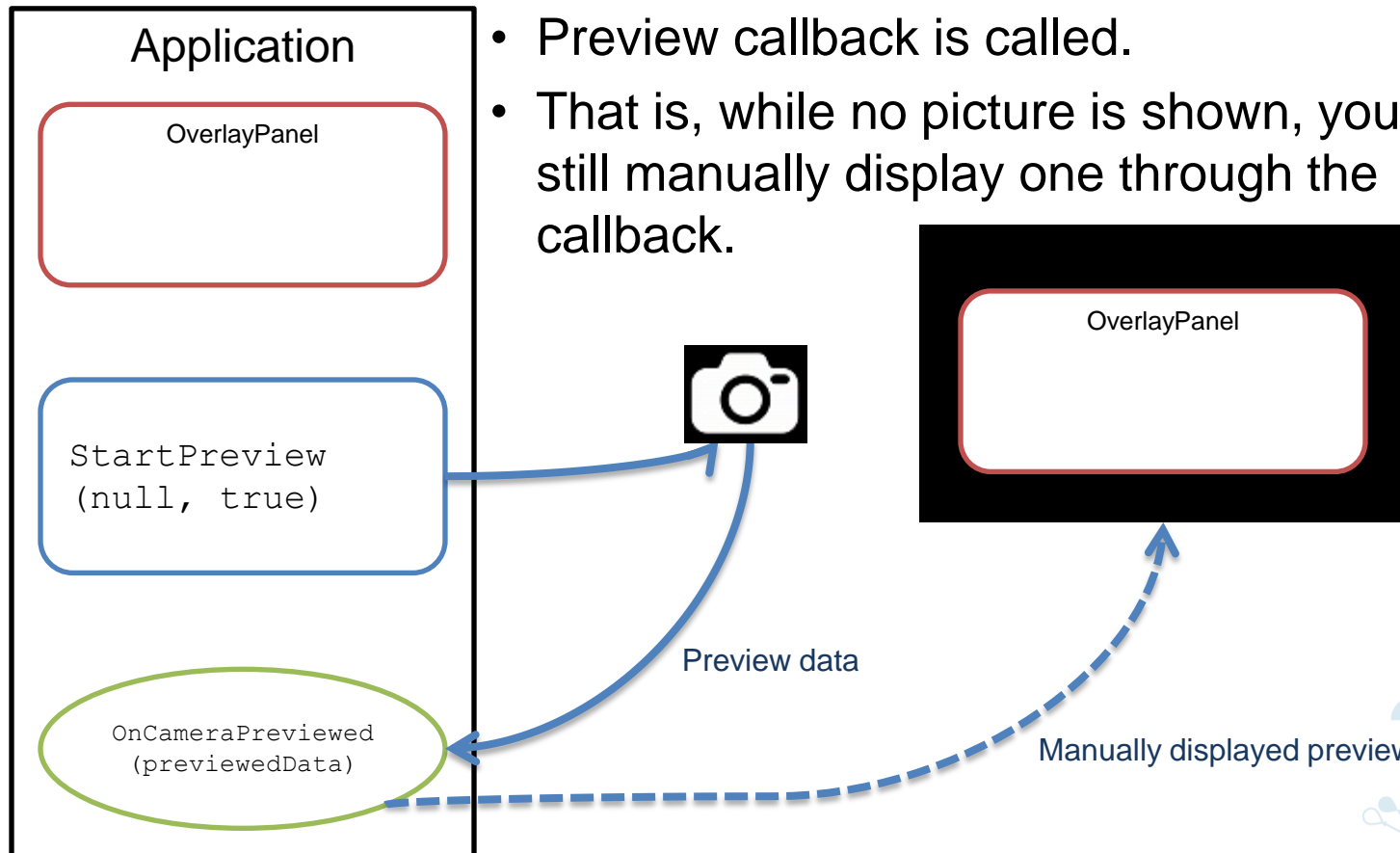
- When the application does not pass an `OverlayPanel` to the `StartPreview()` method, it can still draw on the `OverlayPanel` manually through the `OnCameraPreviewed()` event handler.
- Get the buffer information through `Ui::Controls::OverlayPanel::GetBackgroundBufferInfo()`.
- The application must set the callback parameter to true so that the preview data is sent to the `OnCameraPreviewed()` event handler.



Camera (12/12)

Callback and no OverlayPanel

- Camera is working.
- No preview is displayed.
- Preview callback is called.
- That is, while no picture is shown, you can still manually display one through the callback.



Example: Preview and Take a Picture (1/4)

Wire-up a listener, camera object, overlay panel, canvas, and start a real-time image preview through the camera.

- Open `<BADA_SDK_HOME>\Examples\MediaContent\Media\src\CameraExample.cpp`

1. Create a class and implement an `ICameraEventListener` in it.

2. Create the listener.

3. Construct a camera using the listener and a camera sensor:
`Camera::Construct(listener, cameraSelection)`

4. Switch the camera on:
`Camera::PowerOn()`

5. Set camera options (if not set, default, device-dependent values are used):

```
Camera::Set<PreviewFormat | PreviewResolution |  
IsoLevel | Brightness | Effect | Contrast | Exposure  
| WhiteBalance>()
```

Example: Preview and Take a Picture (2/4)

6. Create an overlay panel:

```
Ui::Controls::OverlayPanel::Construct()
```

7. Add the overlay panel to a parent container:

```
Ui::Container::AddControl(overlayPanel)
```

8. Set the current form:

```
Ui::Controls::Frame::SetCurrentForm(form)
```

9. Show the current form:

```
Ui::Controls::Frame::Draw()
```

```
Ui::Controls::Frame::Show()
```

10. Get a background buffer information through the `OverlayPanel's GetBackgroundBufferInfo()` method:

```
Ui::Controls::OverlayPanel::GetBackgroundBufferInfo()
```

11. Start the preview:

```
Camera::StartPreview(bufferInfo, false)
```

12. If the second parameter is "true", the camera preview data is accessible through the `OnCameraPreviewed()` callback with a frequency equal to the camera's frame rate:

```
ICameraEventListener::OnCameraPreviewed(previewedData)
```

Example: Preview and Take a Picture (3/4)

- Set the options for the picture, take a picture, save it, and clean up.
 - Open `\<BADA_SDK_HOME>\Examples\MediaContent\Media\src\CameraExample.cpp`

13. Set camera picture options:

```
Camera::SetCaptureFormat()
```

```
Camera::SetCaptureResolution()
```

```
Camera::SetQuality()
```

14. Take a picture with the camera's Capture method:

```
Camera::Capture()
```

15. Get the picture through the OnCameraCaptured() event handler:

```
ICameraEventListener::OnCameraCaptured(capturedImage)
```

16. Save the picture:

- If it is in JPEG format:

```
Io::File::Write()
```

- If it is a bitmap object:

```
Media::Image::EncodeToFile()
```



Example: Preview and Take a Picture (4/4)

17. Start the preview:

```
Camera::StartPreview(bufferInfo, false)
```

18. Stop the preview:

```
Camera::StopPreview()
```

19. Switch off the camera:

```
Camera::PowerOff()
```



Sample Application: "CameraCapture"

- Open the project in `\<BADA_SDK_HOME>\Samples\CameraCapture\src`.
- `CameraCapture` shows how you can add camera functionality to your software by getting a real-time preview, taking pictures, and saving them as `*.jpg` files.

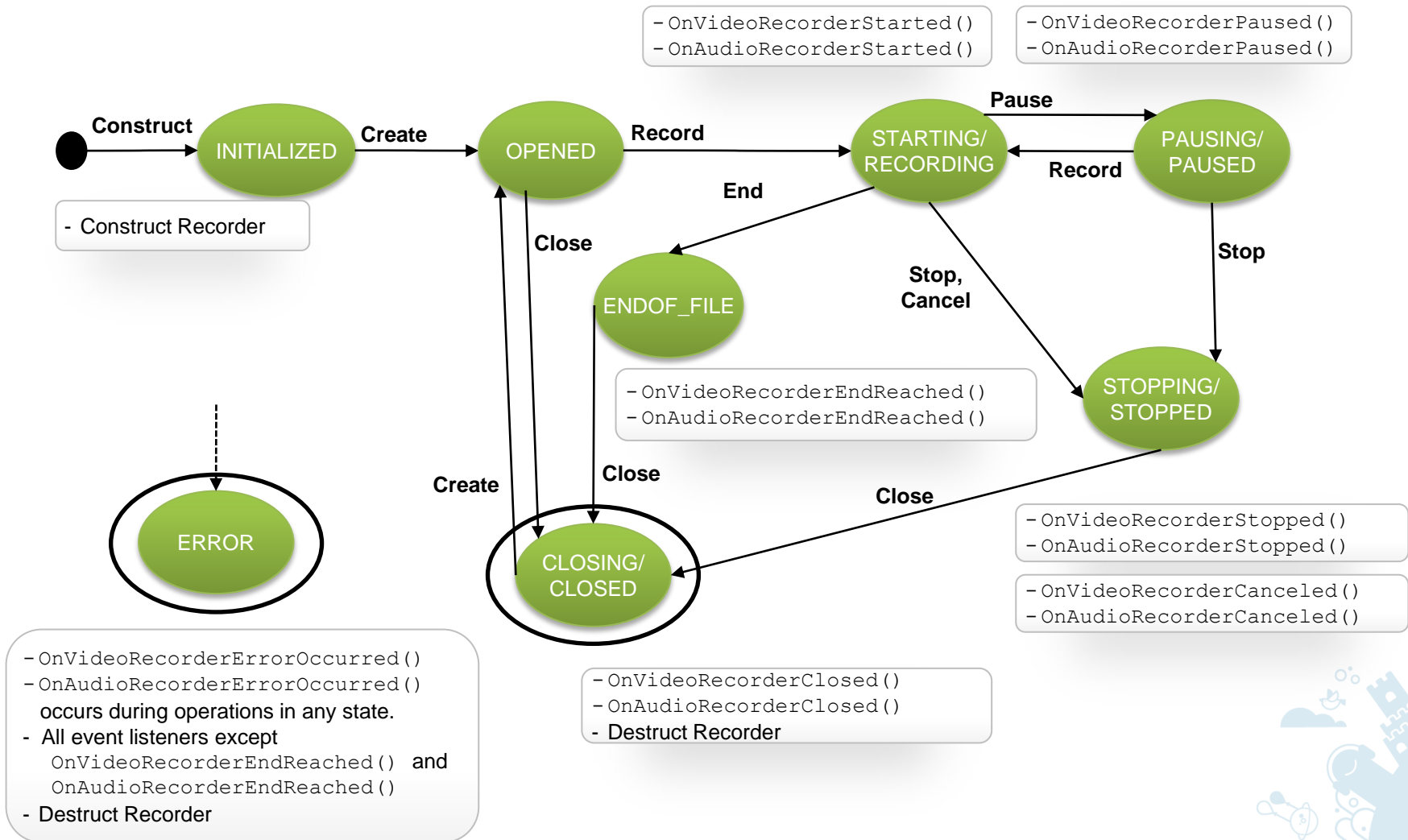


Recorder (1/2)

- Input:
 - Audio input sources depend on internal or connected microphones.
 - Video input sources are the device's internal camera sensors. External video inputs are not supported.
 - `AMedia::Camera` object serves as the interface with the hardware. Input is processed through that object.
- Recording controls include Record, Stop, Pause, and Mute.
- Supported formats:
 - `VideoRecorder`: MP4, 3GP
 - `AudioRecorder`: AMR, WAV
- Video resolutions are device-dependent.
- Only the device's internal cameras can be used as a video input source. External video inputs are not supported.
- The resolutions of the video recorder and the preview on the camera must be equal.

Recorder (2/2)

Recorder states are similar to player states:



Example: Record a Video (1/2)

Wire-up a listener, camera, start a preview, and set your video recording options.

- Open `<BADA_SDK_HOME>\Examples\MediaContent\Media\src\VideoRecorderExample.cpp`

1. Create a class and implement an `IVideoRecorderEventListener` in it.
2. Create the listener.
3. Construct a camera and start the preview. (Refer to “Example: Preview and Take a Picture ”.)
4. Construct a `VideoRecorder`:
`VideoRecorder::Construct(listener, camera)`
5. Create a file to save the video stream:
`VideoRecorder::CreateVideoFile()`
6. Set your video options:
`VideoRecorder::Set<Codec | Format | Mode | Quality | RecordingResolution* | MaxRecordingSize | MaxRecordingTime>()`

* *The resolution of the recorder must be the same as the resolution of the camera.*

Example: Record a Video (2/2)

- Start recording, stop recording, then save the video and clean up.
 - Open `<BADA_SDK_HOME>\Examples\MediaContent\Media\src\VideoRecorderExample.cpp`
- 7. Start recording:
`VideoRecorder::Record()`
`IVideoRecordingEventListener::OnVideoRecorderStarted()`
- 8. Stop recording and close the file in the `OnVideoRecorderStopped()` event handler:
`VideoRecorder::Stop()`
`IVideoRecorderEventListener::OnVideoRecorderStopped()`
- 9. Close the video recorder:
`VideoRecorder::Close()`
- An additional step is needed if the maximum time or file size is reached. In this case, you must close the file in the `OnVideoRecorderEndReached()` event handler:
`IVideoRecorderEventListener::OnVideoRecorderEndReached()`.

Sample Application: "VoiceRecorder"

- Open the project in `\<BADA_SDK_HOME>\Samples\VoiceRecorder\src`.
- `VoiceRecorder` shows how to:
 - Start and stop audio recording.
 - Set the maximum recording time and size.
 - Set the recording quality (low, medium, or high).
- Defaults:
 - Maximum recording time = 60 seconds.
 - Maximum recording size = 5 MB.
 - Quality = high.
- The absolute maximum recording time and size is device-dependent.
- The recorded data is saved as an WAVE file.



Sample Application: “CameraCapture” for Video Recorder

- Open the project in `\<BADA_SDK_HOME>\Samples\CameraCapture\src`.
- CameraCapture shows how to:
 - Control the camera as a source.
 - Start and stop video recording.
 - Set the recording resolution.
 - Check the recording time and size.
- Defaults:
 - Maximum recording time = 120 seconds.
 - Maximum recording size = 5 MB.
 - Quality = default.
- The absolute maximum recording time and size is device-dependent.
- The recorded data is saved as a MP4 file.



DRM Information

- Rights management is handled automatically for licensed and valid DRM content.
- The DRM information class lets the application get DRM information from DRM-protected files, such as DRM Info-type, DRM Info-Method, Right Status, Permission Type and Right Constraints.

DRM info - type	DRM info - method
OMA DRM v1	Forward lock
OMA DRM v2	Combined delivery
OMA DRM v2.1	Separate delivery
WMDRM	Super distribution

Right status	Right constraints
Valid	Count
Future use	Datetime
Expired	Interval
Pending	Time-count
No rights	Accumulated

Permission type
Any
Play
Display
Execute
Print



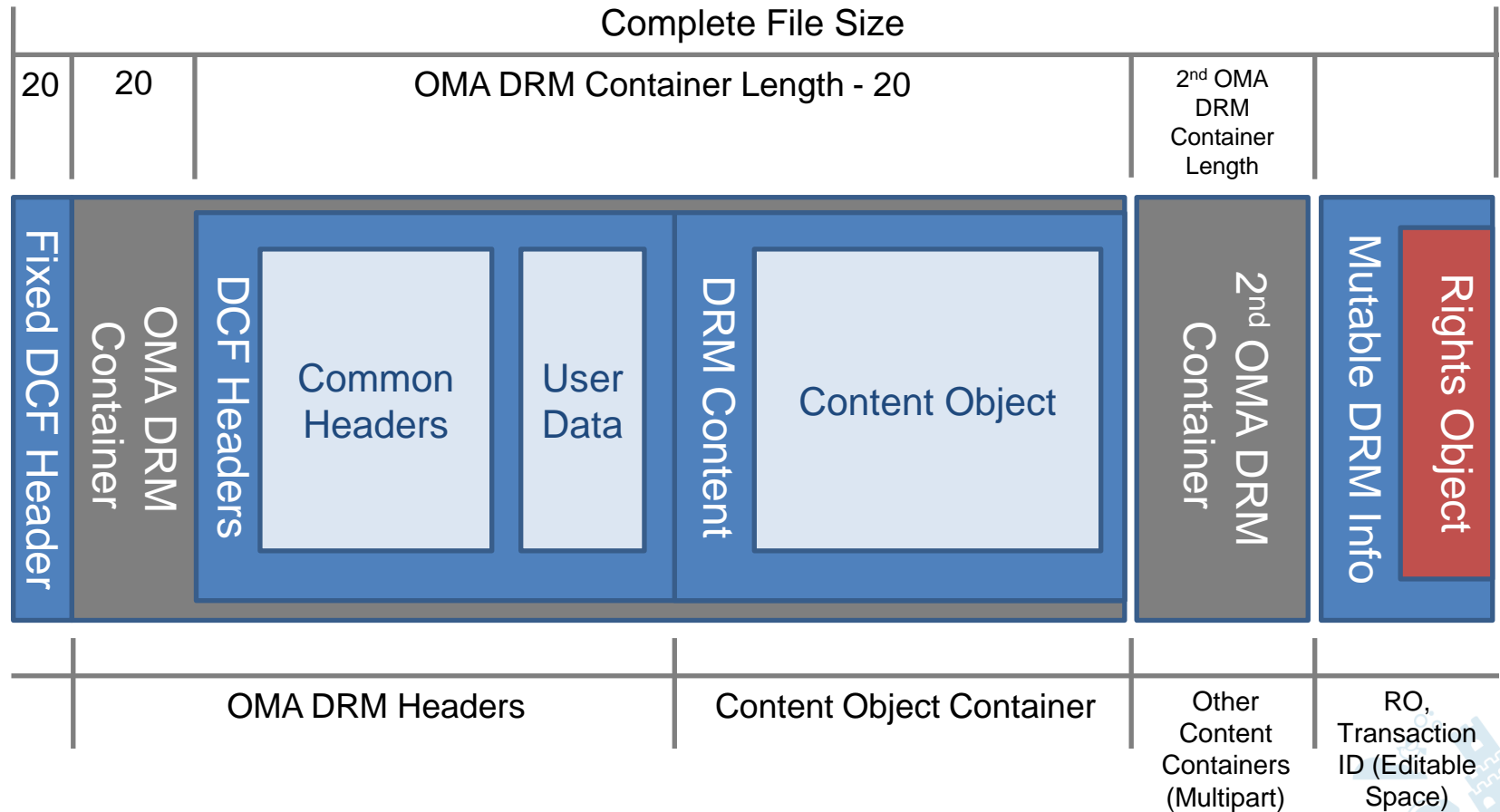
Right ConstraintInfo

- The `ConstraintInfo` class contains the current 'Right Constraint' information for DRM-protected content.
- The value of the `ConstraintInfo` is updated after the DRM rights object has been consumed by a player or a viewer.

	Property	Type	Description
ConstraintInfo	StartTime/EndTime	DateTime	The starting and ending date and time for which a permission can be granted over an asset.
	Interval	DateTime	A period of time during which the permissions can be exercised over the DRM content.
	OriginalCounter	Int	The total number of times a permission can be granted over an asset.
	LeftCounter	Int	The remaining number of times a permission can be granted over an asset.
	TimedCount	Int	A count constraint with the addition of an optional timer attribute.
	AccumulatedTime	Int	The maximum period of metered usage-time during which the rights can be exercised over the DRM content.

OMA DRM Format

The OMA DRM format is as illustrated below.



* Illustration adapted from "DRM Content Format: Approved Version 2.0 – 03 Mar 2006", Open Mobile Alliance OMA-TS-DCF-V2_0-20060303-A.

Example: Get DRM Information

Get information from a DRM-protected file.

- Open `\<BADA_SDK_HOME>\Examples\MediaContent\Media\src\DrminoExample.cpp`

1. **Construct a `Drmino` object:**

```
Drmino::Construct()
```

2. **Get the content title of the DRM file:**

```
Drmino::GetContentTitle()
```

3. **Get the content type of the DRM file:**

```
Drmino::GetContentType()
```

4. **Get the DRM constraints of the DRM file:**

```
Drmino::GetConstraintInfo()
```

5. **Get the number of times the DRM-protected file can be played back:**

```
DrminoConstraintInfo::GetLeftCounter()
```



FAQ

Where do I store media files during development, such as images to test a media program from the IDE?

- If media files are private content for the application, store the media files in the project's `Res` folder, which is copied to the Simulator file system (`/Home/Res`).



Review

1. Can you use an external microphone to record audio?
2. Can you use an external video camera to record video?
3. Can you use an external camera to take pictures?
4. Can you record 24/96 audio with bada?
5. Can you record HD video?
6. Can you loop audio or video?
7. You want to apply some funky special effects to a picture preview. What do you pass to the `StartPreview()` method?



Answers

1. Yes. You can use a device's internal MIC or an external one.
2. No. You must use the device's internal video capabilities.
3. No.
4. It all depends. Bit depth and frequency are device-dependent, so you can do it only if the device supports it. However, no mobile devices support 24/96 audio, so the practical answer is no.
5. As with video, resolutions are all device-dependent. So the practical answer is no, you cannot record HD video.
6. You can loop audio, but not video.
7. Pass null for the `OverlayPanel` parameter, but pass a valid callback. You can then access the picture data in the callback, do your funky effects, and draw the new picture on the `OverlayPanel`.





bada

The platform with more opportunities
Invitation to Adventure