Audio recording and playback options in Windows Phone

This article provides an overview and suggestions on the usage of the different Windows Phone 8 audio handling APIs available to a third party developer.

01 Sep 2013

Introduction

Windows Phone 8 provides many different options for capturing and playing audio. This article aims to give an overview of these options and acts as a portal through which the reader can access information and essential resources related to the audio handling in Windows Phone 8.

In particular, this article explores the options available for 3rd party developers to utilize the superb audio recording quality provided by the Nokia Lumia Windows Phone 8 devices, and was inspired by the top notch microphones packed on most of the Nokia Lumia Windows Phone 8 devices, such as Nokia Lumia 920 with as many as 3 High Dynamic Range Microphones, and Nokia Lumia 820 / 720 / 620 with 2 of the same species. For more information see the Nokia Rich Recording Technology - whitepaper.

Most of the code snippets shown are for WASAPI (Windows Audio Session API) because it provides better audio than the other approaches but (at least until now) was not as well documented on Windows Phone as the alternatives.

The source code for the AudioRecorder example application used to test and experiment with the audio APIs is available at Nokia Developer GitHub.

API comparison tables

Options for capturing audio:

Class	API / Namespace	Sample rate	Sample size	Channels	Audio encoding
Microphone	Microsoft.Xna.Framework.Audio	16000 Hz	16 bits	1	PCM
IAudioClient2 and IAudioCaptureClient	WASAPI	adjustable	adjustable	adjustable (1)	PCM
AudioVideoCaptureDevice	Windows.Phone.Media.Capture	-	-	-	AAC / AMR (2)

Options for rendering audio:

Class	API / Namespace	Sample rate	Sample size	Channels	Audio encoding
SoundEffectInstance and SoundEffect	Microsoft.Xna.Framework.Audio	8000-48000 Hz	16 bits	1/2	PCM
DynamicSoundEffectInstance	Microsoft.Xna.Framework.Audio	8000-48000 Hz	16 bits	1/2	PCM
IAudioClient2 and IAudioRenderClient	WASAPI	adjustable	adjustable	adjustable (3)	PCM
IXAudio2 and IXAudio2VoiceCallback	XAudio2	adjustable	adjustable	adjustable	PCM
MediaElement	System.Windows.Controls	from source data	from source data	from source data	WAV/AAC/etc.

(1) It seems that by default WASAPI captures audio in mono. It seems possible to override the default using certain flags however. See WASAPI section for more information.

(2) Capturing PCM encoded audio is not possible even though it is mentioned in CameraCaptureAudioFormat enumeration documentation.

(3) It seems that by default WASAPI expects to render stereo data. It seems possible to override the default using certain flags however. See WASAPI section for more information.

Empirical findings and recommendations

The difference can be heard when listening to a recording made with AudioRecorder example application (16000 = XNA vs. 44100 = WASAPI). Simple speech is a good candidate for testing. The sound in higher sample rate is somewhat sharper, clearer, while the lower sample rate recording sounds like having some static (white noise) in the background. This is only natural, when the other has almost three times the samples per time unit. Supposedly the difference in audio quality comes only from the difference in sample rate. Both had a sample size of 16-bits.

Concerning, AudioVideoCaptureDevice, while PCM isn't available, and the sound is AAC/AMR encoded and thus possibly packed or processed in some way, it does loose some info.

WASAPI is the way to go, if you need the highest possible quality available and don't want to be stuck in 16000 Hz 16-bit Mono you get from XNA Microphone class or AAC/AMR encoded data from AudioVideoCaptureDevice. Also while with the other two APIs you can't set the audio quality properties, with WASAPI you can.

If you don't need to push for the best possible quality, you are probably better with the other audio APIs.

XNA APIs

There are a number of examples on how to capture and render audio using the classes in Microsoft.Xna.Framework.Audio namespace and thus they are not covered here in detail. See the following documentation and examples for more detailed information on XNA audio handling:

- How to access the microphone in a Windows Phone app
- Microphone Sample
- How to access and manage the Microphone raw data in WP
- Sound Recording in Windows Phone 7, a great article offering insight on any kind of audio application design and implementation.
- A Nokia Developer example application AudioRecorder can also be used as a starting point in learning to use XNA Audio APIs. It features capturing and rendering audio using Microphone and DynamicSoundEffectInstance.

Some things to consider when using XNA Audio APIs are listed below:

- XNA audio services are available also in Windows Phone 7 devices.
- XNA audio services require that FrameworkDispatcher.Update method to be called in order to dispatch messages that are in the XNA Framework message queue. See MSDN documentation for details.
- Microphone class captures PCM encoded audio in constant 16000 Hz - 16-bit - mono.
- SoundEffectInstance and SoundEffect classes that can be used to render audio, require a stream object that points to the head of a valid PCM wave file, which must be in the RIFF bitstream format (see SoundEffect.FromStream for details).
- DynamicSoundEffectInstance can be used to render raw PCM data, even dynamically created data, rather than just pre-recorded WAV files. DynamicSoundEffectInstance expects data to have a 16-bit sample size, but sample rate as well as channel count (mono-stereo) can be set.

WASAPI (Windows Audio Session API)

Together with XAudio2 the WASAPI (Windows Audio Session API) form the Native audio APIs for Windows Phone 8. Windows Phone 8 supports a subset of the WASAPI interfaces, as described in Audio Capture and Render APIs for native code for Windows Phone. For a managed C# code to access WASAPI functionality, a Windows Phone Runtime Component needs to be implemented to interact with WASAPI and expose the needed methods and events to the managed code. It seems to be hard to find any well documented examples on using WASAPI to capture and render audio. Therefore an example in a form of code extract demonstrating how to do those tasks is given later on in this article.

For extensive WASAPI documentation see About WASAPI in MSDN.

- WASAPI can be used for both capturing and rendering audio, both of which can be accomplished using event-based techniques as well as using timers and procedural techniques.
- The ChatterBox VoIP sample app is one of the rare code examples offering hands-on implementation example on WASAPI usage on Windows Phone 8. This example also shows how to use event based approach to reading and writing audio data from/to WASAPI.
- The Windows Audio Session (WASAPI) sample demonstrates the use of XAudio2 in a Windows 8.1 app, but it can also offer guidance for using WASAPI in Windows Phone 8.
- A Nokia Developer example application AudioRecorder can also be used as a starting point in learning to use WASAPI. It

features a Windows Phone Runtime Component used to capture and render audio and controlling the WASAPI from managed C# code. It features a timer based technique to read and write audio data to/from WASAPI. With this example application it is easy to hear the difference between lower quality 16000 Hz audio data captured by XNA Microphone and higher quality 44100 Hz audio data captured using WASAPI.

Other available audio APIs

AudioVideoCaptureDevice

AudioVideoCaptureDevice class offers possibility to record AAC and AMR encoded audio data. Capturing PCM encoded audio is not possible even though it is mentioned in CameraCaptureAudioFormat enumeration documentation as explained in Recording PCM on Windows Phone 8 using AudioVideoCaptureDevice.

- It seems to be hard to find any well documented example on using `AudioVideoCaptureDevice` to record audio. Therefore a code extract demonstrating how to do that is given later on in this article.
- Capturing AAC or AMR encoded audio with `AudioVideoCaptureDevice` works well together with `MediaElement`, which is able to play AAC and AMR (among others) encoded audio files.
- Possibility to record straight to a stream or create a sink to handle audio samples in native code.

MediaElement

MediaElement class is an easy-to-use C# control - just define and name it in xaml-file so that code-behind, such as an event handler code, can refer to a markup element after it is constructed during processing by a XAML processor. To make the code extract mentioned in `AudioVideoCaptureDevice` complete, it also demonstrates how to play the recorded AAC encoded audio with `MediaElement`.

Few pointers here too:

- Plays audio files, WAV, AAC & AMR (the output formats of AudioVideoCaptureDevice) and others. See Supported Media Formats, Protocols, and Log Fields for details.
- Offers Play/Pause/Stop/Speed/Position/Volume functionality as demonstrated in How to: Control a MediaElement.
- An important detail is that "passing a generic stream to SetSource(System.IO.Stream) is not supported in Silverlight for Windows Phone. However, the `IsolatedStorageFileStream` class, which derives from Stream, is supported on Silverlight for Windows Phone" as stated in MediaElement.SetSource Method.
- See also Quickstart: Audio and video for Windows Phone.

XAudio2

Together with WASAPI the XAudio2 form the Native audio APIs for Windows Phone 8. For a full XAudio2 documentation see XAudio2 APIs. XAudio2 is a low-level, low-latency audio API. It is a part of DirectX APIs made available in Windows Phone 8 SDK, making it possible to share audio handling source code between Windows, Xbox and Windows Phone 8 projects. For a managed C# code to access XAudio2 functionality, a Windows Phone Runtime Component needs to be implemented to interact with XAudio2 and expose the needed methods and events to the managed code.

- There is a nice article on C# XAudio2 Sound Playback for Windows Phone, complete with source code demonstrating the use of XAudio2 in detail.
- There's also a Nokia Developer example application DrumkitX which plays drum sounds using XAudio2 API.
- The XAudio2 audio file playback sample demonstrates the use of XAudio2 in a Windows 8.1 app, but it can also offer guidance if implementing XAudio2 features in Windows Phone 8.

Code Extracts

Capturing audio using WASAPI

This code extract, taken from a Nokia Developer example application AudioRecorder shows how to use WASAPI capture and render audio. These are just extracts from the source code of AudioRecorder available at Nokia Developer GitHub. Remember that all applications wanting to capture audio need `ID_CAP_MICROPHONE` capability.

To interact with WASAPI a Windows Phone Runtime Component must be created. See Creating Windows Runtime Components in C++, Array and WriteOnlyArray (C++/CX) and Visual C++ Language Reference (C++/CX) for more details on the techniques used in AudioRecorder.

The header file of the only class in the Windows Phone Runtime Component implemented in AudioRecorder is shown below:

```
#include <windows.h>

#include <synchapi.h>
#include <audioclient.h>
#include <phoneaudioclient.h>

namespace WasapiAudioComp
{
    public ref class WasapiAudio sealed
    {
    public:
        WasapiAudio();
        virtual ~WasapiAudio();

        bool StartAudioCapture();
        bool StopAudioCapture();
        int ReadBytes(Platform::Array<byte>^* a);

        bool StartAudioRender();
        bool StopAudioRender();
        void SetAudioBytes(const Platform::Array<byte>^ a);
        bool Update();
        void SkipFiveSecs();

    private:
        HRESULT InitCapture();
        HRESULT InitRender();

        bool started;
        int m_sourceFrameSizeInBytes;

        WAVEFORMATEX* m_waveFormatEx;

        // Devices
        IAudioClient2* m_pDefaultCaptureDevice;
        IAudioClient2* m_pDefaultRenderDevice;

        // Actual capture and render objects
        IAudioCaptureClient* m_pCaptureClient;
        IAudioRenderClient* m_pRenderClient;

        BYTE* audioBytes;
        long audioIndex;
        long audioByteCount;
    };
}
```

To capture audio using WASAPI, the capture device and client must first be initialized. In the AudioRecorder the capture device is initialized to capture audio in 44100 Hz 16-bit stereo. The captured audio is not actually stereo, due to some software limitations, but having both channels helps later while rendering the same data through WASAPI render client, which seems to expect stereo data.

```
/**
 * Start audio capturing using WASAPI.
 * @return The success of the operation.
```

```
 */
bool WasapiAudio::StartAudioCapture()
{
    bool ret = false;

    if (!started)
    {
        HRESULT hr = InitCapture();
        if (SUCCEEDED(hr))
        {
            ret = started = true;
        }
    }

    return ret;
}

/**
 * Initialize WASAPI audio capture device.
 * @return The success of the operation.
 */
HRESULT WasapiAudio::InitCapture()
{
    HRESULT hr = E_FAIL;

    LPCWSTR captureId = GetDefaultAudioCaptureId(AudioDeviceRole::Default);

    if (NULL == captureId)
    {
        hr = E_FAIL;
    }
    else
    {
        hr = ActivateAudioInterface(captureId, __uuidof(IAudioClient2),
(void**)&m_pDefaultCaptureDevice);
    }

    if (SUCCEEDED(hr))
    {
        hr = m_pDefaultCaptureDevice->GetMixFormat(&m_waveFormatEx);
    }

    // Set the category through SetClientProperties
    AudioClientProperties properties = {};
    if (SUCCEEDED(hr))
    {
        properties.cbSize = sizeof AudioClientProperties;
        properties.eCategory = AudioCategory_Other;
        // Note that AudioCategory_Other is the only valid category for capture and
loopback streams.
        // From: http://msdn.microsoft.com/en-
us/library/windows/desktop/hh404178(v=vs.85).aspx
        hr = m_pDefaultCaptureDevice->SetClientProperties(&properties);
    }

    if (SUCCEEDED(hr))
    {
```

```
        WAVEFORMATEX temp;
        MyFillPcmFormat(temp, 2, 44100, 16); // stereo, 44100 Hz, 16 bit

        *m_waveFormatEx = temp;
        m_sourceFrameSizeInBytes = (m_waveFormatEx->wBitsPerSample / 8) *
m_waveFormatEx->nChannels;

        // using device to capture stereo requires the flag 0x8800000, or at least some
part of it
        hr = m_pDefaultCaptureDevice->Initialize(AUDCLNT_SHAREMODE_SHARED, 0x88000000,
1000 * 10000, 0, m_waveFormatEx, NULL);
    }

    if (SUCCEEDED(hr))
    {
        hr = m_pDefaultCaptureDevice->GetService(__uuidof(IAudioCaptureClient),
(void**)&m_pCaptureClient);
    }

    if (SUCCEEDED(hr))
    {
        hr = m_pDefaultCaptureDevice->Start();
    }

    if (captureId)
    {
        CoTaskMemFree((LPVOID)captureId);
    }

    return hr;
}
```

The AudioRecorder example uses a polling mechanism to retrieve audio data from the Windows Phone Runtime Component by frequently calling `ReadBytes` method. The returned byte array is then stored in managed code as well as used to visualize the relative loudness of the captured audio.

```
/**
 * Read accumulated audio data.
 * @param byteArray The byte array to be filled with audio data.
 * @return The number of audio bytes returned.
 */
int WasapiAudio::ReadBytes(Platform::Array<byte>^* byteArray)
{
    int ret = 0;
    if (!started) return ret;

    BYTE *tempBuffer = new BYTE[MY_MAX_RAW_BUFFER_SIZE];
    UINT32 packetSize = 0;
    HRESULT hr = S_OK;
    long accumulatedBytes = 0;

    if (tempBuffer)
    {
        hr = m_pCaptureClient->GetNextPacketSize(&packetSize);
```

```
        while (SUCCEEDED(hr) && packetSize > 0)
        {
            BYTE* packetData = nullptr;
            UINT32 frameCount = 0;
            DWORD flags = 0;
            if (SUCCEEDED(hr))
            {
                hr = m_pCaptureClient->GetBuffer(&packetData, &frameCount, &flags,
nullptr, nullptr);
                unsigned int incomingBufferSize = frameCount * m_sourceFrameSizeInBytes;

                memcpy(tempBuffer + accumulatedBytes, packetData, incomingBufferSize);
                accumulatedBytes += incomingBufferSize;
            }

            if (SUCCEEDED(hr))
            {
                hr = m_pCaptureClient->ReleaseBuffer(frameCount);
            }

            if (SUCCEEDED(hr))
            {
                hr = m_pCaptureClient->GetNextPacketSize(&packetSize);
            }
        }

        // Copy the available capture data to the array.
        auto temp = ref new Platform::Array<byte>(accumulatedBytes);
        for(long i = 0; i < accumulatedBytes; i++)
        {
                temp[i] = tempBuffer[i];
        }
        *byteArray = temp;
        ret = accumulatedBytes;

        // Reset byte counter
        accumulatedBytes = 0;
    }

    delete[] tempBuffer;

    return ret;
}
```

To stop capturing audio, the capture device is stopped and then released alongside capturing client:

```
/**
 * Stop audio capturing using WASAPI.
 * @return The success of the operation.
 */
bool WasapiAudio::StopAudioCapture()
{
    bool ret = false;

    if (started)
    {
```

```
        HRESULT hr = S_OK;

        if (m_pDefaultCaptureDevice)
        {
            hr = m_pDefaultCaptureDevice->Stop();
        }

        if (m_pCaptureClient)
        {
            m_pCaptureClient->Release();
            m_pCaptureClient = NULL;
        }

        if (m_pDefaultCaptureDevice)
        {
            m_pDefaultCaptureDevice->Release();
            m_pDefaultCaptureDevice = NULL;
        }

        if (m_waveFormatEx)
        {
            CoTaskMemFree((LPVOID)m_waveFormatEx);
            m_waveFormatEx = NULL;
        }

        if (SUCCEEDED(hr))
        {
            started = false;
            ret = true;
        }
    }

    return ret;
}
```

Rendering audio using WASAPI

To render audio using WASAPI, the render device and client must first be initialized:

```
/**
 * Start audio rendering using WASAPI.
 * @return The success of the operation.
 */
bool WasapiAudio::StartAudioRender()
{
    bool ret = false;

    if (!started)
    {
        HRESULT hr = InitRender();
        if (SUCCEEDED(hr))
        {
            ret = started = true;
        }
    }
```

```
    return ret;
}

/**
 * Initialize WASAPI audio render device.
 * @return The success of the operation.
 */
HRESULT WasapiAudio::InitRender()
{
    HRESULT hr = E_FAIL;

    LPCWSTR renderId = GetDefaultAudioRenderId(AudioDeviceRole::Default);

    if (NULL == renderId)
    {
        hr = E_FAIL;
    }
    else
    {
        hr = ActivateAudioInterface(renderId, __uuidof(IAudioClient2),
(void**)&m_pDefaultRenderDevice);
    }

    if (SUCCEEDED(hr))
    {
        hr = m_pDefaultRenderDevice->GetMixFormat(&m_waveFormatEx);
    }

    // Set the category through SetClientProperties
    AudioClientProperties properties = {};
    if (SUCCEEDED(hr))
    {
        properties.cbSize = sizeof AudioClientProperties;
        properties.eCategory = AudioCategory_Other;
        hr = m_pDefaultRenderDevice->SetClientProperties(&properties);
    }

    if (SUCCEEDED(hr))
    {
        WAVEFORMATEX temp;
        MyFillPcmFormat(temp, 2, 44100, 16); // stereo, 44100 Hz, 16 bit

        *m_waveFormatEx = temp;
        m_sourceFrameSizeInBytes = (m_waveFormatEx->wBitsPerSample / 8) *
m_waveFormatEx->nChannels;

        hr = m_pDefaultRenderDevice->Initialize(AUDCLNT_SHAREMODE_SHARED, 0x88000000,
1000 * 10000, 0, m_waveFormatEx, NULL);
    }

    if (SUCCEEDED(hr))
    {
        hr = m_pDefaultRenderDevice->GetService(__uuidof(IAudioRenderClient),
(void**)&m_pRenderClient);
    }

    if (SUCCEEDED(hr))
```

```
    {
        hr = m_pDefaultRenderDevice->Start();
    }

    if (renderId)
    {
        CoTaskMemFree((LPVOID)renderId);
    }

    return hr;
}
```

In the AudioRecorder example, the audio data to be rendered is given to Windows Phone Runtime Component before rendering takes place. Other, a more memory friendly way, would be to give the audio data piece by piece.

```
/**
 * Set audio data to be rendered.
 * @param byteArray The byte array to be rendered.
 */
void WasapiAudio::SetAudioBytes(const Platform::Array<byte>^ byteArray)
{
    delete audioBytes;

     // no need for the wav-header
    audioBytes = new BYTE[byteArray->Length-44];
    int availableBytes = byteArray->Length-44;

    for(long i = 0; i < availableBytes; i++)
    {
        audioBytes[i] = byteArray[i+44];
    }

    audioIndex = 0;
    audioByteCount = availableBytes;
}
```

The AudioRecorder example uses an update mechanism to feed audio data for WASAPI to render by frequently calling Update method.

```
/**
 * Feeds the render device with audio data.
 * @return True if there is audio data be rendered.
 */
bool WasapiAudio::Update()
{
    bool ret = false;
    if (!started) return ret;

    HRESULT hr = S_OK;
    UINT32 bufferFrameCount;
    UINT32 numFramesPadding;
    UINT32 numFramesAvailable;
    BYTE *pData = NULL;
    DWORD flags = 0;
```

```
    // Get the actual size of the allocated buffer.
    hr = m_pDefaultRenderDevice->GetBufferSize(&bufferFrameCount);

    // See how much buffer space is available.
    if (SUCCEEDED(hr))
    {
        hr = m_pDefaultRenderDevice->GetCurrentPadding( &numFramesPadding);
        numFramesAvailable = bufferFrameCount - numFramesPadding;
    }

    if (SUCCEEDED(hr))
    {
        // Grab all the available space in the shared buffer.
        hr = m_pRenderClient->GetBuffer(numFramesAvailable, &pData);
    }

    if (SUCCEEDED(hr))
    {
        if (audioIndex + (long)numFramesAvailable * m_sourceFrameSizeInBytes <
audioByteCount)
        {
            memcpy(pData, audioBytes+audioIndex, numFramesAvailable *
m_sourceFrameSizeInBytes);
            audioIndex += numFramesAvailable * m_sourceFrameSizeInBytes;
            hr = m_pRenderClient->ReleaseBuffer(numFramesAvailable, 0);
        }
        else
        {
            hr = m_pRenderClient->ReleaseBuffer(0, AUDCLNT_BUFFERFLAGS_SILENT);
        }
    }

    if (SUCCEEDED(hr))
    {
        hr = m_pDefaultRenderDevice->GetCurrentPadding(&numFramesPadding);
    }

    if (SUCCEEDED(hr) && numFramesPadding > 0)
    {
        ret = true;
    }

    return ret;
}
```

To stop rendering audio, the render device is stopped and then released alongside capturing client:

```
/**
 * Stop audio rendering using WASAPI.
 * @return The success of the operation.
 */
bool WasapiAudio::StopAudioRender()
{
    bool ret = false;

    if (started)
```

```
    {
        HRESULT hr = S_OK;

        if (m_pDefaultRenderDevice)
        {
            hr = m_pDefaultRenderDevice->Stop();
        }

        if (m_pRenderClient)
        {
            m_pRenderClient->Release();
            m_pRenderClient = NULL;
        }

        if (m_pDefaultRenderDevice)
        {
            m_pDefaultRenderDevice->Release();
            m_pDefaultRenderDevice = NULL;
        }

        if (m_waveFormatEx)
        {
            CoTaskMemFree((LPVOID)m_waveFormatEx);
            m_waveFormatEx = NULL;
        }

        if (SUCCEEDED(hr))
        {
            started = false;
            ret = true;
        }
    }

    return ret;
}
```

Capture and render AAC encoded audio

This code extract shows how to capture AAC encoded audio with AudioVideoCaptureDevice and render it later with MediaElement. Remember that all applications wanting to capture audio need `ID_CAP_MICROPHONE` capability.

For audio rendering purpose a `MediaElement` is defined in **MainPage.xaml** alongside with some buttons to control recording and playing audio.

```
<phone:PhoneApplicationPage
    x:Class="AudioVideoRecorder.MainPage"
    ...

    <!--LayoutRoot is the root grid where all page content is placed-->
    <Grid x:Name="LayoutRoot" Background="Transparent">
        ...

        <!--ContentPanel - place additional content here-->
        <Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">

            <StackPanel>
                <Button x:Name="RecordButton" Content="Record"
```

```
Click="RecordButton_Click"/>
                <Button x:Name="StopButton" Content="Stop" Click="StopButton_Click"/>
                <Button x:Name="PlayButton" Content="Play" Click="PlayButton_Click"/>
                <MediaElement x:Name="AudioElement" MediaEnded="MediaEnded"/>
            </StackPanel>
        </Grid>
    </Grid>
</phone:PhoneApplicationPage>
```

Event handlers and helper functions in code behind file **MainPage.xaml.cs** perform actual work to capture and play audio.

```
namespace AudioVideoRecorder
{
    public partial class MainPage : PhoneApplicationPage
    {
        private AudioVideoCaptureDevice mic;
        private IRandomAccessStream randomAccessStream;
        private Boolean isRecording = false;

        public MainPage()
        {
            InitializeComponent();
        }

        private async void RecordButton_Click(object sender, RoutedEventArgs e)
        {
            await MicStartAsync();
            await StartRecordingAsync();
            isRecording = true;
            PlayButton.IsEnabled = false;
            RecordButton.IsEnabled = false;
            StopButton.IsEnabled = true;
        }

        private async void StopButton_Click(object sender, RoutedEventArgs e)
        {
            if (isRecording)
            {
                await StopRecordingAsync();
                isRecording = false;
            }
            else
            {
                AudioElement.Stop();
            }

            PlayButton.IsEnabled = true;
            RecordButton.IsEnabled = true;
            StopButton.IsEnabled = false;
        }

        private void PlayButton_Click(object sender, RoutedEventArgs e)
        {
            if (AudioElement.Source != null && AudioElement.Source.ToString().Length >
0)
            {
```

```
                AudioElement.Play();
            }
            else
            {
                try
                {
                    using (var isf = IsolatedStorageFile.GetUserStoreForApplication())
                    {
                        using (var isfs = new IsolatedStorageFileStream("test.aac",
FileMode.Open, isf))
                        {
                            AudioElement.SetSource(isfs);
                        }
                    }
                }
                catch (Exception ex)
                {
                }
            }

            PlayButton.IsEnabled = false;
            RecordButton.IsEnabled = false;
            StopButton.IsEnabled = true;
        }

        public async Task MicStartAsync()
        {
            mic = await AudioVideoCaptureDevice.OpenForAudioOnlyAsync();
        }

        public async Task StartRecordingAsync()
        {
            await CreateFileStreamForAudioAsync("test.aac");
            await mic.StartRecordingToStreamAsync(randomAccessStream);
        }

        public async Task StopRecordingAsync()
        {
            await mic.StopRecordingAsync();
            randomAccessStream.AsStream().Dispose();
        }

        public async Task CreateFileStreamForAudioAsync(string fileName)
        {
            StorageFolder local = Windows.Storage.ApplicationData.Current.LocalFolder;
            StorageFile file = null;

            try
            {
                file = await local.GetFileAsync(fileName);
            }
            catch (Exception ex)
            {
            }

            if (file == null)
```

```
            {
                file = await local.CreateFileAsync(fileName);
            }

            // To free the file, the AudioElement Source is cleared.
            AudioElement.ClearValue(MediaElement.SourceProperty);
            randomAccessStream = await file.OpenAsync(FileAccessMode.ReadWrite);
        }

        private void MediaEnded(object sender, RoutedEventArgs e)
        {
            PlayButton.IsEnabled = true;
            RecordButton.IsEnabled = true;
            StopButton.IsEnabled = false;
        }
    }
}
```

Other audio related information

Using wave format in files

- Streaming Data from a WAV File
- Saving Microphone stream to wave format in Windows Phone
- Windows Dev Center on WAVEFORMATEX structure

Sharing to My Music / Music Hub / Music Library

- Windows Phone 8: Media file access
- Saving a file to a specific location on WP8 device
- API to add playlists in Zune