```
 1  using System;
 2  using System.Collections.Generic;
 3  using System.IO;
 4  using System.Linq;
 5  using System.Runtime.InteropServices.WindowsRuntime;
 6  using Windows.Foundation;
 7  using Windows.Foundation.Collections;
 8  using Windows.UI.Xaml;
 9  using Windows.UI.Xaml.Controls;
10  using Windows.UI.Xaml.Controls.Primitives;
11  using Windows.UI.Xaml.Data;
12  using Windows.UI.Xaml.Input;
13  using Windows.UI.Xaml.Media;
14  using Windows.UI.Xaml.Navigation;
15
16  //Import needed assemblies
17  using Microsoft.MixedReality.WebRTC;
18  using System.Diagnostics;
19  using Windows.Media.Capture;
20  using Windows.ApplicationModel;
21  using TestAppUwp.Video; //Namespace in StreamSamplePool.cs and VideoBridge.cs
22  using Windows.Media.Core;
23  using Windows.Media.Playback;
24  using Windows.Media.MediaProperties;
25  using TestAppUwp; //Namespace in NodeDssSignaler.cs
26
27
28  namespace MR_Stream_UWP
29  {
30
31      public sealed partial class MainPage : Page
32      {
33          //Main Page Class Variables
34          private PeerConnection _peerConnection; //Private variable of type
              PeerConnection
35          private MediaStreamSource _localVideoSource; //Variable for wrapping
              local video stream for Media Foundation playback pipeline
36          private VideoBridge _localVideoBridge = new VideoBridge(3); //Variable
              for managing video frame delivery to Media Foundation playback pipeline
               (queue capacity = 3 frames)
37          private bool _localVideoPlaying = false; //Boolean to indicate whether
              local video is playing
38          private object _localVideoLock = new object(); //Lock to ensure
              I420VideoFrameReady and SampleRequest events are not fired in parallel
              on multiple threads
39          private NodeDssSignaler _signaler; //NodeDssSignaler
40          private object _remoteVideoLock = new object();
41          private bool _remoteVideoPlaying = false;
42          private MediaStreamSource _remoteVideoSource;
43          private VideoBridge _remoteVideoBridge = new VideoBridge(5);
44          private LocalAudioTrack _localAudioTrack; //Audio Track
45          private LocalVideoTrack _localVideoTrack; //Video Track
```

```
46
47          public MainPage()
48          {
49              this.InitializeComponent(); //Initialize mainpage
50              this.Loaded += OnLoaded; //Handle the loaded event when XAML user
                  interface is loaded
51              Application.Current.Suspending += App_Suspending; //Handle the
                  application exit event to clean up resources
52          }
53
54          //Function to handle on loaded event
55          private async void OnLoaded(object sender, RoutedEventArgs e)
56          {
57              ////////////////////// ACCESS WEBCAM AND
                  MICROPHONE //////////////////////
58
59              //Request access to microphone and camera
60              var settings = new MediaCaptureInitializationSettings();
61              settings.StreamingCaptureMode = StreamingCaptureMode.AudioAndVideo;
62              var capture = new MediaCapture();
63              await capture.InitializeAsync(settings); //Initialize the media
                  capture with the associated settings
64
65              //Get a list of the available video capture devices (i.e. the
                  webcams)
66              IReadOnlyList<VideoCaptureDevice> deviceList = await
                  DeviceVideoTrackSource.GetCaptureDevicesAsync();
67
68              //Print the video capture devices to the Debug console
69              foreach (var device in deviceList)
70              {
71                  Debugger.Log(0, "", $"Webcam {device.name} (id: {device.id})\n");
72              }
73
74              ////////////////////// CREATING A PEER CONNECTION //////////////////////
75
76              //Instantiate the peer connection
77              _peerConnection = new PeerConnection();
78
79              //Create a PeerConnectionConfiguation object to hold the peer
                  connection configuration parameters
80              var config = new PeerConnectionConfiguration
81              {
82                  IceServers = new List<IceServer>
83                  {
84                      new IceServer{ Urls = { "stun:stun.l.google.com:19302" } } //
                        Free google STUN server (NOT TO BE USED FOR PRODUCTION)
85                  }
86              };
87
88              //If previous code runs, print a success message to the debugger
89              Debugger.Log(0, "", "Peer connection initialized successfully.\n");
```

```
90
91              ////////////////////// CREATE AUDIO AND VIDEO TRACKS //////////////////////
92
93              //Declare private variables to store audio/video tracks and their sources
94              DeviceAudioTrackSource _microphoneSource; //Audio Source
95              DeviceVideoTrackSource _webcamSource; //Video Source
96              //LocalAudioTrack _localAudioTrack; //Audio Track
97              //LocalVideoTrack _localVideoTrack; //Video Track
98              RemoteVideoTrack _remoteVideoTrack; //Remote video track
99
100             //Create a new video track source to obtain the frames of the local video capture device (i.e. the webcam)
101             //May want to include a LocalVideoDeviceInitConfig Object to set the video configuration, here it is left default
102             _webcamSource = await DeviceVideoTrackSource.CreateAsync();
103
104             //Subscribe to the I420VideoFrameReady Event
105             _webcamSource.I420AVideoFrameReady += LocalI420AFrameReady;
106
107             //Set up the configuration of the local video track
108             var videoTrackConfig = new LocalVideoTrackInitConfig
109             {
110                 trackName = "webcam_track"
111             };
112
113             //Create the local video track using the webcam source and the configuration profile
114             _localVideoTrack = LocalVideoTrack.CreateFromSource(_webcamSource, videoTrackConfig);
115
116             //Create the local audio track source
117             _microphoneSource = await DeviceAudioTrackSource.CreateAsync();
118
119             //Set up the configuration of the local audio track
120             var audioTrackConfig = new LocalAudioTrackInitConfig
121             {
122                 trackName = "microphone_track"
123             };
124
125             //Create the local audio track using the audio source and the configuration profile
126             _localAudioTrack = LocalAudioTrack.CreateFromSource (_microphoneSource, audioTrackConfig);
127
128
129             ////////////////////// ADDING THE TRANSCEIVERS //////////////////////
130
131             ////Create the transceiver variables of type Tranciever
132             //Transceiver _audioTransceiver;
133             //Transceiver _videoTransceiver;
```

```
134
135             ////Create the audio/video transceivers using the AddTransceiver
                  method
136             //_audioTransceiver = _peerConnection.AddTransceiver
                  (MediaKind.Audio);
137             //_videoTransceiver = _peerConnection.AddTransceiver
                  (MediaKind.Video);
138
139             ////Attach the local audio/video tracks to the transceivers
140             //_audioTransceiver.LocalAudioTrack = _localAudioTrack;
141             //_videoTransceiver.LocalVideoTrack = _localVideoTrack;
142
143             ////////////////////// SINALING ////////////////////
144             //Subscribe to signaling events
145             _peerConnection.LocalSdpReadytoSend += Peer_LocalSdpReadytoSend;
146             _peerConnection.IceCandidateReadytoSend +=
                  Peer_IceCandidateReadytoSend;
147
148             // Initialize the signaler
149             _signaler = new NodeDssSignaler()
150             {
151                 //HttpServerAddress = "http://127.0.0.1:3000/",
152                 HttpServerAddress = "http://172.21.112.1:3000/", //Connected via
                      vEthernet IP, Settings>For Developers
153                 LocalPeerId = "UWP_APP",
154                 RemotePeerId = "HOLOLENS",
155             };
156
157             _signaler.OnMessage += async (NodeDssSignaler.Message msg) =>
158             {
159                 switch (msg.MessageType)
160                 {
161                     case NodeDssSignaler.Message.WireMessageType.Offer:
162                         // Wait for the offer to be applied
163                         await _peerConnection.SetRemoteDescriptionAsync
                          (msg.ToSdpMessage());
164                         // Once applied, create an answer
165                         _peerConnection.CreateAnswer();
166                         break;
167
168                     case NodeDssSignaler.Message.WireMessageType.Answer:
169                         // No need to await this call; we have nothing to do
                          after it
170                         _peerConnection.SetRemoteDescriptionAsync
                          (msg.ToSdpMessage());
171                         break;
172
173                     case NodeDssSignaler.Message.WireMessageType.Ice:
174                         _peerConnection.AddIceCandidate(msg.ToIceCandidate());
175                         break;
176                 }
177             };
```

```
178             _signaler.StartPollingAsync();
179
180             ////////////////////// ESTABLISHING WEB-RTC
                  CONNECTION //////////////////////
181
182             //Print the connection and ICE server status to Debug console
183             _peerConnection.Connected += () => {
184                 Debugger.Log(0, "", "PeerConnection: connected.\n");
185             };
186             _peerConnection.IceStateChanged += (IceConnectionState newState) => {
187                 Debugger.Log(0, "", $"ICE state: {newState}\n");
188             };
189
190
191             _peerConnection.VideoTrackAdded += (RemoteVideoTrack track) => {
192                 _remoteVideoTrack = track;
193                 _remoteVideoTrack.I420AVideoFrameReady +=
                      RemoteVideo_I420AFrameReady;
194             };
195         }
196
197         private void App_Suspending(object sender, SuspendingEventArgs e)
198         {
199             //If there is an active peer connection
200             if (_peerConnection != null)
201             {
202                 _peerConnection.Close(); //Close the peer connection
203                 _peerConnection.Dispose(); //Clean up peer connection resources
204                 _peerConnection = null; //set the peer connection variable to
                      null
205             }
206
207             localVideoPlayerElement.SetMediaPlayer(null); //Clear the media
                  player of localVideoPlayerElement
208
209             //Stop the signaler and clean up resources
210             if (_signaler != null)
211             {
212                 _signaler.StopPollingAsync();
213                 _signaler = null;
214             }
215
216             //Clear the media player of remote content
217             remoteVideoPlayerElement.SetMediaPlayer(null);
218         }
219
220         //Utility Method to build MediaStramSource instance to encapsulate video
              stream in I420 format
221         //Note webRTC uses I420 format
222         private MediaStreamSource CreateI420VideoStreamSource(uint width, uint
              height, int framerate)
223         {
```

```
224             if (width == 0)
225             {
226                 throw new ArgumentException("Invalid zero width for video.",
                      "width");
227             }
228             if (height == 0)
229             {
230                 throw new ArgumentException("Invalid zero height for video.",
                      "height");
231             }
232             // Note: IYUV and I420 have same memory layout (though different
                  FOURCC)
233             // https://docs.microsoft.com/en-us/windows/desktop/medfound/video-
                  subtype-guids
234             var videoProperties = VideoEncodingProperties.CreateUncompressed(
235                 MediaEncodingSubtypes.Iyuv, width, height);
236             var videoStreamDesc = new VideoStreamDescriptor(videoProperties);
237             videoStreamDesc.EncodingProperties.FrameRate.Numerator = (uint)
                  framerate;
238             videoStreamDesc.EncodingProperties.FrameRate.Denominator = 1;
239             // Bitrate in bits per second : framerate * frame pixel size *
                  I420=12bpp
240             videoStreamDesc.EncodingProperties.Bitrate = ((uint)framerate * width
                   * height * 12);
241             var videoStreamSource = new MediaStreamSource(videoStreamDesc);
242             videoStreamSource.BufferTime = TimeSpan.Zero;
243             videoStreamSource.SampleRequested += OnMediaStreamSourceRequested;
244             videoStreamSource.IsLive = true; // Enables optimizations for live
                  sources
245             videoStreamSource.CanSeek = false; // Cannot seek live WebRTC video
                  stream
246             return videoStreamSource;
247         }
248
249         //Handler for the MediaStreamSourceRequested, finds a suitable video
              bridge based on source that invoked event
250         private void OnMediaStreamSourceRequested(MediaStreamSource
              sender,MediaStreamSourceSampleRequestedEventArgs args)
251         {
252             VideoBridge videoBridge;
253             if (sender == _localVideoSource)
254                 videoBridge = _localVideoBridge;
255             else if (sender == _remoteVideoSource)
256                 videoBridge = _remoteVideoBridge;
257             else
258                 return;
259             videoBridge.TryServeVideoFrame(args);
260         }
261
262         //I420 Frame Ready event handler, enqueues new captured video frames into
                the video bridge
263         private void LocalI420AFrameReady(I420AVideoFrame frame)
```

```
264         {
265             lock (_localVideoLock)
266             {
267                 if (!_localVideoPlaying)
268                 {
269                     _localVideoPlaying = true;
270 
271                     // Capture the resolution into local variable useable from the lambda below
272                     uint width = frame.width;
273                     uint height = frame.height;
274 
275                     // Defer UI-related work to the main UI thread
276                     RunOnMainThread(() =>
277                     {
278                         // Bridge the local video track with the local media player UI
279                         int framerate = 30; // assumed, for lack of an actual value
280                         _localVideoSource = CreateI420VideoStreamSource(
281                             width, height, framerate);
282                         var localVideoPlayer = new MediaPlayer();
283                         localVideoPlayer.Source = MediaSource.CreateFromMediaStreamSource(
284                             _localVideoSource);
285                         localVideoPlayerElement.SetMediaPlayer(localVideoPlayer);
286                         localVideoPlayer.Play();
287                     });
288                 }
289             }
290             // Enqueue the incoming frame into the video bridge; the media player will
291             // later dequeue it as soon as it's ready.
292             _localVideoBridge.HandleIncomingVideoFrame(frame);
293         }
294 
295         //Handler to defer work to main thread
296         private void RunOnMainThread(Windows.UI.Core.DispatchedHandler handler)
297         {
298             if (Dispatcher.HasThreadAccess)
299             {
300                 handler.Invoke();
301             }
302             else
303             {
304                 // Note: use a discard "_" to silence CS4014 warning
305                 _ = Dispatcher.RunAsync
306                   (Windows.UI.Core.CoreDispatcherPriority.Normal, handler);
307             }
308         }
309 
310         private void Peer_LocalSdpReadytoSend(SdpMessage message)
```

```
310         {
311             var msg = NodeDssSignaler.Message.FromSdpMessage(message);
312             _signaler.SendMessageAsync(msg);
313         }
314 
315         private void Peer_IceCandidateReadytoSend(IceCandidate iceCandidate)
316         {
317             var msg = NodeDssSignaler.Message.FromIceCandidate(iceCandidate);
318             _signaler.SendMessageAsync(msg);
319         }
320 
321         private void RemoteVideo_I420AFrameReady(I420AVideoFrame frame)
322         {
323             lock (_remoteVideoLock)
324             {
325                 if (!_remoteVideoPlaying)
326                 {
327                     _remoteVideoPlaying = true;
328                     uint width = frame.width;
329                     uint height = frame.height;
330                     RunOnMainThread(() =>
331                     {
332                         // Bridge the remote video track with the remote media player UI
333                         int framerate = 30; // assumed, for lack of an actual value
334                         _remoteVideoSource = CreateI420VideoStreamSource(width, height,
335                             framerate);
336                         var remoteVideoPlayer = new MediaPlayer();
337                         remoteVideoPlayer.Source = MediaSource.CreateFromMediaStreamSource(
338                             _remoteVideoSource);
339                         remoteVideoPlayerElement.SetMediaPlayer(remoteVideoPlayer);
340                         remoteVideoPlayer.Play();
341                     });
342                 }
343             }
344             _remoteVideoBridge.HandleIncomingVideoFrame(frame);
345         }
346 
347 
348         //Handler for the on-click event of the ConnectButton
349         private void ConnectButton_Click(object sender, RoutedEventArgs e)
350         {
351             Debugger.Log(0, "", "You have clicked the connect button\n");
352 
353             _peerConnection.CreateOffer();
354 
355             //Create the transceiver variables of type Tranciever
356             Transceiver _audioTransceiver;
```

```
357             Transceiver _videoTransceiver;
358 
359             //Create the audio/video transceivers using the AddTransceiver method
360             _audioTransceiver = _peerConnection.AddTransceiver(MediaKind.Audio);
361             _videoTransceiver = _peerConnection.AddTransceiver(MediaKind.Video);
362 
363             //Attach the local audio/video tracks to the transceivers
364             _audioTransceiver.LocalAudioTrack = _localAudioTrack;
365             _videoTransceiver.LocalVideoTrack = _localVideoTrack;
366 
367         }
368     }
369 }
370 
```