**Preface**

This documentation contains information on the Apollo Lens project that is relevant to various audiences. To help clarify what is most necessary to specific audiences, the following tags will be used to describe which audience(s) a section have been written for.

**[USER]** Relevant for users, such as surgeons, who intend on using this program for viewing DICOM images and/or streaming live video.

**[TECH]** Relevant for any reader interested in a more technical description of the applications and functionality, likely for review or certification purposes.

**[DEVS]** Relevant to readers who are working to further develop the Apollo Lens project.

**ApolloLens Documentation v1.0**

# Overview

**[USER]** **[TECH] [DEVS]**

The Apollo Lens project is a suite of applications for the Microsoft HoloLens that allows a surgeon to virtually place interactive screens in the Operating Room (OR) for viewing DICOM image files, as well as sending and/or receiving a video feed between the surgeon’s HoloLens and an external video source such as a laparoscopic camera.

Instead of requiring physical monitors in the OR for viewing data such as DICOM image libraries or live video feeds, and potentially requiring another person to assist with adjusting the brightness and contrast of DICOM images, Apollo Lens users can access this data on their own. Using hand signals and voice commands, they can easily place and move screens for viewing data, watching a live feed from a laparoscope, navigating through a DICOM image library and modifying the brightness and contrast of individual images.

# Applications

**[USER]** **[TECH] [DEVS]**

The overall Apollo Lens project is comprised of several applications for both HoloLens and Desktop. Devices must be connected to the internet to use each of these applications, and HoloLens devices must be connected to the internet via Wi-Fi. Descriptions of each of these are described below.

## Scan Gallery

HoloLens application that allows users to virtually view and interact with a series of DICOM medical images. These images may be stored on the HoloLens, or they may be loaded from another device running the Scan Server application. After selecting an image set, users will be able to navigate sequentially through the collection of loaded images. They also may manipulate the brightness and contrast of any given image, as well as restore it to its original state.

## Scan Server

Desktop application used to load DICOM medical images to the HoloLens. DICOM image libraries must be downloaded to the computer running Scan Server. With this application, a user can use a file browser to locate the folder containing the DICOM image files and send these to the HoloLens for viewing.

## Apollo Lens Client

HoloLens application that allows the user to view a live video feed from another device. This application will only connect to a device running Apollo Lens Source.

## Apollo Lens Source

Desktop application for streaming video to a HoloLens running Apollo Lens Client. Specifications for framerate and resolution can be modified before streaming to the device.

## [DEVS] Apollo Lens Basic

Desktop application for displaying a live video feed from the current device’s camera. This is used solely for testing/debugging purposes, and it will not send a video over the network.

# Usage

**[USER]** **[TECH] [DEVS]**

The following sections describe how to setup and use the Apollo Lens system for viewing DICOM images and a live video feed. Devices must be connected to the internet to use each of these applications, and the HoloLens device must be connected to the internet via Wi-Fi. Diagrams for these sections will use the following graphics:

A close up of a logo

Description automatically generated

**Microsoft HoloLens device [1]**

Used for running Apollo Lens HoloLens applications

Must be connected to the internet via Wi-Fi

**Computer**

Used for running Apollo Lens Desktop applications

Must be connected to the internet

**Laparoscopic camera**

Needs to stream video to a computer running Apollo Lens Source

**AWS Server**

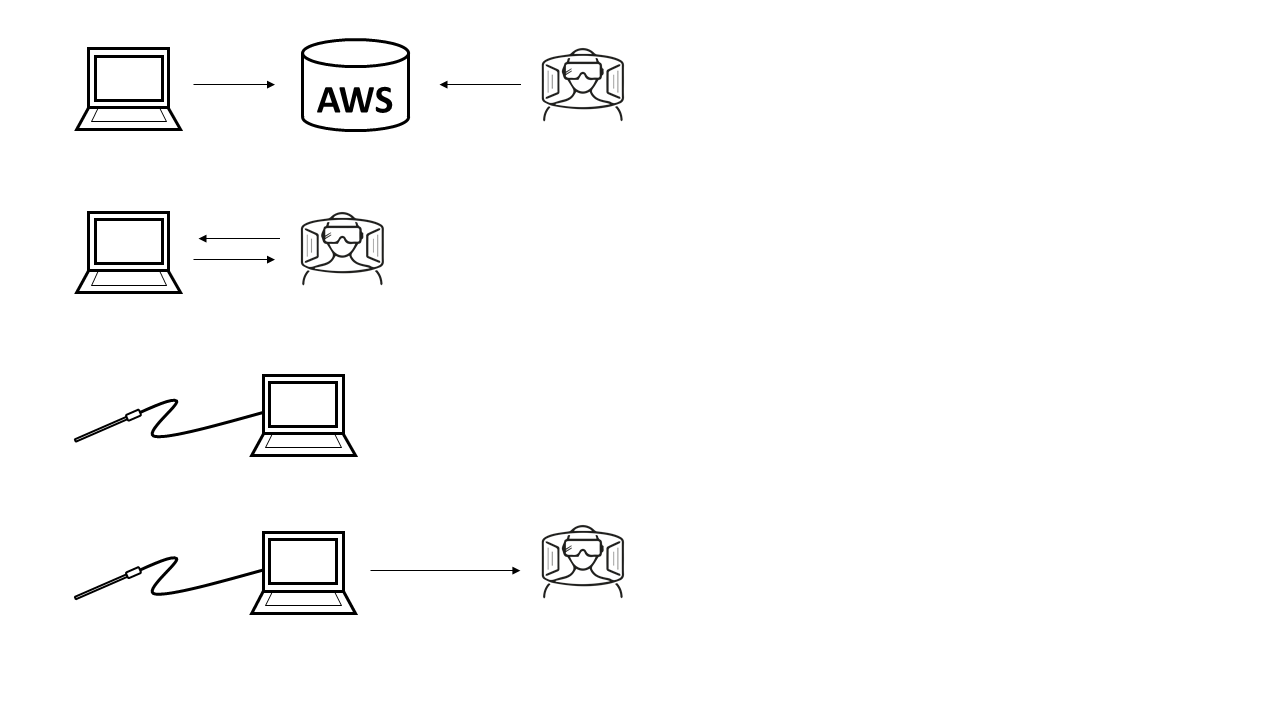
Used to connect devices for sending/receiving video

## Viewing a Live Video Feed

Requirements:

* HoloLens device running Apollo Lens Client
* Computer running Apollo Lens Source
* Camera source connected to the computer running Apollo Lens Source (i.e. laparoscope)

A HoloLens running Apollo Lens Client (Client) can receive a live video feed from a computer running Apollo Lens Source (Source). Both devices must be connected to the internet, and the HoloLens must be connected to the internet via Wi-Fi. The computer must be connected to a camera, such as a laparoscope, to capture video. Figure 1 shown below outlines the setup for viewing a live video feed on a HoloLens device.



Running **ApolloLensSource**

Select **“Connect to Source”**

Ensure the laparoscopic camera is connected to the computer

A private connection between the two systems will then be established

Connect both to the AWS signaling server  
The two systems will exchange network information  
to setup a private connection

Running **ApolloLensClient**

Video will then be sent from the computer to the HoloLens

It is never sent to the AWS server

**Figure 1** Setup for laparoscopic video feed to HoloLens

First, launch Source on the computer. A button labeled “Connect to Signaling Server” will be printed to the screen, along with a log displaying “Initializing WebRTC” and “Done”. This log will display information regarding the connection between itself and a peer connection. Clicking the button will attempt to connect to the signaling server. Upon successful connection, two buttons will appear. These are labeled “Connect to Source” and “Say Hi”. The “Say Hi” button is used for testing connection to its peer. Clicking it sends a message containing the string “Hello, World!” to the connected peer device, and “Send message: Hello, World! to connected peers” is printed to the log. The delivered message will be printed in the peer’s log. Similarly, if the connected peer device sends a “Say Hi” message, “Hello, World!” will be printed to the log. Clicking “Connect to Source” will connect the user to an active instance of Source and begin displaying a live video feed. If the source closes its Source application, the feed will cease. The log will be hidden once “Connect to Source” is selected. Figure 2 shown below outlines the usage of Source.

A screenshot of a cell phone

Description automatically generated

Select “Connect to Server”

The camera will activate and begin streaming video to Apollo Lens Client

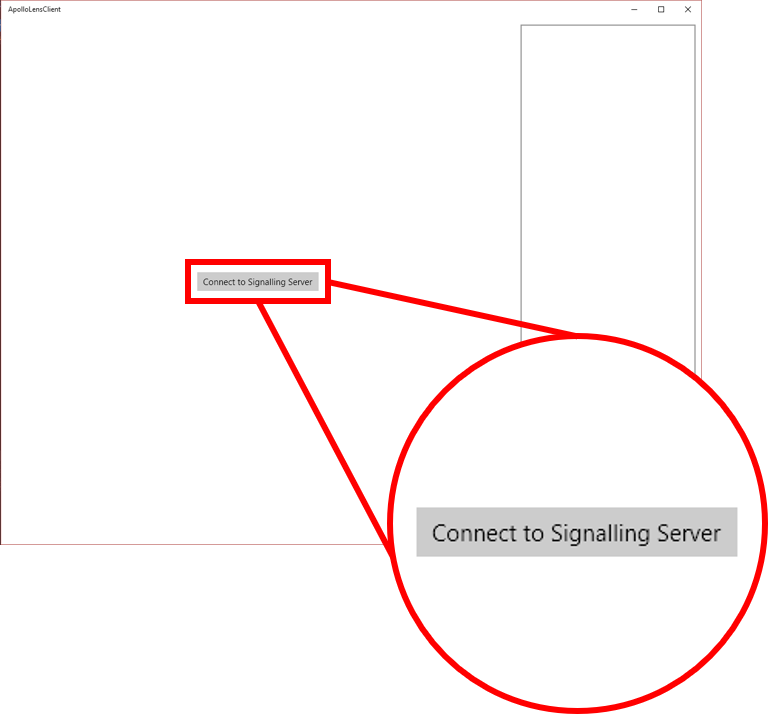
A picture containing screenshot

Description automatically generated

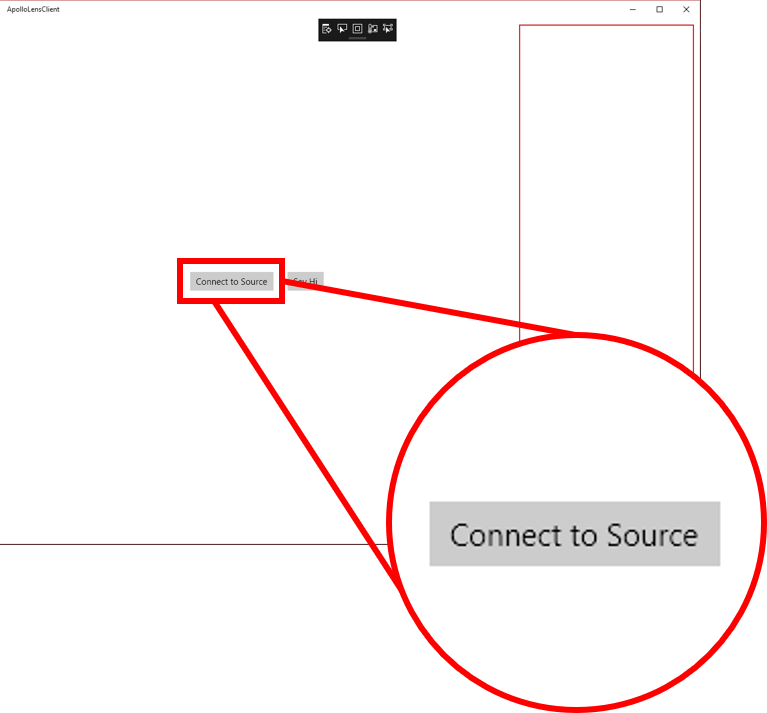
“Disconnect from Server”  
can be selected without interrupting the stream to Apollo Lens Client

**Figure 2** How to use Apollo Lens Source

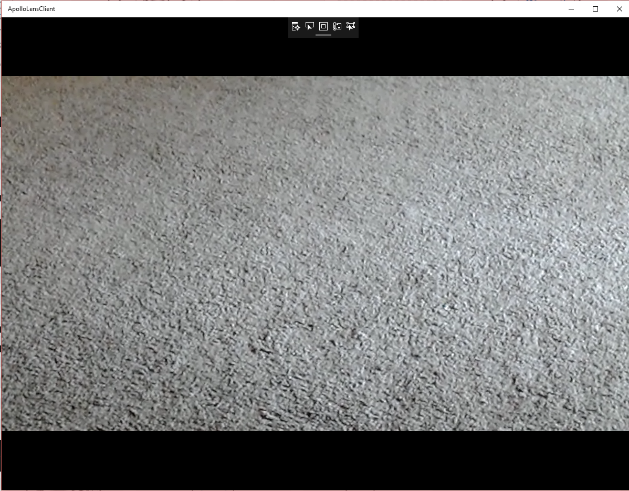
Then, launch Client on the HoloLens. A button labeled “Connect to Server”, two drop down tabs, and a log will be printed to the screen. This log will display “Initializing WebRtc” and “Done,” as well as information regarding the connection between itself and its peer upon connection. The two drop down tabs are used to set the media device and capture format. Media device refers to which camera device will be used for streaming from Source. Capture format refers to the quality of outbound video, specifically resolution and framerate. Clicking the “Connect to Server” button will attempt to connect to the signaling server. Upon successful connection, video will be ready to stream from the device, and when a device running Client connects, it will receive a live feed of video. Additionally, two buttons will appear, labeled “Disconnect from Server” and “Say Hi”. The “Say Hi” button is used for testing connection to its peer. Clicking it sends a message containing the string “Hello, World!” to the connected peer device, and “Send message: Hello, World! to connected peers” is printed to the log. The delivered message will be printed in the peer’s log. Similarly, if the connected peer device sends a “Say Hi” message, “Hello, World!’ will be printed to the log. Clicking “Disconnect from Server” will return the user to the initial screen for setting video specifications. Connection to the signaling server will cease, but video will continue streaming to an already connected instance of Client. Figure 3 shown below outline the usage of Client.



Select “Connect to Signalling Server”



Then select “Connect to Source”



Video stream displayed from  
Apollo Lens Source

**Figure 3** How to use Apollo Lens Client

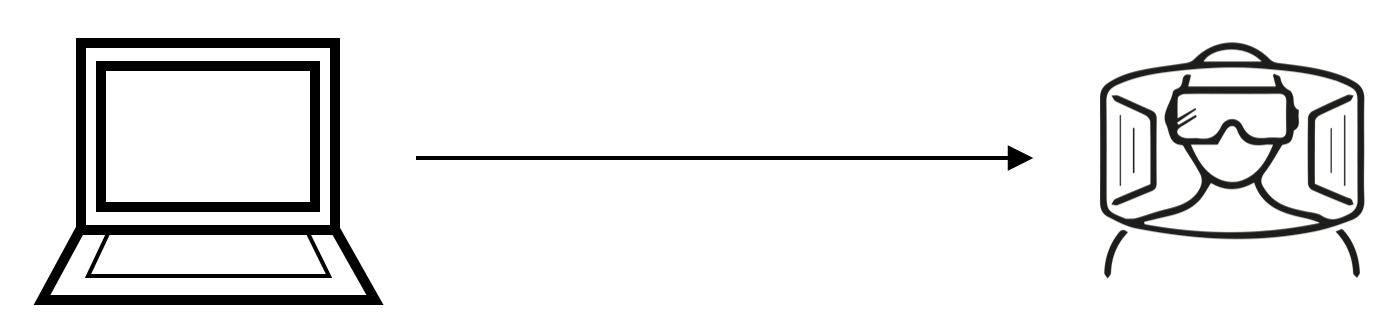
## Viewing DICOM Images

Requirements:

* HoloLens device running Scan Gallery
* Computer running Scan Server (optional)
* Set of DICOM image files, saved to either the HoloLens or the computer running Scan Server

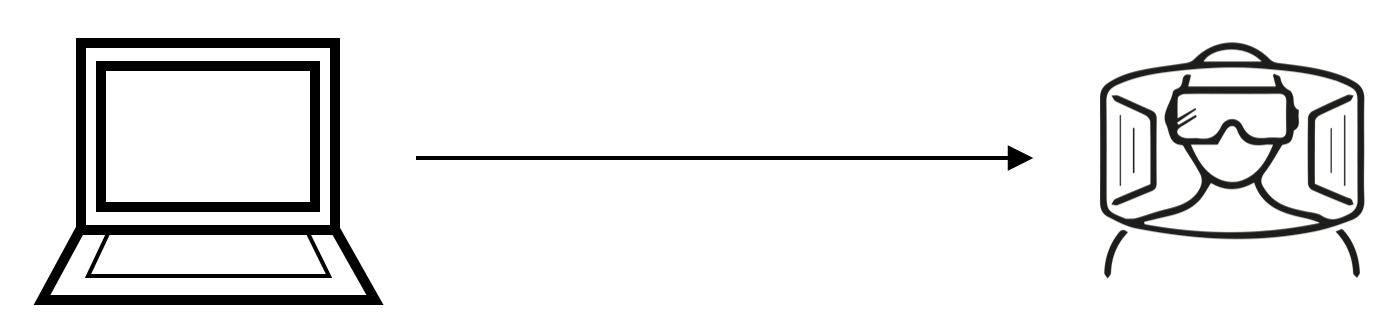
A HoloLens running Scan Gallery allows a user to virtually display and interact with a set of DICOM images. These images can be loaded to Scan Gallery for viewing if they have been saved to a HoloLens running Scan Gallery. Alternatively, DICOM images saved to a computer running Scan Server can be loaded to a HoloLens running Scan Gallery. Both of the setups for these methods are outlined below in Figures 4 and 5.

Running **ScanGallery**



Images are stored directly on the HoloLens

**Figure 4** Loading images from HoloLens to Scan Gallery



DICOM images sent to HoloLens

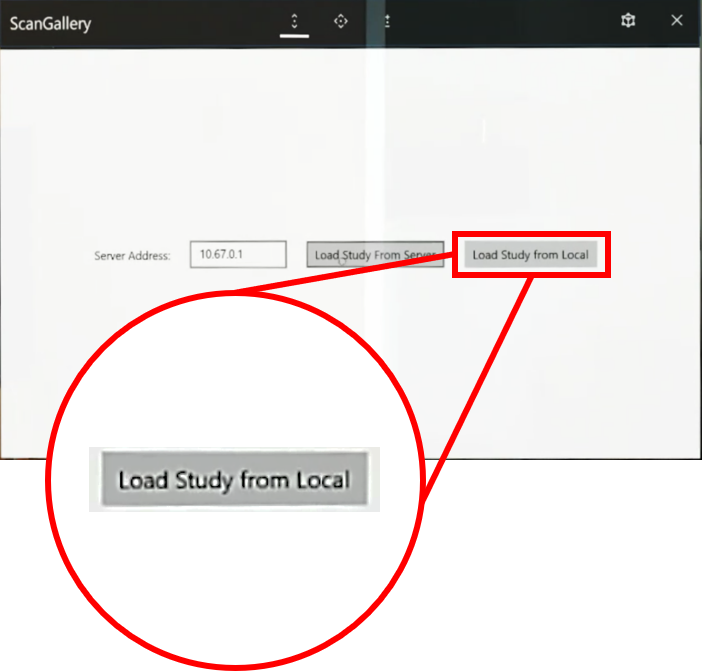
Running **ScanServer**

with DICOM images downloaded

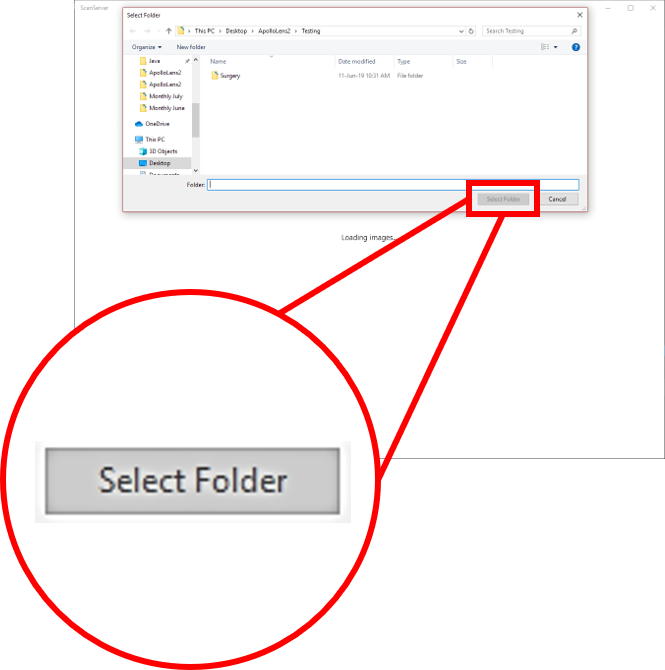
Running **ScanGallery**

**Figure 5** Loading images from Scan Server to Scan Gallery

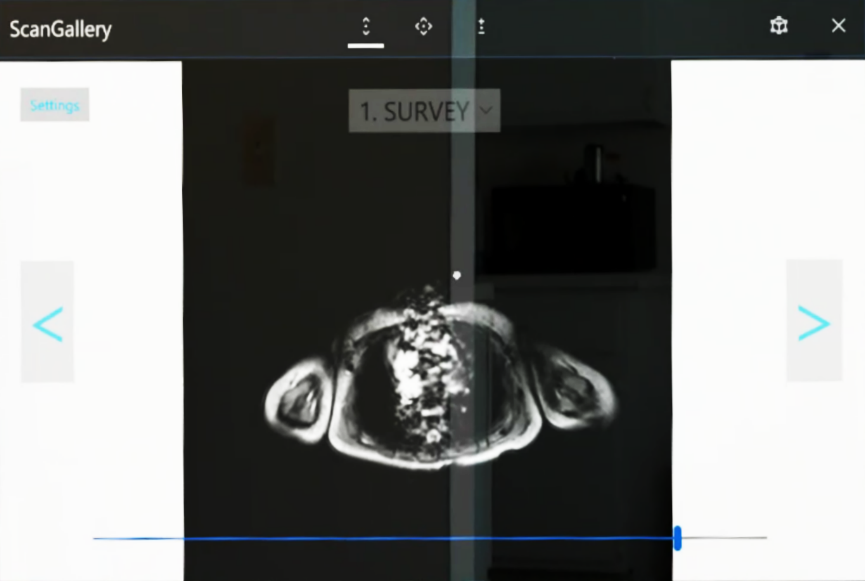
To begin viewing a saved DICOM image library, launch Scan Gallery on the HoloLens. If the desired DICOM image set has been saved to the HoloLens, select “Load Study From Local” and navigate to the folder containing the DICOM images. Figure 6 shown below outlines the process for loading a set of DICOM images stored on the HoloLens using Scan Gallery.



Select “Load Study from Local”



Navigate to the folder containing the DICOM images  
and select “Select Folder”



Selected image set will be loaded

**Figure 6** Setup for viewing DICOM images stored on HoloLens

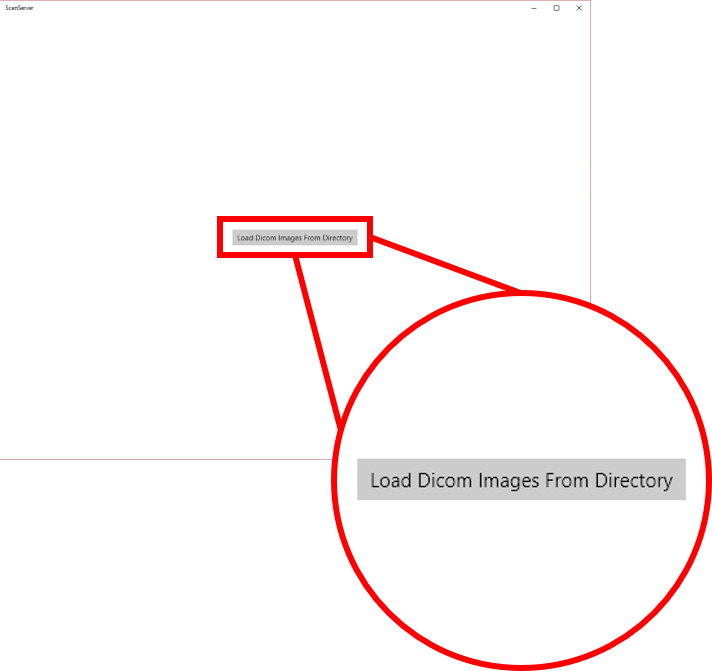
If the image library is instead stored on a computer, have that computer launch Scan Server. On the computer, click “Load Dicom Images From Directory” and navigate to the folder containing the DICOM images. The text “Waiting for connection” will be displayed to the screen once an image set has been selected.

NOTE: “Waiting for connection” will remain on the screen even after a connection has been established.

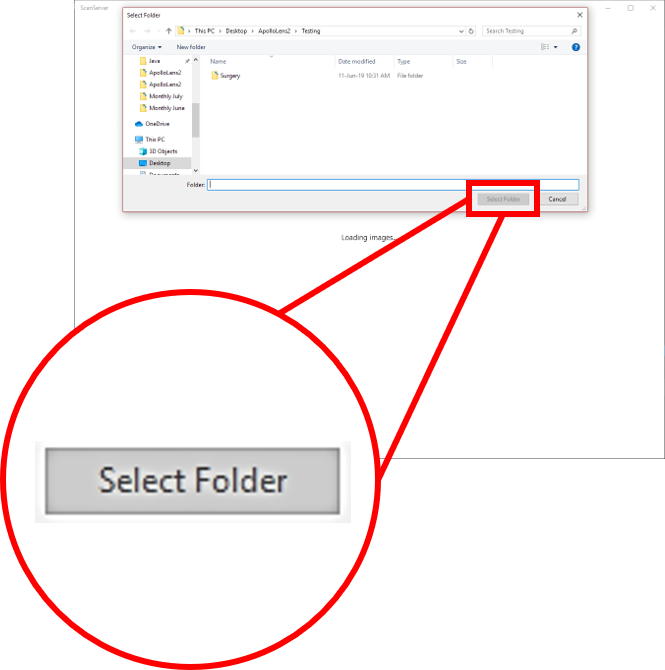
The IP address of the computer running Scan Server will be required to load its selected DICOM image set. Return to the computer running Scan Server and find its IP address. For Windows devices, the IP address can be found using the following steps:

1. Open Settings
2. Open Network & Internet
3. Under Status, select View your network properties
4. The required IP address can be found under “Default Gateway”

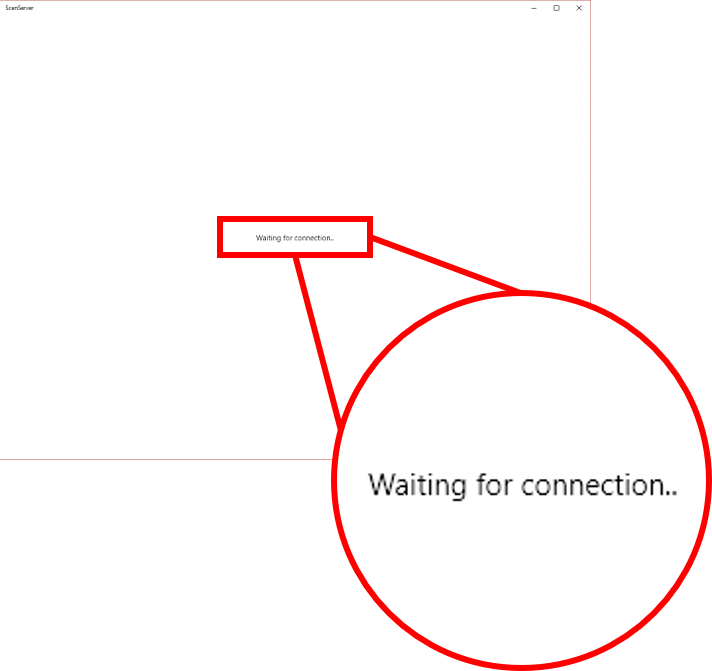
Return to the HoloLens and enter this address in Scan Gallery, and then select “Load Study From Server”. Figure 7 shown below outlines the process for loading a set of DICOM images stored on a computer, running Scan Server, to a HoloLens device, running Scan Gallery.



Launch Scan Server on Desktop  
and Select “Load Images from Directory”

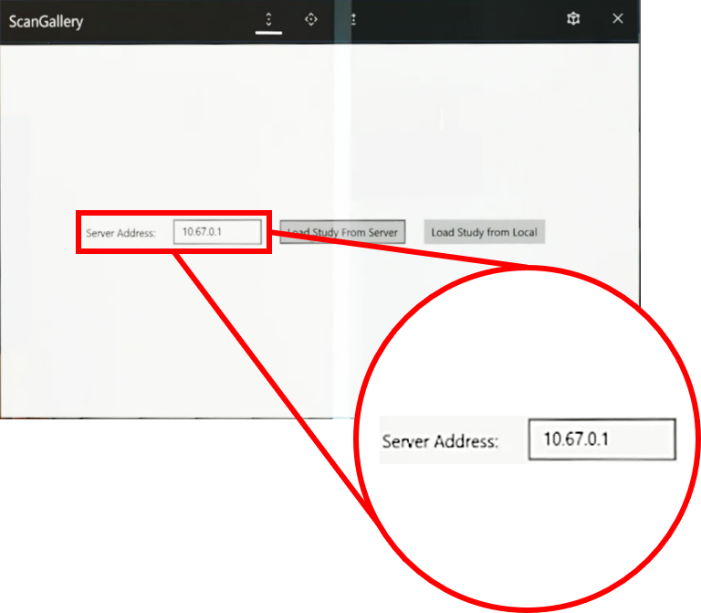


Navigate to the folder containing the DICOM images  
and select “Select Folder”

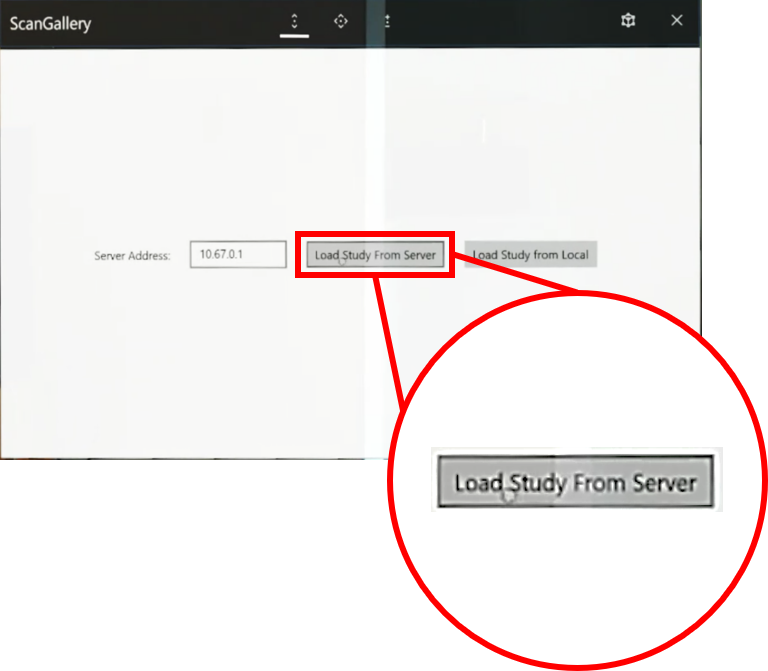


The image set can now be loaded to Scan Gallery on HoloLens

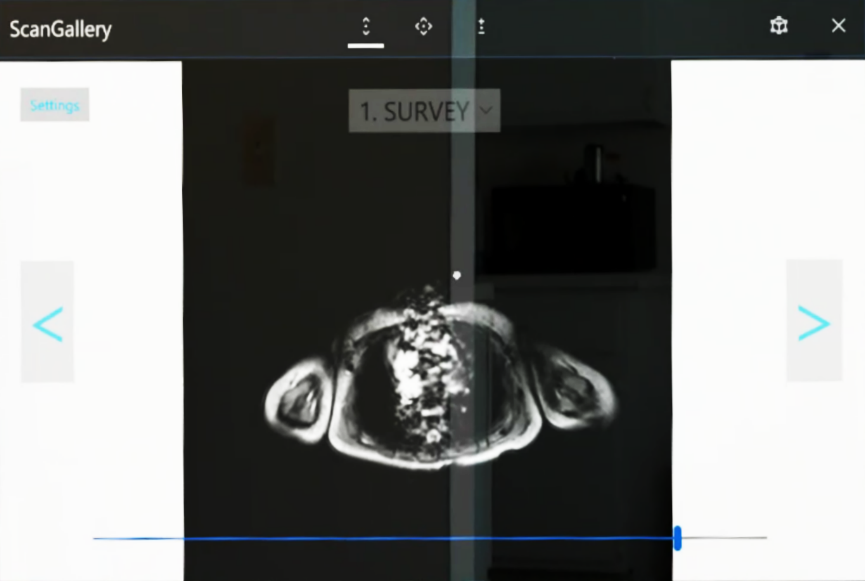
NOTE: “Waiting for connection” will remain on the screen even after a connection has been established.



Launch Scan Gallery on HoloLens  
and set the Server Address to the  
IP Address of the machine running Scan Server



Select “Load Study from Server”



Selected image set will be loaded

**Figure 7** Setup for loading DICOM images stored on a computer to HoloLens

After loading an image set from either Local or Server, the first image in the collection. A drop-down tab will be displayed in the top left corner containing the options to increase and decrease brightness and contrast, as well to restore the image to its original brightness and contrast values. This tab also allows the user to change the image display options. These are:

* **None:** Image is displayed at its original size
* **Fill:** Image is resized to fill the window, aspect ratio is not preserved
* **Uniform:** Image is resized to best fill the window while preserving its aspect ratio
* **UniformToFill:** Image is resized to fill the window while preserving its aspect ratio, some image clipping may occur

By default, the image display will be set start as Uniform.

Users may navigate sequentially through the series of images by selecting the Next or Previous arrows at the sides of the screen or the scroll bar at the bottom of the screen. They may also view a different series within the set of images loaded to Scan Gallery using the drop down tab at the top of the screen. Additionally, the following voice commands are also supported:

* **Next:** Navigate to the next image in the series
* **Previous:** Navigate to the previous image in the series
* **Scroll Left:** Incrementally navigates to the previous image until the beginning of the series is reached, or until the “Stop Scrolling” command is used
* **Scroll Right:** Incrementally navigates to the next image until the end of the series is reached, or until the “Stop Scrolling” command is used
* **Stop Scrolling:** Stops a “Scroll Left” or “Scroll Right” command
* **Switch Series:** Allows the user to select a different series of DICOM images from the collection of all images loaded to Scan Gallery
* **Select Series:** Same effect as “Switch Series”

# Implementation

**[TECH] [DEVS]**

## Loaded DICOM Images

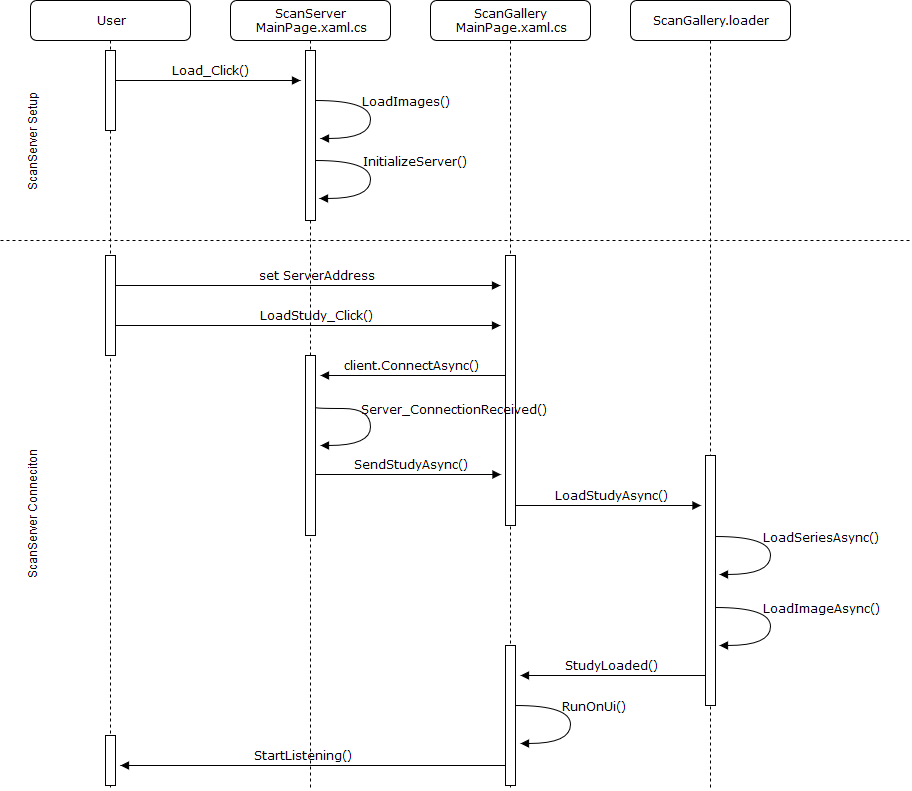
All DICOM format images used by this program must be stored on disk – on the HoloLens if using “Load Study From Local”, and on the device running Scan Server if using “Load Study From Server”. When loading images that are stored on the HoloLens device, an internet connection is not required, as all files accessed are stored locally. When using the Scan Server to load a study however, an internet connection is required. The DICOM images are transferred using a WebSocket (WS) connection, initiated by the Scan Server device. DICOM images cannot be accessed from an online source directly using Apollo Lens applications. Images must be saved to disk for use.

## Scan Server Setup

Images should be loaded to the Scan Server before attempting a connection from Scan Gallery. Upon launching Scan Server, the user will select a folder, and the Load\_Click() event will trigger, causing two asynchronous methods to be run, LoadImages() and Server\_ConnectionReceived(). LoadImages() handles the loading of each DICOM image that needs to be uploaded, and progress is displayed using a loading bar. Server\_ConnectionReceived() handles the sending of information from Scan Server to a Scan Gallery connection. Following Load\_Click(), Scan Server will be ready to deliver information as soon as it connects to an instance of a Scan Gallery.

## Scan Gallery – Scan Server Connection

Scan Gallery and Scan Server connect to one another using a UWP StreamSocket. The user launches Scan Gallery and specifies the ServerAddress for the device running Scan Server. Selecting “Load from Server” causes two asynchronous methods to run. One initiates the connection to Scan Server, and it will be notified when the connection has been established. In response, Scan Server sends the previously loaded study to Scan Gallery. The other method listens for a study to be received, and upon receiving it, runs several methods to determine what should be displayed to the user with regards to study, series, and image. An event triggers once all these complete, notifying Scan Gallery to update the UI. Finally, voice commands are activated for use with Scan Gallery with the StartListening() method. The process for setup and connection to Scan Server is outlined below in Figure 8.



**Figure 8** Loading images to Scan Gallery using Scan Server

NOTE: The Scan Gallery namespace defined in ScanGallery/MainPage.xaml.cs contains a DicomNetworking component, loader. In this diagram, it has been separated from the rest of Scan Gallery to clarify the sequence of events that must occur before a study is loaded.

## Video Streaming with WebRTC

Apollo Lens Client and Apollo Lens Source applications connect to one another for live video streaming using Web Real-Time Communication (RTC) protocol. Both begin by establishing a connection to a signaling server. It is recommended to setup this server on an AWS Elastic Beanstalk. Afterwards, one makes a request to establish a peer connection between the two, and they exchange network information so that they may communicate directly, without the need to relay information through the signaling server. Video is sent from one peer to another – it is never sent to the server.

## Connecting to the Signaling Server

Both applications will each connect to the server upon selecting “Connect to Server” from their start menu, and upon establishing this connection, will run Initialize(). This method readies the application for communication with the signaling server and sending or receiving media over the peer connection. There is no specific order in which the two applications should connect to the signaling server, but Apollo Lens Source should connect to the signaling server before “Connect to Source” is selected in Apollo Lens Client. The process for connecting both applications to the server is outlined below in Figure 9.

A screenshot of a social media post

Description automatically generated

**Figure 9** Connecting to the signaling server

## Establishing a Peer Connection

Once both applications are connected to the signaling server, they may begin establishing a private means of communication over a peer connection. Each application has a Conductor that controls how the application behaves and changes in response to events at runtime. In addition to input from the user, it handles communication relayed through the signaling server for establishing a peer connection. These Conductors contain two notable components used in this process – RTCPeerConnection peerConnection and IWebRtcSignaller signaler. The peerConnection component is used to communicate directly between the two peers. It contains information on the endpoints used to communicate and transfer video and audio tracks. The signaler communicates with the signaling server, and it is used in establishing the peer connection.

NOTE: The following section will use Client and Source to refer to Apollo Lens Client and Apollo Lens Source, respectively. Additionally, in the following figures, each application’s MainPage.xaml.cs is extended to Conductor, peerConnection, and signaller to better reflect the file structure used to implement this system.

To begin this process, the user selects “Connect to Source” in Client. This method triggers StartCall() in its conductor, and a peerConnection component is initialized. An offer message is created – this is a message requesting to establish a peer connection, and it contains network information on the requester. Then the local description is then set, which gives the peerConnection component details on the local end of the two-way peer communication being established. Finally, the offer is sent to the signaling server, and relayed to Source. Figure 10a shown below outlines the process by which Client initiates a connection with Source.

A close up of a map

Description automatically generated

**Figure 10a** Initiating connection between Client and Source

When Source receives the offer, it generates its own peerConnection component. It then selects an Internet Connectivity Establishment (ICE) candidate, a collection of protocols and routing which can be used for the peer connection. This ICE candidate is sent as a message to the Client, and the client adds it to its peerConnection. Following this, the Source sets its remote description for its peer connection, details on the remote end of the two-way peer communication being established, using the offer received. An answer message is then created – a response to the offer that contains network information on the responder. The local description is then set, and the answer is sent to the signaling server to be relayed to Client. Figure 10b shown below outlines the process by which Source responds to Client’s offer.

A close up of a map

Description automatically generated

A close up of a map

Description automatically generated

**Figure 10b** Confirming connection between Client and Source

When Client receives the answer, it generates its own ICE candidate, and sends it to the Source through the server. When Source receives the ICE candidate message, it adds the candidate to its peerConnection. Finally, the Client sets its remote description. Now both Source and Client have local and remote descriptions indicating the two endpoints for communication over a peer connection, and the server is no longer used. Video and audio tracks may be sent directly from one peer to another using their RTCPeerConnection components, peerConnection. Figure 10c shown below outlines the process by which Client and Source finish establishing a peer connection and begin transferring data through it.

A close up of a map

Description automatically generated

A close up of a map

Description automatically generated

**Figure 10c** Establishing peer connection between Client and Source

A complete view of Figure 10, detailing the process for setting up a peer connection following WebRTC protocol can be found at the end of this document.

NOTE: Since Client and Source both communicate privately, Source may disconnect from the server after communication has been established between the two peers. The stream will not be interrupted by this since a peer connection is being used.

# Development

**[DEVS]**

**Requirements**

Microsoft HoloLens Mixed Reality headset

Visual Studio 2019 or higher, Community edition is sufficient

Windows 10 UWP SDK version 10.0.17763.0 or higher

C# version 7.3 or higher, requires use of enum

## Build Instructions

For each UWP project component for ApolloLens applications, the following steps must be taken before building. First, ensure the appropriate version of the Windows 10 UWP SDK is used for each application.

Go to Project > {app name} Properties

To build the application from Visual Studio, Windows 10 UWP SDK must be version 10.0.17763.0 or higher. To specify this when building, use the following:

Project > {app name} Properties > Targeting

Set Target Version to Windows 10, version 1809 (10.0; Build 17763) or higher.

version 1809 / build number 17763

(Target Version 10.0.17763.0)

Applications should be built using x86 and their respective build target.

Building and deploying HoloLens applications requires the build target to be set to Device, if the HoloLens is connected via USB, or Remote Machine, if deployment is to be done wirelessly. This includes ApolloLensClient and ScanGallery. Regardless of whether built as Debug or Release, these apps will be accessible form the HoloLens’ start menu and can be launched without the use of Visual Studio.

When building desktop applications, the build target should be set to Local Machine. This includes ApolloLensSource and ScanServer. If it is build as Release, the application’s .exe will appear on the device’s start menu, and can be launched outside of Visual Studio.

While applications are designed with specific platforms in mind, any of the UWP applications may be built to either HoloLens or Desktop by specifying the desired build target.

NOTE: Because the application is UWP, it cannot be launched by trying to open the .exe found under ApolloLens/{app name}/bin/x84/Release/{app name}.exe

## Side Loading Applications

Applications can also be installed on Windows 10 computers that do not have Visual Studio 2019 installed. To do so, open the Apollo Lens solution in Visual Studio and navigate to the Solution Explorer. Right click the project to be side loaded and use the following steps:

Go to Store > Create App Packages.

Select distribution method: Sideloading

Verify the Output location for the App Package

Ensure the only Architecture checked is x86

Create the App Package

Generated packages are saved to {app name}/AppPackages/ by default. The folder name should follow the format {app name}\_{version number}\_Debug\_Test/. Copy this directory to the computer where the application will be sideloaded and find the Add-AppDevPackage.ps1. Right click it and select Run with PowerShell. If this is successful, the application will be installed to the computer and will be compatible with other instances of Apollo Lens applications

NOTE: You will need to verify the computer’s operating system is compatible with the desired application’s build version.

## Internal Compiler Errors

If a build fails due to an internal compiler error, perform the following steps:

1. Close all projects currently open in Visual Studio
2. Clear all NuGet caches  
   Go to Tools > Options > NuGet Package Manager > Package Management  
   And click Clear All NuGet Cache(s)
3. Close Visual Studio
4. Delete .vs/ from the solution’s root directory \*
5. Delete all instances of bin/ and obj/ in the each of the solution’s subdirectories \*
6. Re-open Visual Studio and load ApolloLens.sln
7. In the Solution Explorer, right click Solution ‘ApolloLens’ and select Restore NuGet Packages

Projects should now be able to built normally.

\* Alternatively, a python script is included to perform steps 4 and 5. As input it takes the path to the ApolloLens root directory from the current working directory. It will delete .vs/ and all instances of bin/ and obj/ in each subdirectory. Run this using python3 internal\_compiler\_error\_fix.py from a bash terminal.

<https://developercommunity.visualstudio.com/content/problem/133644/internal-compiler-error-5.html>

## Debugger Errors

Occasionally, Visual Studio may skip over all breakpoints and warn you that symbols have not been loaded. If this occurs, perform the following steps:

1. Right click your project in the Solution Explorer
2. Go to Debug > Start new instance

If symbols are still not loaded, try one of the following:

Verify Source Link Support is Activated

1. Go to Tools > Options > Debugging > General
2. Verify the checkbox for Enable Source Link Support is checked

Load Symbols Manually

1. Begin running the project in Debug mode
2. Open the Modules window (shortcut Ctrl + Alt + U)
3. Select all
4. Right click > Symbol Settings > Load all symbols

## HoloLens Setup

It is recommended to use an actual HoloLens device for development. Avoid using the Microsoft HoloLens emulator.

Connecting to the HoloLens can be done using the following link:

https://{hololens ip addr}

Or using a wired connection with the following link:

<http://127.0.0.1:10080>

A developer must also have proper UWP credentials. To reset the username and password, connect the HoloLens to a computer, and then follow the link below:

https://{hololens\_ip\_addr}/devicepair.htm

Be sure to replace {hololens ip addr} with your device’s IP Address.

## Server Configuration

A server is used for communication between ApolloLensClient and ApolloLensSource. The necessary files to run this server can be found under Server/. It is recommended to run this server using an AWS Elastic Beanstalk. Instructions to set this up are as follows:

1. Create a .zip of the contents of Server/
2. Register for AWS  
   <https://portal.aws.amazon.com/gp/aws/developer/registration/index.html>
3. Create an EC2 Elastic Beanstalk project  
   <https://aws.amazon.com/elasticbeanstalk/>
4. Under Application code select Upload your code
5. Upload the zipped contents of Server/
6. Click Create application
7. After a few minutes, the console will print “Successfully launched environment … ”  
   indicating the server is ready for use
8. Navigate to the project’s Dashboard and copy the URL link for server env
9. Open Library/Utilities/ServerConfig.cs and change AwsAdress to the new URL  
   NOTE: Be sure to change the URL scheme from https:// to ws://
10. Open ApolloLensClient/MainPage.xaml.cs and find OnNavigatedTo()  
    Ensure the ConnectToServerButton.Click event handler uses ServerConfig.AwsAddress
11. Open ApolloLensSource/MainPage.xaml.cs and find OnNavigatedTo()  
    Ensure the ConnectToServerButton.Click event handler uses ServerConfig.AwsAddress

This URL was used for the most recent development period using AWS:  
<https://us-east-2.console.aws.amazon.com/elasticbeanstalk/home?region=us-east-2#/applications>

# File Structure

**[DEVS]**

There are several directories for UWP applications, and thus follow a similar structure:

* ApolloLensBasic/
* ApolloLensClient/
* ApolloLensSource/
* ScanGallery/
* ScanServer/

Each of these contain two important pairs of identically named files, App and MainPage. Both App and MainPage contain a .xaml file and a .xaml.cs file. The .xaml file defines the structure of the UI seen within each app, while the .xaml.cs file manages logic and events that affect the UI at runtime.

App handles application-wide behavior, supplementing the default Application class. Aside from using a different namespace, all App.xaml.cs files are identical.

MainPage handles behaviors of specific features of the application. MainPage.xaml.cs files do differ greatly; their functionality will be detailed below.

ApolloLensBasic/**MainPage.xaml.cs**

Displays a video feed from a camera device connected to the system running this application. Used for testing/debugging WebRTC library.

ApolloLensClient/**MainPage.xaml.cs**

Allows the user to connect to an instance of ApolloLensSource for viewing a live video feed from the device.

ApolloLensSource/**MainPage.xaml.cs**

Allows the user to connect to an instance of ApolloLensClient for sending a live video feed from the current device. The specific device used for streaming and video quality must be determined before streaming.

ScanGallery/**MainPage.xaml.cs**

Allows the user to load a set of DICOM images for viewing, as well as adjust brightness and contrast. Images loaded may be stored on disk, or they may be loaded from a device running ScanServer.

### ScanServer/MainPage.xaml.cs

### Uses DicomNetworking to start a local server and send DICOM image files to a listener.

### Library/

Various interfaces used throughout all applications, most implementations for interfaces are done in this folder.

Library/**Imaging/**

Manages access to, and manipulation of, medical imaging artifacts. Images loaded are expected to be in a DICOM format.

Library/Imaging/**DicomNetworking.cs**

Defines a two sided protocol for sending and receiving a DICOM study over input and output streams. This has asynchronous methods for sending a study, as well as loading a study, a series of images, and an individual image.

Library/Imaging/**DicomParser.cs**

Takes converts images from DICOM format into ImageTransferObjects (see ImageTransferObject.cs for more information on this class). This conversion allows the user to manipulate the image using commands found in SmartBitmap.cs.

Also note that FoDicom was needed primarily to parse DICOM files to XML for data extraction, and to render DICOM image files to bitmaps. dicomFile.Dataset.WriteToXml() was used because the developer had more experience using Linq to XML than the FoDicom API.

Library/Imaging/**ImageCharacteristics.cs**

Contains classes for ImageCharacteristicsBase, Brightness, and Contrast. Base class uses a discretized iterator-like object for describing image brightness and contrast.

Library/Imaging/**ImageCollection.cs**

Definition of an interface for, and the implementation of, a collection of images with adjustable brightness and contrast. Has an event handler that fires upon completion of any method, causing the current image to change. Contains methods for creating and adding to image collections. Supports changing the image series, sequentially navigating through a given image series, and manipulating image brightness and contrast. Also has a reset method, allowing the user to restore an image to its original brightness and contrast.

Library/Imaging/**ImageTransferObject.cs**

Contains the definition for the ImageTransferObject class, the minimum amount of data needed to describe an image for transfer out of DicomParser or over DicomNetworking.

Library/Imaging/**SmartBitmap.cs**

This interface is what directly modifies an image’s bitmap when adjustments are made to brightness or contrast. It saves the original image, so changes are non-destructive. Also caches a modified image to avoid recalculating requests for the same brightness and contrast.

Library/**Signalling/**

Interfaces used for connection and communication in applications using WebRTC.

Library/Signalling/**IBasicSignaller.cs**

A basic signaler interface that supports sending and receiving strings.

Library/Signalling/**MessageProtocol.cs**

Interface for wrapping and unwrapping messages in JSON format. An enumeration is used as a message type signifier, allowing messages to be distinguished by an enum instead of a raw string.

Library/Signalling/**ProtocolSignaller.cs**

Interface that combines the concepts of a message protocol and a basic signaler. Wraps and unwraps messages using the protocol, as well as sends and receives them over the basic signaler.

Library/Signalling/**WebSocketSignaller.cs**

Implementation of the IBasicSignaller interface using websockets. Used for connection and communication involving the signaling server.

Library/**Utilities/**

Various helper files utilized throughout ApolloLens applications.

Library/Utilities/**Logger.cs**

Allows injecting any method to log a message. Can log to user interface, a file, a stringbuilder, etc. Just bind a method / lamda to WriteMessage, then use Log or LogLine to fire the event, from anywhere.

Library/Utilities/**ServerConfig.cs**

Contains addresses used to connect to signaling server for WebRTC applications.

Library/Utilities/**Util.cs**

Implementation of a Python style for loop, for internal use only.

Library/Utilities/**VisibilityExtensions.cs**

Simplified method for showing and hiding UI elements.

Library/**WebRtc/**

Used for conducting two-way communication using WebRTC protocol. Contains Interfaces which define Tasks and classes used by both Conductor.cs and WebRtcSignaller.cs (found under WebRtcImplNew/). For more information on this functionality, see the WebRTC section below.

Library/WebRtc/**IConductor.cs**

Defines an interface capable of accessing local media and conducting a configurable two-way audio and video call between two devices. This was implemented using WebRtc nuget package in other assemblies. The IConductor interface was used to wrap the Org.WebRtc namespace, with the intention of designing a more easily usable and configurable for a UWP example conductor.

Library/WebRtc/**IWebRtcSignaller.cs**

Defines the IUISignaller and IWebRtcSignaller interfaces. IUISignaller is used for two-way communication with plain text messages and running the task for shutdown. IWebRtcSignaller also supports these tasks in addition to handling messages for ICE Candidates, offers, and answers.

### WebRtcImplNew/

Contains the implementations of the interfaces defined in Library/WebRtc/.

WebRtcImplNew/**Conductor.cs**

The implementation of the conductor interface, used for controlling behaviors during communication via WebRTC protocol. Also contains methods for establishing a peer connection and event handlers for signaler.

WebRtcImplNew/**WebRtcSignaller.cs**

The implementation of the signaler interface, used for communication via WebRTC protocol. Received messages are interpreted based on message.Type. There are five different types of signaling messages that can be received, they are:

* **Offer:** Sent by the peer connection to establish a peer-to-peer connection
* **Answer:** Sent in response to an offer along the same signal channel
* **IceCandidate:** Contains information regarding an available method of communication  
  A pair of candidates will be chosen to allow the two peers to send media to one another
* **Plain:** Message containing only plain text, this is only used for testing purposes
* **Shutdown:** Notification to shutdown the connection to the peer

**WebRtcImplOld/**

Nearly identical to WebRtcImplNew/, differing only in which version of the WebRTC nuget package is used (New uses a newer package, whereas old uses an older package) All files contained follow the same structure as described for WebRtcImplNew/.

Server/**server.js**

Javascript file used to run the server required for linking WebRTC applications (namely ApolloLensClient and ApolloLensSource).

NOTE: All instances of Properties/AssemblyInfo.cs has been excluded from directories in this overview of the ApolloLens file structure, as each of these files strictly contain internal information (i.e. build version) and do not have a functional influence on the application.

# Appendix

**[TECH] [DEVS]**

## Video Streaming with WebRTC

WebRTC protocol for establishing a peer connection between ApolloLensClient and ApolloLensSource. StartCall() Initiates the process, occuring when the user select “Connect to Source” from ApolloLensClient. A signaling server is used to setup this connection, but is not used after connection is established. In Figure 10 below, each application’s MainPage.xaml.cs was extended to Conductor, peerConnection, and signaller to better represent the file structure used to implement this system.

A close up of a map

Description automatically generated

**Figure 10** WebRTC Peer connection setup for Apollo Lens Client and Apollo Lens Source

Application specific state chart diagrams outlining the process explained under Usage are listed below. These explore each application’s functionality in greater detail.

## Scan Gallery

A screenshot of a cell phone

Description automatically generated

**Figure 11** Usage of the Scan Gallery application

## Scan Server

A screenshot of a cell phone

Description automatically generated

**Figure 12** Usage of the Scan Server application

## Apollo Lens Client

A screenshot of a cell phone

Description automatically generated

**Figure 13** Usage of the Apollo Lens Client application

## Apollo Lens Source

A screenshot of a cell phone

Description automatically generated

**Figure 14** Usage of the Apollo Lens Source application

## References

**[1]** Ailenis (2019) *Microsoft Hololens, Virtual Reality, Augmented Reality, Eyewear, White PNG*. [Online]. Available: *https://www.kisspng.com/png-microsoft-hololens-virtual-reality-augmented-reali-3404403/*. [July 8, 2019].