# SAFR® Windows SDK Documentation

Documentation Version = 3.048

Publish Date = August 19, 2022



**Note:** Documentation pertaining to the macOS platform is no longer being actively maintained.

# Contents

# 1 SAFR SDK Overview

## 1.1 Purpose

The SAFR SDK is a set of shared libraries that together implement an object tracking mechanism, face recognition and event reporting. The object tracker is used to locate and track different types of objects (for example, human faces and badges) in a video file or live video stream. Face recognition can learn new identities or match existing identities. The event reporter can notify your application of detection and recognition events. Object detection, face recognition, tracking and event reporting are performed in real time.

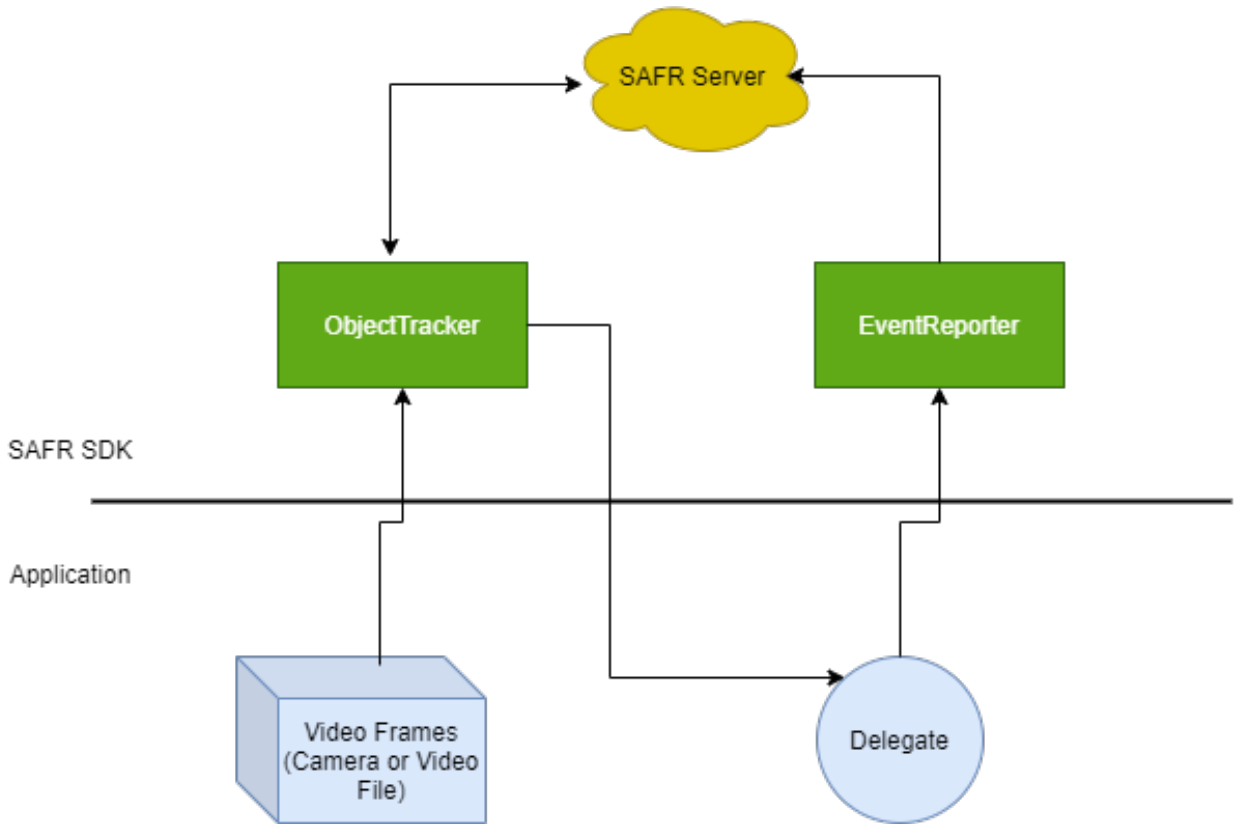
The SDK features a C API that exposes a set of ArgusKit objects. All SDK-defined functions and type definitions are prefixed with the `ARK` namespace qualifier. The SDK is distributed as a `tar.gz` file containing all necessary headers, documentation, and shared libraries. It also includes samples showing how the SDK can be used from your own application, and how the SDK components (shared libraries and auxiliary files) must be packaged with your application in order to ensure ArgusKit works correctly with your application.

While the SDK API design and architecture is the same across all supported platforms, the implementation language of the API may be different. For example, the API may be a C API for some platforms, while it is a Java or C# API for other platforms. Also, the packaging of the SDK is different for each platform to accommodate the technical requirements and limitations of the platform.

## 1.2 SDK Organization

The SAFR SDK defines a number of objects that an application uses to track faces or badges. The application creates these objects, configures them, and then interacts with them. Once the application no longer needs a SAFR SDK object, it is expected to release (*free*) the SAFR SDK objects. All SAFR SDK objects use reference counting to manage the lifetime of the objects.

The following diagram shows the two most important SAFR SDK objects: the `ObjectTracker` and the `EventReporter`, and how an application interacts with them:




The application receives video frames from a camera or a video file and hands them off to an ObjectTracker. The ObjectTracker processes the video frames internally to detect and track objects. The ObjectTacker sends the video frames to a SAFR Server (whether in the cloud or installed on-premises) for recognition. The ObjectTracker continuously informs the application about the current state of the ObjectTracker by invoking the ObjectTracker delegate. This delegate is provided by the application and receives information about objects found in the video input stream, objects that have disappeared, and objects that have changed their state. For example, an object may have been detected in one video frame, and it may have been recognized as a specific person in a later video frame. The ObjectTracker informs the application about this change of state.

An application is free to use the information about a tracked object any way it likes. It could display this information in a video overlay in its UI or it could process the information to generate statistical information or events. The SAFR SDK comes with an EventReporter able to process a stream of tracked objects to detect certain kinds of events that are then automatically reported to a SAFR Server.

**Note**: For SAFR SDK system requirements, go to SAFR.com.

# 2 Windows SDK Installation

Ensure that you have C# 7.0 and the .Net 4.6.2 or newer framework installed before you begin working with the SDK. Microsoft Visual Studio 2017 or newer is strongly recommended.

Also note that the SDK supports 64-bit x86 CPUs only. Make sure you select the x64 configuration in the configuration pop-up menu. You may have to create a x64 configuration in the configuration manager if the project doesn't already contain one.

## 2.1 SDK Installation

Install the SAFR SDK and create a Visual Studio project by doing the following:

1. The SDK is distributed as a `.zip` file containing documentation, samples, and a NuGet package that makes it easy to add the SDK to your project.
   - Create a SAFR Account and download the SAFR Windows SDK from the Download Portal.
   - Unzip the SDK.
2. Create a project in Visual Studio.
3. Change project build configuration from **Any CPU** to **x64**. Applications that want to use the SAFR SDK have to be built for the x64 CPU architecture. Neither "Any CPU" nor "x86" are supported.
   - Open the Build Configuration Manager (Build > Configuration Manager).
   - In the **Platform** list box for your project, select **<New...>**.
   - Select **x64** for the platform and **Any CPU** for the **Copy Settings From** listbox.
   - Click **OK**, and select the **x64 Platform** the Solution Platform Build listbox.
4. Install the SAFR SDK NuGet package:
   - In Visual Studio, open the Package Manager console (Tools > NuGet Package Manager).
   - Run `Install-Package C:\ArgusSDK\nu-package\RealNetworks.Argus.SDK.<SDK Version Number>.nupkg` to install the standard SDK package. To install the Lite SDK package, replace `.Argus.` with `.Argus.Lite.`
5. Update the post build step to copy face detector dependencies to the output folder.
   - Copy the following code for the standard SDK package:

     ```
     copy $(SolutionDir)packages\RealNetworks.Argus.SDK.<SDK Version
        number>\content\bin\$(PlatformName)\$(ConfigurationName) $(TargetDir)
     ```

     For the Lite SDK package, replace `.Argus.` with `.Argus.Lite.`

# 3 Use the Windows SDK

Use the objects provided by the SAFR SDK by importing the `RealNetworks.ArgusKit` assembly into your C# source file. The following code example shows how to do that and the basic steps for setting up an object tracker:

```
using RealNetworks.ArgusKit;
using System.Windows;

namespace Sample
{
    public partial class MainWindow : Window
    {
        // The object tracker
        private ObjectTracker ObjectTracker = null;

        public MainWindow()
        {
            InitializeComponent();

            // Select the cloud environment
            var cloud = CloudEnvironment.Prod;

            // Define the cloud account
            var account = new CloudAccount("user", "pwd");

            // Create the object tracker configuration
            var config = new ObjectTrackerConfiguration();
            config.Recognizer.Cloud = cloud;
            config.Recognizer.Account = account;
            ...

            // Create the object tracker
            ObjectTracker = new ObjectTracker(config);
            ObjectTracker.DidTrack += OnDidTrack;
        }

        // We assume that this function is invoked from a video player or
           a camera
        // that outputs a BitmapSource for every video frame. We wrap the
           video
        // frame up and pass it to the object tracker.
        private void
           OnNewVideoFrameAvailable(RealNetworks.CameraKit.VideoFrame
           frame)
        {
            ObjectTracker?.TrackObjects(frame);
        }

        // Invoked by the object tracker every time the tracking state
           has changed
        private void OnDidTrack(object sender,
           ObjectTrackerDidTrackEventArgs e)
        {
            TrackingResult result = e.Result;
```

```
            // Process the tracking result
            ...
        }
    }
}
```

**Note**: The only supported solution platform is "x64". Other configurations, including "Any CPU" and "x86", are not supported.

# 4 Utility Objects

This section describes utility objects used with many of the other SDK functions.

## 4.1 Recognizing Faces

You should set the cloud environment and cloud account information in the object tracker's recognizer object to enable face recognition. You also need to set the directory in the account object. The easiest path towards a working object tracker configuration object is to use the **ObjectTrackerConfiguration.ConfigurationForPreset()** function with the **ConfigurationPreset.Recognition** argument. Invoke this function to get a pre-filled configuration suitable for face recognition and then set the cloud environment and cloud account information before you pass the configuration object to the object tracker.

The object tracker will then return tracked object instances with a detected object type equal to Face.

## 4.2 Detecting Persons

After you get a basic object configuration object set up as described in the Recognizing Faces section above, you can enable the detection of persons by setting the **EnableRecognizedObjectDetector** property in the object tracker's detector configuration object to `true`. This will cause the object tracker to return tracked objects with a detected object type equal to **RecognizedObject**.

## 4.3 CloudAccount

The **CloudAccount** object stores the account name and password for the cloud account used for cloud based functionality, such as face recognition.

## 4.4 CloudEnvironment

The **CloudEnvironment** object encapsulates the cloud service the ArgusKit should connect to in order to execute cloud-based functions, such as face recognition.

# 5 Tracking Object in a Video

This section describes the SDK objects which you use to detect and recognize objects.

The most important SDK object in this category is the object tracker. The object tracker allows you to detect and recognize objects of different types, get information about them and it takes care of tracking their location and identity over time.

## 5.1 ObjectTrackerConfiguration

An **ObjectTrackerConfiguration** object stores the configuration information for an object tracker, the object detector and the face recognizer. The configuration object is initialized with default values that allow the tracker to detect and recognize faces. You may also instantiate a configuration object based on one of a number of predefined configuration presets. A preset encapsulates all the configuration information needed to use the object tracker for a specific type of task (e.g. to recognize faces or to recognize and learn faces).

After you have instantiated a configuration object you should set the cloud account, cloud environment, and the face recognizer directory name. This is the minimum required information that you need to provide to allow the object tracker to successfully detect and recognize faces.

Main functionality switches:

```
bool ObjectTrackerConfiguration.Detector.Face.Enabled // Detect faces
bool ObjectTrackerConfiguration.Detector.Face.LivenessEnabled // Detect
   RGB liveness
bool ObjectTrackerConfiguration.Detector.Badge.Enabled // Detect badges
bool ObjectTrackerConfiguration.Detector.RecognizedObject.Enabled //
   Detect persons
bool ObjectTrackerConfiguration.Recognizer.DetectIdentity // Recognize
   person identity
bool ObjectTrackerConfiguration.Recognizer.DetectOcclusion // Detect
   occlusion
bool ObjectTrackerConfiguration.Recognizer.DetectMask // Detect mask
bool ObjectTrackerConfiguration.Recognizer.DetectGender // Detect gender
bool ObjectTrackerConfiguration.Recognizer.DetectAge // Detect age
bool ObjectTrackerConfiguration.Recognizer.DetectSentiment // Detect
   sentiment
bool ObjectTrackerConfiguration.Recognizer.DetectLiveness // Detect 3D
   Liveness (when using Intel RealSense camera)
```

## 5.2 ObjectTracker

The **ObjectTracker** is the heart of ArgusKit. It receives a stream of video frames which it analysis to find objects. It tracks found objects as long as they remain visible in the video stream. Object detection, recognition, and the tracking are executed in real time. An object tracker is connected to a delegate which is defined by a set of C# event handlers which you set on the object tracker after you have instantiated it and before you call **beginTracking()** on it. The **ObjectTracker** informs its event handlers for every frame about the current state of the objects which it is tracking. The event handlers may then use the tracked object APIs to learn what kind of objects the tracker found and what their current spatial location, size, and state is.

## 5.3 TrackingResult

A **TrackingResult** object contains a snapshot of the current state of the object tracker. The object tracker delivers a new tracking result at every frame boundary. The tracking result contains a list of tracked objects that have appeared in the current frame, that have disappeared and a list of tracked objects that have

changed their current state in the current frame. For example, if a tracked object has been detected in a previous frame and the object tracker has now been able to recognize the tracked object as a specific identity, then the tracked object is included in the list of updated tracked objects.

## 5.4 TrackedObject

A **TrackedObject** represents a single and unique instance of an object that the object tracker has been able to detect in the input video stream and which it is actively tracking. A tracked object has a type that indicates whether it is a badge or the face of a person. A tracked object also has an axis-aligned bounding box which tells you where in the input video frame the tracked object can be found and what its size is.

Once a tracked object has been recognized as a specific person it includes additional information about that person. At a minimum it includes a unique person ID. But it may include additional information like the person's name if a name was previously assigned to that person ID. You use the **applyChange()** API on the object tracker to assign a name to a recognized person.

Main TrackedObject properties:

```
DetectedObjectType TrackedObject.ObjectType // Badge / Face /
   RecognizedObject
string TrackedObject.Person.PersonId // Recognized person ID
string TrackedObject.Person.Name // Recognized person name
double? TrackedObject.Person.Sentiment // sentiment: -1 to 1
double? TrackedObject.Person.Occlusion // occlusion: 0 to 1
double? TrackedObject.Person.CenterPoseQuality // center pose quality: 0
   to 1
double? TrackedObject.Person.SharpnessQuality // sharpness quality: 0 to 1
double? TrackedObject.Person.ContrastQuality // contrast quality: 0 to 1
Gender TrackedObject.Person.Gender // Male / Female / Unknown
Age? TrackedObject.Person.Age
PersonIdClass? TrackedObject.Person.IdClass // Unknown / Unidentified /
   Stranger / NoConcern / Concern / Threat
double? TrackedObject.Person.SimilarityScore // similarity score: 0 to 1
double? TrackedObject.Person.Liveness // liveness score
bool? TrackedObject.Person.LivenessConfirmed // liveness detected
bool? TrackedObject.Person.Mask // mask detected
double? TrackedObject.Person.MaskConfidence // mask detection confidence
```

## 5.5 TrackedBadge

A **TrackedBadge** is a tracked object which encodes an integer number in the form of a special pattern.

## 5.6 TrackedFace

A **TrackedFace** represents the face of a person. If the object tracker is able to recognize the face as belonging to a specific person then the tracked face provides a reference to the corresponding person object. The person object holds information about that person such as the unique person ID.

## 5.7 Person

A **Person** object provides information about a person that has been registered with the ArgusKit face recognition service. Each person has a unique identifier, called a **person ID**. You may assign a name and a set of tags to a person with the help of a **TrackedFaceChange** object and the object tracker **applyChange()** object tracker function.

## 5.8 TrackedFaceChange

A person change object stores attributes that should be applied to the person record on file in the ArgusKit face recognition service. This object allows you to change a person's name, tags, age, or gender information.

## 5.9 VideoFrame

A **VideoFrame** object encapsulates a single decoded video frame which should be passed to an object tracker instance.

# 6 Analyzing Images

This pages describes the image analyzer object which you use to find faces, persons, badges, etc in an image.

## 6.1 ImageAnalyzerConfiguration

An image analyzer configuration object stores the configuration information for an image analyzer, the object detector, face recognizer, badge detector, and shape detector. The configuration object is initialized with default values allowing the image analyzer to detect and recognize faces. You may also instantiate a configuration object based on one of a number of predefined configuration presets. A preset encapsulates all the configuration information needed to use the image analyzer for a specific type of task (e.g. to recognize faces or to recognize and learn faces).

After you have instantiated a configuration object, set the cloud account, cloud environment, and the face recognizer directory name. This is the minimum required information you need to provide to allow the object tracker to successfully detect and recognize faces.

If you want to detect faces only without recognizing them, turn off the face recognizer stage by setting the **Recognizer.Enabled** property to `false`.

## 6.2 ImageAnalyzer

The image analyzer processes images in order to find objects like shapes, badges, and faces. Object detection and recognition are executed in real time. An image analyzer is connected to a delegate defined by a set of C# event functions. The image analyzer informs its delegate when analysis is completed. The delegate can then use the image analyzer APIs to learn what kind of objects the image analyzer found and what their current spatial location, size, and state are.

## 6.3 ImageAnalysisResult

An image analysis result object contains a snapshot of the what was detected in an image. The image analyzer result contains a list of objects that have appeared in the current image.

## 6.4 DetectedObject

An analyzed object represents a single and unique instance of an object the image analyzer has been able to detect in the image. An analyzed object has an axis-aligned bounding box that tells you where in the input video frame the tracked object can be found and what its size is.

## 6.5 DetectedFace

An analyzed face represents a human face. An analyzed object may be linked to a person. If the image analyzer is able to recognize the face as belonging to a specific person, the tracked face provides a reference to the corresponding person object.

## 6.6 DetectedBadge

An analyzed badge represents a Rhino tags style badge.

## 6.7 DetectedRecognizedObject

An analyzed shape represents an object like a car or a person.

# 7 Event Reporting

This section describes the SDK objects for use in generating events from object tracker results and to posting them to the Event Server in the SAFR Cloud.

## 7.1 ObjectEventDataStore

This object acts as a local store for events. It is needed by the **PeopleIndexer** object that is the main interface to the event reporting system.

## 7.2 ObjectEventLog

**ObjectEventLog** records object tracker results. You should invoke the **upate()** method with the current object tracker result every time the object tracker invokes your **didTrack()** event handler.

## 7.3 PeopleIndexerConfiguration

The **PeopleIndexerConfiguration** object stores all relevant configuration information for the event reporting system. Create an instance of this object to get a default configuration, and then set the cloud account and environment information to enable reporting to the SAFR Cloud Event Server.

## 7.4 PeopleIndexer

The **PeopleIndexer** object ties together the object event store and the object event log. Pass your configuration object when you create an instance of the people indexer. The people indexer coordinates the event reporting mechanism.

# 8 Camera Objects

The SAFR SDK comes with objects that make it easy to work with USB, IP, and ONVIF cameras. It also includes a generic video player which can play back video as well as stream HTTP and RTSP videos over the network. It also includes a video view which makes it easy to display a video stream on screen.

## 8.1 Camera

Camera is an abstract base class which represents a camera device and its properties. There are concrete subclasses for specific types of cameras: USB cameras, IP cameras, and ONVIF cameras. A USB camera is a camera that is directly connected to the computer and discovered by the operating system. An IP camera is a camera which makes a video stream available which can be connected to via a public (and usually unsecured) HTTP or RTSP URL. An ONVIF camera is a camera which implements the ONVIF camera discovery and control standard. The **CameraManager** uses the ONVIF protocol to discover ONVIF cameras and the **OnvifCamera** class uses the ONVIF protocol to acquire information about the camera and to initiate a video stream.

You acquire camera objects by asking the **CameraManager** for them. The **CameraManager** knows how to discover and manage cameras.

A camera has properties such as its name and a set of available video formats. Some of those properties may be configurable for some types of cameras and read-only for other types. For example, the name of a camera is configurable for IP cameras but read-only for ONVIF cameras because the name of the camera in this case is only configurable via the web-UI provided by the camera software.

Every camera has a globally unique identifier. This identifier is guaranteed to remain stable and may be persisted. You can use this identifier to associate your own configuration information with a camera.

You create a video stream from a camera by asking the camera for a capture session via the **CreateCaptureSession()** function. This function takes a video profile which specifies the video resolution and frames per second that the video stream should have. Note that some cameras may allow you to create more than one capture session for a given video profile and some may allow only one active capture session per video profile.

Use the **DefaultProfile** property to get the default video profile of a camera. The default profile is usually a 1080p profile or close to it. Use the **Profiles** property to get the full list of available profiles. You can iterate through this list of profiles to find one that most closely matches what you're looking for. You can then create a capture session with this profile.

Once you've created a capture session you should register event handlers on the capture session object to receive video frames and errors. You can then start the capture session by setting its **Capturing** property to `true`. You can pause/stop capturing at any time by setting **Capturing** to `false`.

### 8.1.1 IPCamera

An IP camera object represents a type of camera which allows you to stream a video from a fixed and publicly accessible URL. This kind of cameras can't be automatically discovered. You must instead explicitly create this kind of camera and it to the camera manager by calling the camera manager **AddCamera()** function with the correct video stream URL.

### 8.1.2 OnvifCamera

An ONVIF camera object represents a camera which implements the ONVIF camera discovery and managed standard.

An ONVIF camera must be authenticated before you're allowed to access the available video profile information and before you're able to create a capture session.

You authenticate an ONVI camera after it has been discovered by the camera manager and before you create a capture session. Invoke the **Authenticate()** function on the camera object with the correct camera credentials. The camera will start the authentication process and invoke the callback that you provided to the **Authenticate()** function once authentication is complete. The callback is invoked with an appropriate error if authentication was unsuccessful.

### 8.1.3 USBCamera

A USBCamera camera object represents a camera which is directly connected to the computer and discovered by the operating system. Examples of such cameras are cameras which are connected via USB.

## 8.2 CaptureSession

A video capture session is created by invoking the **CreateCaptureSession()** function on a camera object. It represents a video stream with a specific video resolution and frames-per-second setting. A capture session is created in the `paused` state. Set its **Capturing** property to `true` to start the capture session and set it to `false` to pause or stop the capture session.

The **Accelerator** argument of the **CreateCaptureSession()** function specifies the object providing GPU acceleration of video decoding as follows:

| Element | Description |
|---|---|
| null | No GPU acceleration. |
| `RealNetworks.CameraKit.Accelerator.Shared` | Use the GPU representing primary display. |
| `RealNetworks.CameraKit.Accelerator.Create(RealNetworks.CameraKit.Adapter)` | Use a specific GPU Kit Adapter by **Adapter**. (**RealNetworks.CameraKit.Adapter.All** lists all available adapters). |

You should register a frame event handler before starting the capture session. Do this by assigning your event handler function to the **DidDecode** event property. The capture session invokes the **DidDecode** event for every decoded video frame. You can get the video frame from the event argument.

You should register an event handler to the **DidDetectError** event handler property to be notified of capture session errors.

## 8.3 CameraManager

The camera manager is responsible for the discovery of cameras and it allows you easy access to the default camera for a camera position.

You trigger camera discovery by invoking the **TriggerDiscovery** function. Camera discovery operates asynchronously. You should invoke this function at least once at application startup and you may invoke it at additional times. Generically this function should be invoked any time that the list of available cameras may change. (e.g. every time your application becomes the active application)

You should register an event handler with the **DidEndDiscovery** property. The camera manager will invoke this event handler every time a camera discovery session has completed. This event handler is invoked with the list of all the cameras that are known to the camera manager at this moment in time. So this list includes both newly and previously discovered cameras.

The camera manager allows you to look up a camera by its unqiue identifier and it allows you to get the default camera for a camera position. A camera position indicates where a camera is located relative to the user. For example there is a camera position for the "front camera" and another one for the "back camera". A front camera is a camera which typically faces the user while a back camera is a camera which faces away from the user.

## 8.4 VideoPlayer

**VideoPlayer** allows you to play back a video file or a HTTP/RTSP video stream.

**Note**: The video player supports video streams only; it does not support audio streams.

Do the following:

- Create an instance of a video player, and then register an event handler with the **DidDecode** event property to receive the decoded video frames.
- Start playback by setting the **Paused** property to `false`. Pause playback at any time by setting the **Paused** property to `true`.
- Stop playback in preparation of disposing the video player by calling **Stop()**.
- Stop playback and dispose the player at the same time by calling **Dispose()**.

The video player enforces the video clock if the URL points to a video file, but it does not enforce the video clock if the URL points to a HTTP or RTSP video stream. In this case, the video player decodes video frames as fast as they arrive from the camera. It does this to minimize the video playback latency.

## 8.5 VideoView

A video view displays the video frames from a capture session or video player on the screen. The video timing is provided by the capture session or video player.

**Note**: The video view itself does not impose video timing. It immediately displays a video frame when you set it on the view.

# 9 Face Blur App

This is a console application used for blurring faces in images from a specified source folder. The app finds faces in images if they exist and blurs them, saving the resulting images to a specified destination folder.

Below are all the arguments you can use with the app. If an argument isn't passed to the app, then that argument's default value is used.

| Argument | Description |
|---|---|
| -src [path] | [Required] Source directory. For example, `C:\Users\user\Pictures\maroon.jpg` |
| -dst [path] | [Required] Destination directory. For example, `C:\Users\user\Pictures\maroonblur.jpg` |
| -blur [integer] | Blur factor in number of pixels. Default is `10` |
| -shape [string] | Options are `"oval"` and `"rect"`. Default is `"oval"` |
| -scaleImageToYPixels [integer] | Input image-maximum resolution. Bigger images are scaled down to this resolution; the default is `720`. |
| -minSearchedFacePixels [integer] | Minimal size of searched face. Default is `80`. |
| -minRequiredFacePixels [integer] | Minimal required size of detected face. Default is `0`. |
| -v | Version number, formatted as follows: `Product version: <version number>`. For example, `Product version: 1.0.0.0` |

For example, this command produced the image below:

```
blurfaces -src C:\Users\user\Pictures\maroon.jpg -dst
   C:\Users\user\Pictures\maroonblur.jpg -blur 5
```

**Note**: For choosing "oval" blur shape pixel size is scaled. Detected face height is divided by specified blur parameter so desired face area is masked with a grid of pixels of corresponding size being colored as center

pixel in that area.

## 9.1 Technical Notes

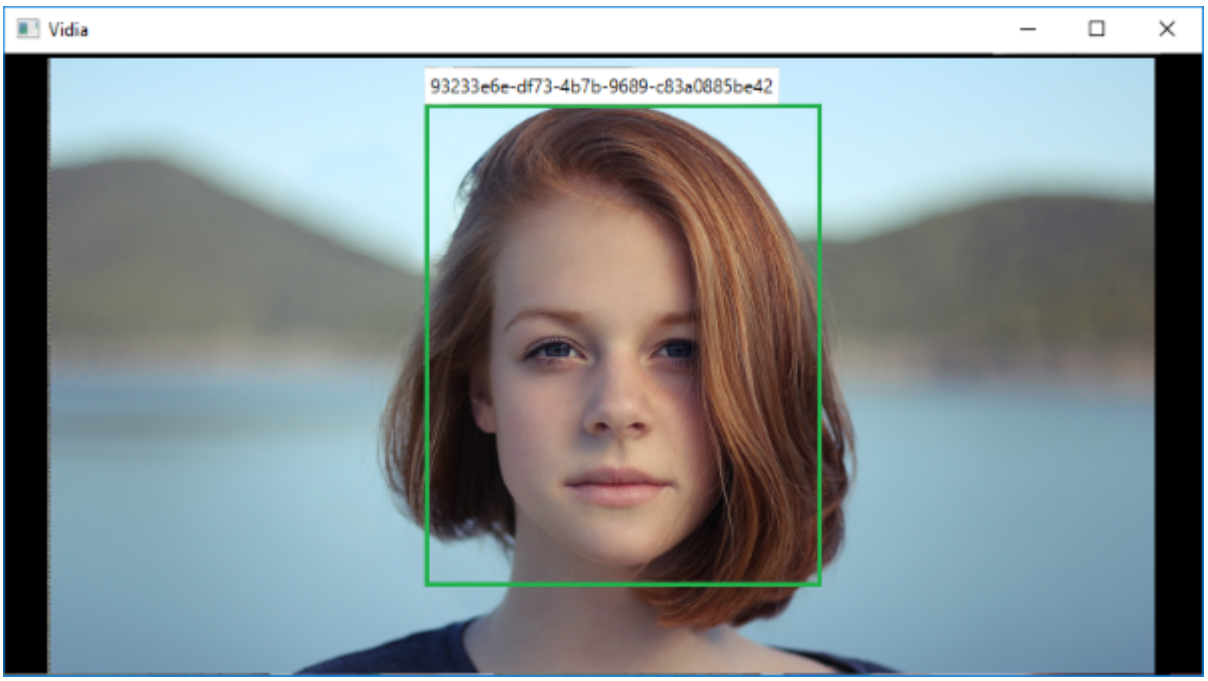
To compile the code and run the app, do the following:

1. Download SAFR Windows SDK from the Download Portal.
2. Install the SAFR SDK NuGet package. In the package manager console:
    1. Run `Install-Package C:\ArgusSDK\nu-package\RealNetworks.Argus.SDK.<SDK Version Number>.nupkg`
    2. Set the Solution Platforms selector option to **x64**. The **Any CPU** option is not supported.

## 9.2 Vidia App

This app demonstrates how to do the following:

- Use the CameraKit to find a camera and capture frames from it.
- Use the ArgusKit ObjectTracker to detect, recognize, and track objects in a video stream.
- Use the ArgusKit event reporting facilities to generate events and report them to the cloud.




Do the following to compile the code and run the app:

1. Download the SAFR Windows SDK from the Download Portal.
2. Right-click the **Vidia** project, and select **Manage NuGet Packages** from the menu.
3. Point the NuGet package manager to the directory containing the downloaded SAFR SDK.
4. Select the SAFR SDK, and click **Install**.
5. Set the Solution Platforms selector to **x64**. The **Any CPU** option is not supported.
6. Replace USER*âĂĚ*âĂĚ*and**PWD* with your SAFR Account credentials.
7. Compile and run the project.

# 10 Windows WPF App

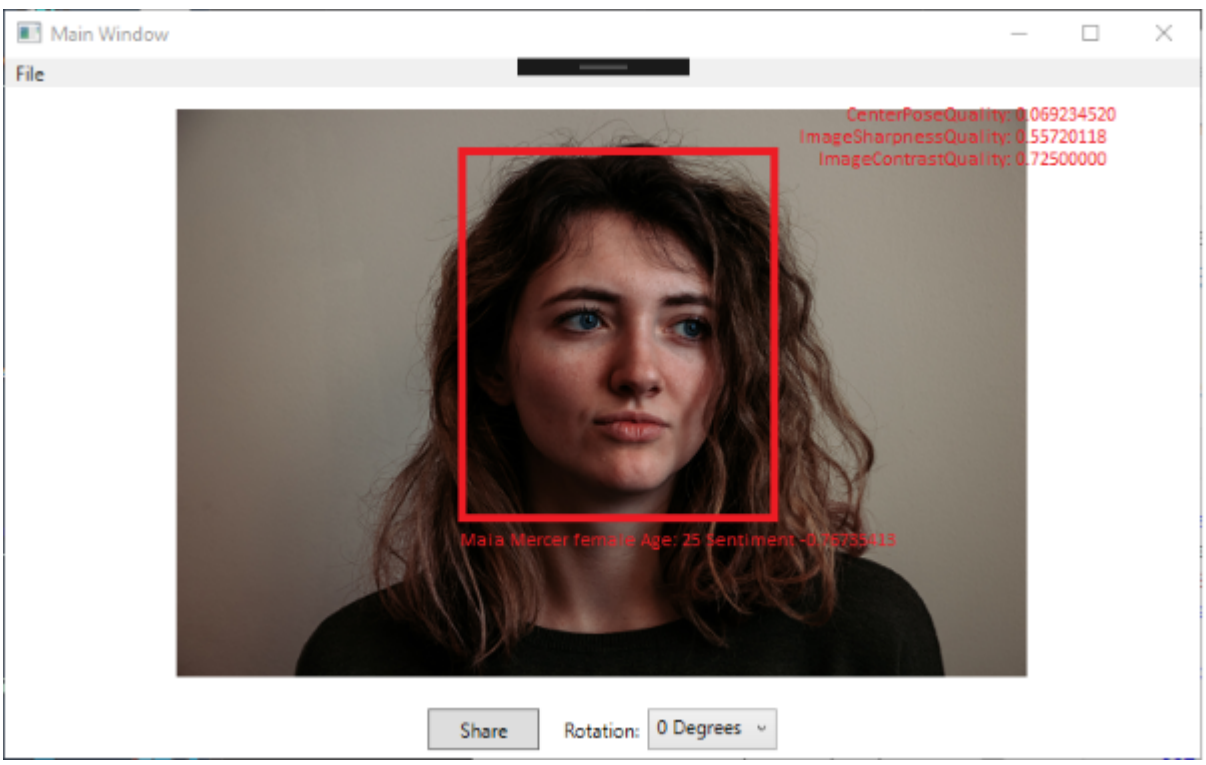
The Windows Presentation Foundation (WPF) sample app provides a wide range of features to explore. It demonstrates how to do the following:

- Camera video capture.
- Face detection.
- Object tracking.
- Registering new faces in the directory.
- Face image quality metrics.
- Recognition of identification, age, gender, and sentiment.
- Detection and recognition configuration.
- Connecting to your SAFR Cloud Account.
- Camera rotation.

## 10.1 Main Window View

The Main Window view has the following features:

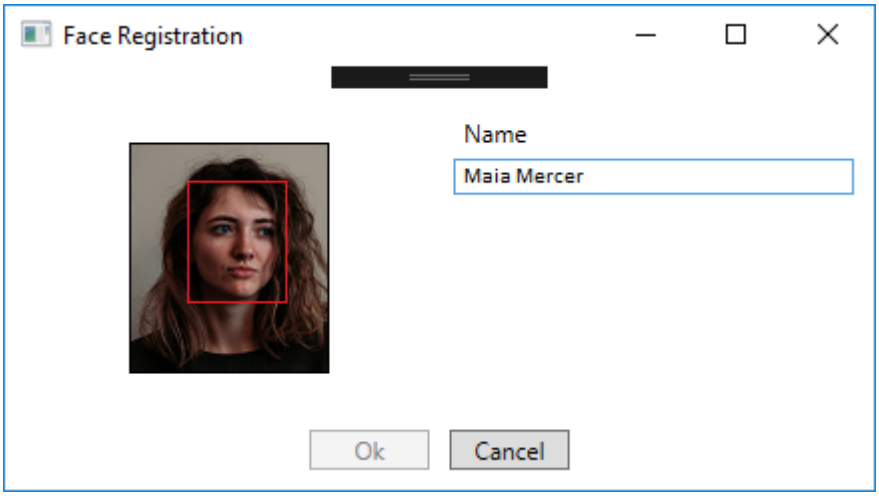
- Detects and tracks faces in video from an attached camera.
- Face quality metrics are shown in the top right.
- Available recognition information is shown under the face.
- The share button allows saving of the detected face to a file, with the name of the person if available.
- The camera feed can be rotated using the rotation control at the bottom of the view.




## 10.2 Face Registration View

Names can be registered with detected faces.

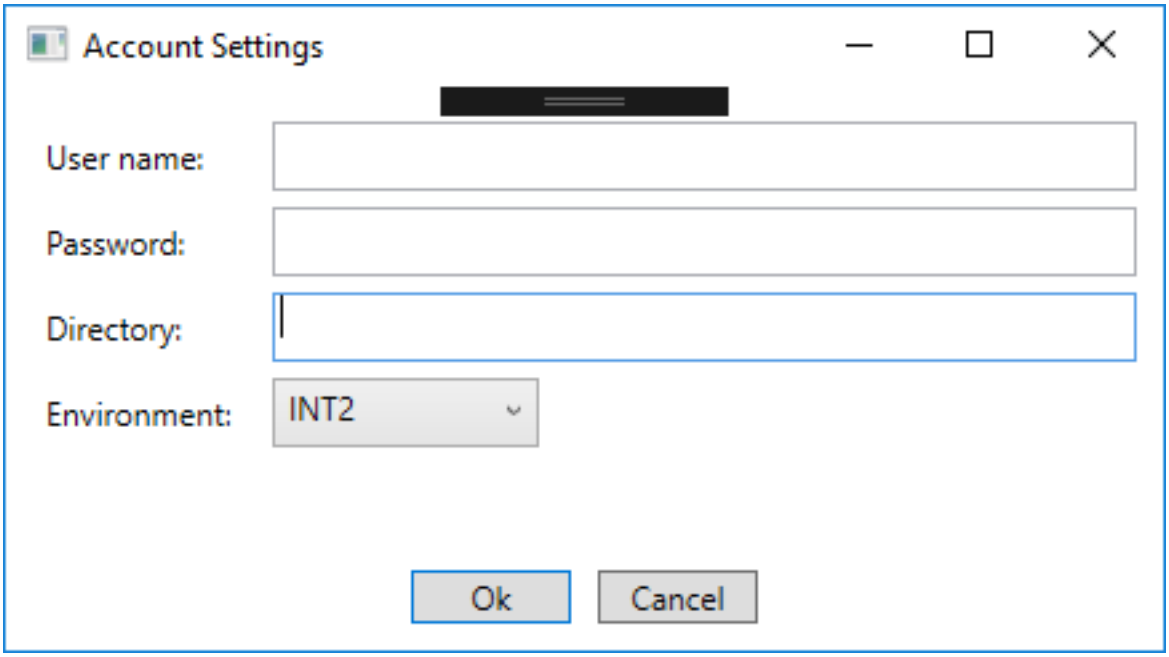
**Note**: Valid account credentials entered in Account Settings are required to use face registration.




## 10.3 Account Settings view

To enable identification, enter your SAFR Cloud account information.

Cloud accounts can be created here: https://safr.real.com/signup?sector=enterprise&accountType=local

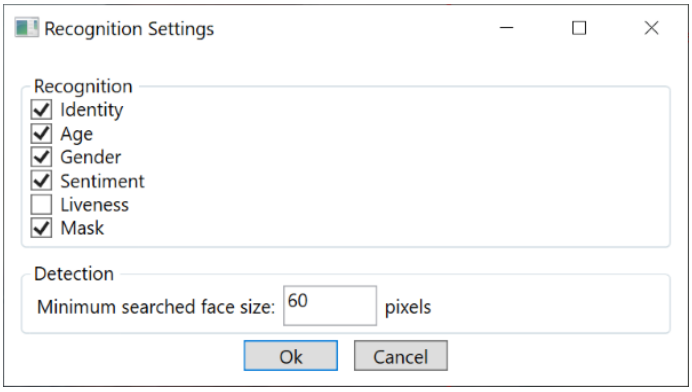



## 10.4 Recognition Settings View

In this view, you can enable or disable different types of recognition and detection parameters.

Under **Detection**, **Minimum Searched Face Size** defines the minimum face size that can be detected. A searched size of 80 pixels, for example, can manage to detect faces as small as 60x60 but with lower certainty. Lowering this number enables SAFR to detect much smaller faces but also greatly increases CPU usage.

**Note**: This parameter does not impact face recognition accuracy.




## 10.5 Shared Image

In the Main Window view, click **Share** to save the currently detected face and name. Below is an example of a shared image:


## 10.6 Technical Notes

To compile the code and run the app, do the following:

1. Download the SAFR Windows SDK from the Download Portal.
2. Install the SAFR SDK NuGet package.
3. In the package manager console, do the following:
    1. Run `Install-Package C:\ArgusSDK\nu-package\RealNetworks.Argus.SDK.<SDK Version Number>.nupkg`
    2. Set the Solution Platforms selector to **x64**. The **Any CPU** option is not supported.

This sample has been built using the Model-View-ViewModel (MVVM) design pattern: https://blogs.msdn.microsoft.com/msgulfcommunity/2013/03/13/understanding-the-basics-of-mvvm-design-pattern/

*ApplicationModel.cs* contains all the code using the SAFR SDK.

# 11 Imagina App

This sample app demonstrates how to use the **ImageAnalyzer** class in the **ArgusKit** namespace to detect and recognize objects in an image.

To compile and run this app, do the following:

1. Download the SAFR Windows SDK from the Download Portal.
2. Right-click the **Imagina** project, and select **Manage NuGet Packages** from menu.
3. Point the NuGet package manager to the directory containing the downloaded SAFR SDK.
4. Select the SAFR SDK, and click **Install**.
5. Set the **Solution Platforms** selector to **x64**. The **Any CPU** option is not supported.
6. Replace $USER and $PWD with your SAFR Account credentials.
7. Compile and run the project.