

Kinect Ambient Intelligence Toolkit (KAIT)

KAIT was designed to simplify the development of context aware applications using the Kinect. The ultimate goal for KAIT is to support the creation of solutions that don't require direct user interaction but rather understand what the user is doing and deliver services to them transparently.

A simple example of KAIT's capabilities is a digital sign in a lobby. Rather than having a static content loop KAIT makes it easy to have the sign deliver content based on "who" the user is and where they are in the physical environment. But don't think of KAIT as just a digital sign solution. She's far more capable than that!

At her core KAIT is designed to extend the Kinect capabilities to support the acquisition of player demographics (age and gender), face recognition, voice, and player location content triggers as well as assisting in maintaining player state between interactions.

This version of KAIT makes use of NEC's industry leading facial analytics services. The KAIT team and NEC partnered to make using NEC services as easy as adding a Nuget file to a Visual Studio project. But KAIT + NEC provides more than just a Nuget as KAIT deals with issues related to image sampling cross multiple Kinect frames, correlation of biometric data to Kinect skeletons and sessions state (When do we say someone has REALLY left the player space).

KAIT also goes beyond the Kinect and provides the plumbing and scripts necessary to extend its services to Azure (Azure components are NOT required for client side interaction). These components allow guest experiences to be built where business and interaction logic are maintained in the cloud. It also provides KAIT developers with visibility into how people are interacting with KAIT experiences by providing telemetry capture and easy connection to Power BI dashboard!

KAIT's Azure services turn Kinect into an IoT sensor.

KAIT produces 3 telemetry streams: Skeletal, Demographics and Interaction. These streams are delivered via Azure Event Hub and processed by Azure Stream Analytics (ASA) where you can implement advanced business rules. For example business logic could be implemented in ASA using KAIT's data streams to notify guest services when a visitor has been "seen" in the same location for a period of time.

This all sounds very complex but "Don't Panic" KAIT contains all the scripts need to create the cloud plumbing!

But we get ahead of ourselves!

First things first. Let's download and get KAIT working!

Getting Started

KAIT Projects

The KAIT repository contains two primary solutions:

[KAIT.Biometric.Reference.sln](#)

This solution is the simplest and lets you explore most of KAIT's biometric services without all the noise of business use case. KAIT.Biometric.Reference will allow you to exercise skeletal and biometric telemetry capture and transmission. (Like all components in KAIT we tried to design them to deal with real world issues so you can run this solution WITHOUT need for an Azure subscription or internet connection!)

The app is very basic. It demonstrates how to get access to the Kinect Sensor object thru KAIT and subscribe to both the core Kinect Events as well as to the KAIT extension events.

[KAIT.Kiosk.sln](#)

KAIT.Kiosk is a full Kiosk solution that uses 3D space and biometrics to target content for display. It also has the ability to understand the location of objects in the Kinect field of view and display content based on whether they are: Touch, Removed, Replaced, and Released (Pretty cool if I do say so myself).

Prerequisites

- Windows 8.1 or Windows 10 64bit Machine
- Visual Studio 2013 or Better
- Azure SDK 2.5 or Better (not needed for client capabilities but if you want the cool cloud services)
- You will need the Kinect SDK v2
- The Speech SDK Components (64bit Please see the Kinect SDK Browser for these components)

Setup

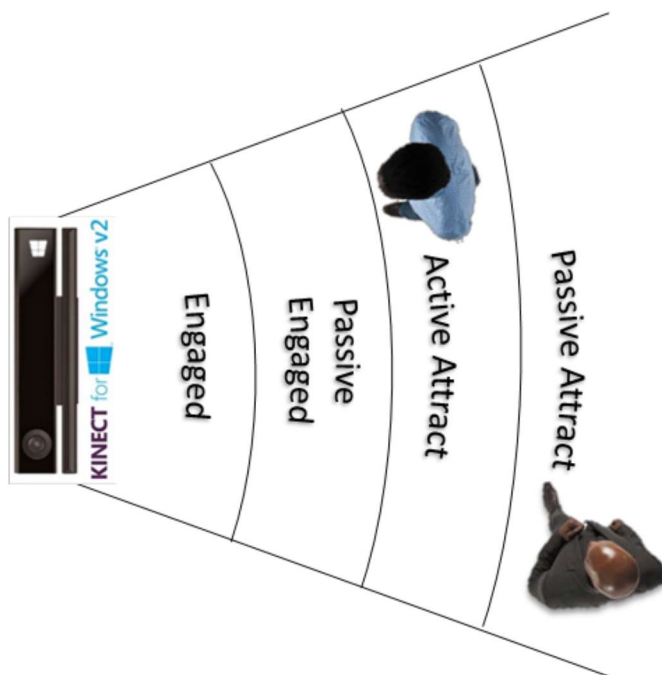
1. Install the prerequisites above (Or you'll run into compile issues)
2. Download KAIT
3. Open the KAIT.BiometricReference.sln
4. Compile
5. Run the KAIT.BiometricSample.exe – Go stand in front of your Kinect sensor say 6 feet away and look at it. In a few seconds you should get your picture and an Age and Gender result. If you DON'T see trouble shooting (Depending on the machine the sampling process can take a minute or two the first time once initialized it'll be fast)
6. Open the KAIT.Kiosk.sln
7. Compile
8. Run the KAIT.Kiosk.exe – You should get a calibration screen with depth data and a bunch of options. Again don't panic just click on close. Now go stand in front of the sensor. As you walk closer and farther away the content should change! If not see trouble shooting

KAIT Kiosk Discussion

The Kiosk app allows you to configure interaction zones. The zones are specific distance from the Kinect sensor where the kiosk will trigger content to be displayed. The zones are defined in meters from the Kinect sensor.

When a user enters a zone the Kiosk framework will loop thru the content for that zone and display it on the screen. For this version of the KAIT Kiosk the solution uses a simple directory structure that matches the interaction zone names.

Example of Zones – Zones can be named anything but must match the content directory name (see more about this later in the document)



Age and Gender Content Selection

The content selection can be further refined using the biometrics services in KAIT. The Kiosk is already designed to provide this service. To enable this, the content directories used by the Kiosk solution must be extended to include folders for age and gender (See the DefaultContent folder in the KAIT.Kiosk solution for an example. There are also configuration files that can be adjusted to further refine the content selection. This is will be discussed later in the document)

There are a couple of app settings related to age and gender processing.

Demographics.Sampling.Range – How far away from the Kinect should the user be before KAIT tries to process demographics from the Kinect color feed. Don't go too far out or the image quality will be too poor. We used the color stream for Kinect for this version of KAIT. But you could use the more interesting IR stream from Kinect. The warning with IR is the image quality is lower and the user would need to be closer to the display before sampling takes place.

```
<!-- How far away from the Kinect Sensor should the player be before we attempt to
perform Biometric of face recognition processing-->
<add key="Demographics.Sampling.Range" value="3.2" />
```

OverSamplingThreshold- Because we are sampling faces from a live video feed the user could have looked away or there was motion blur in the image. So taking several increase the changes we have a good result.

```
<!--BiometricTelemetryService setting indicating how many samples should be taken for an
individual before they should be transmitted-->
<add key="BiometricTelemetryService.OverSamplingThreshold" value="1" />
```

In this version of the KAIT the Age and Gender demographics are provide by NEC's Neoface Engage services. This component is included in KAIT as a Nuget reference. When you first build and compile the KAIT solutions the Neoface Engage Nuget package should be restored and you will receive a 5 day developer license. If you need a longer license please contact NEC Ganz, Allen <allen.ganz@necam.com>

If you want to extent these services KAIT supports a standard set of interfaces for Biometrics processing. This would allow you to build your own based on the NEC components or just extend the ones in KAIT.

Object Tracking and Content Selection

The Kiosk app also supports delivering content based on the user's interaction with objects in the Kinects view. To use this service the KAIT object detection services need to be calibrated.

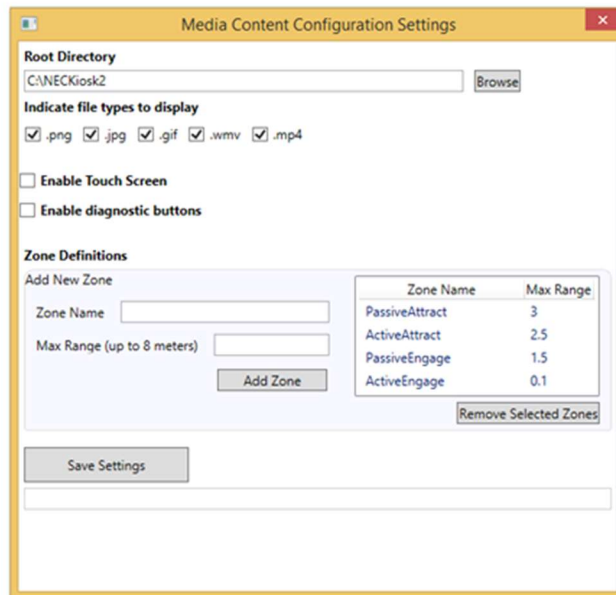
In this version of KAIT object detection is simple. It allows KAIT to find objects at a set distance for the Kinect sensor. All of the objects need to be at a similar distance from the sensor for KAIT to track.

Once the object detection service has been calibrated it will monitor the object(s) location and raise a series of events based on user interactions: Touched, Removed, Replaced, and Released. Think of these as somewhat analogous to mouse events in a windows app. (Important tip: This version of the KAIT object detection is very simple as a result things need to stay or be put back in the same location)

Kiosk App Configuration

The Kiosk App has a number of settings that customization of the experience. Many of them can be changed directly from the app.

Kiosk configuration can be launch by pressing [Ctrl][C]



Root Directory - The content directory for the Kiosk. It must be formatted based on the Zones and Items being monitored by the Kiosk

Enable Touch Screen - Checking this option tell the Kiosk to load the touch screen view when the user reaches the closest zone to the Kinect Sensor

Enable Diagnostics - Checking this option has the Kiosk report all the interaction states to the display as well as providing quick launch buttons to kiosk features

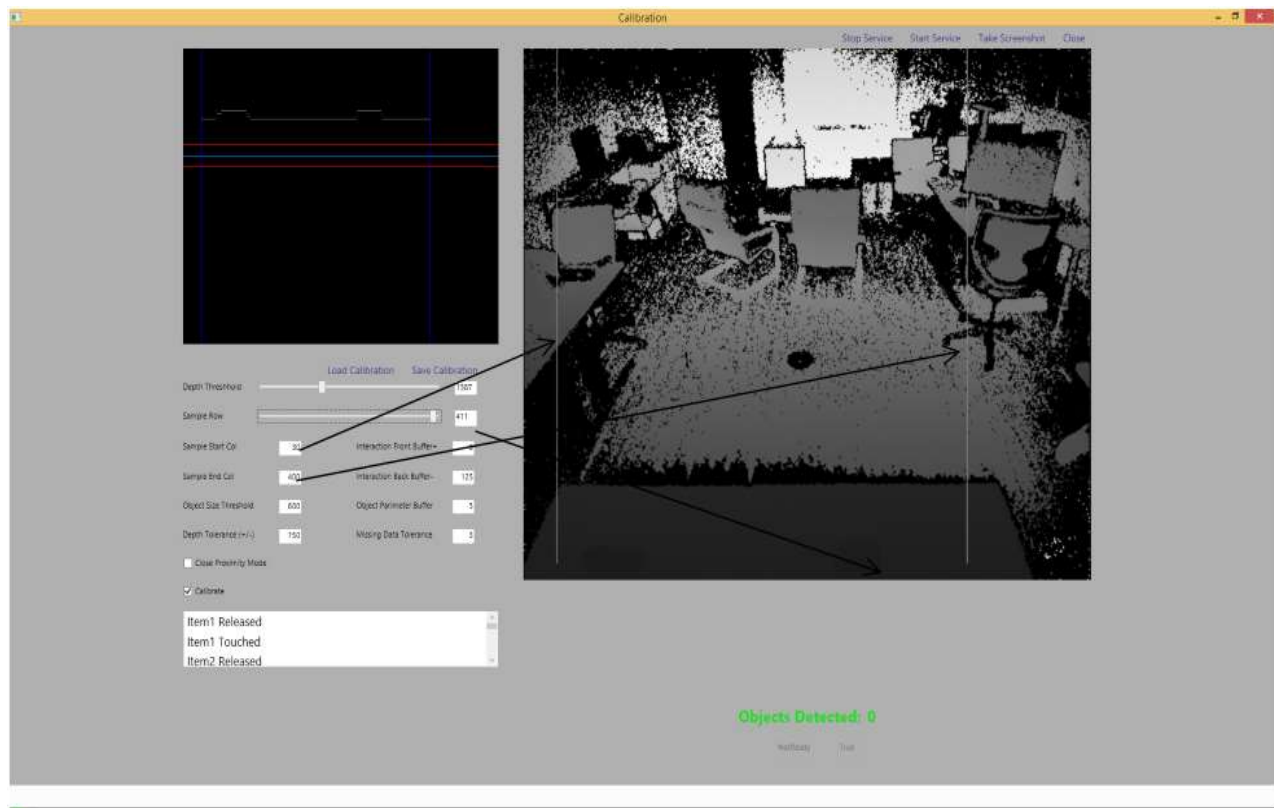
Zone Definition - This allows you to configure the number of zones to be monitored by the Kiosk and their distance from the Kinect sensor. The can be any name but they must also have a content directory by the same name. If you wish to target content based on demographics then you must provide the appropriate directory structure under the Zone name. Please see Kiosk Content.

Save Settings - Persists the configuration and forces the Kiosk to reload its specifications

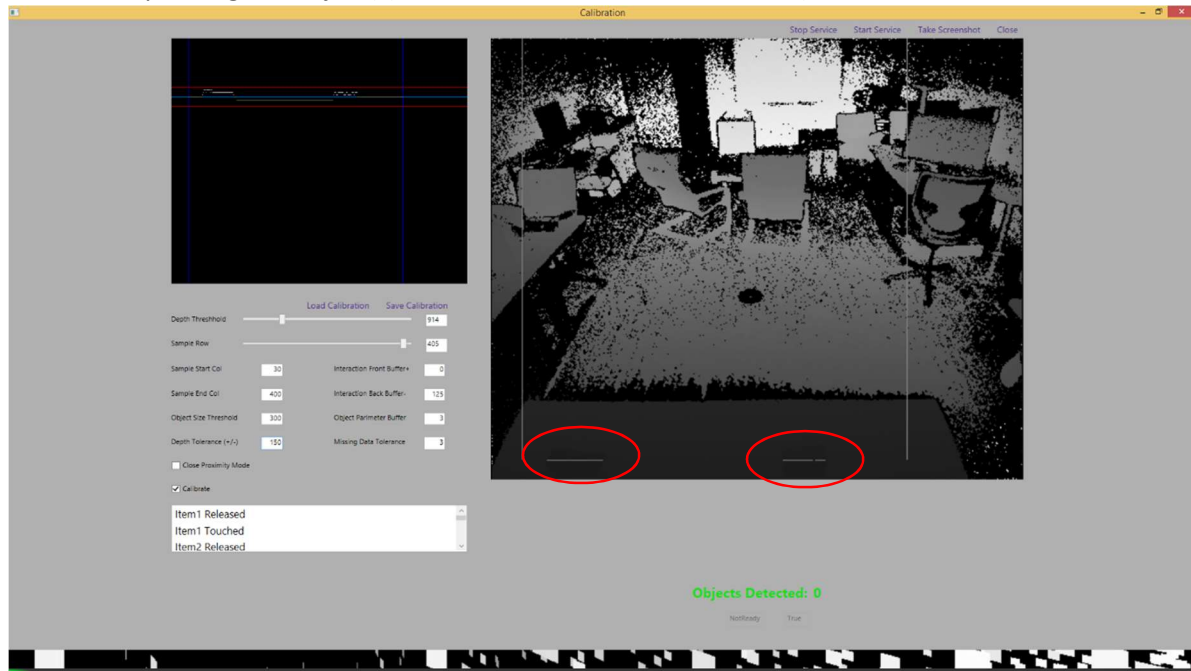
Kiosk Object Detection Calibration

The first time you run the Kiosk it will launch the Object Detections Calibration window. If you don't need it for your solution just click "Close."

Calibration starts with you specifying the "Sampling Row" which represents the horizontal line on the screen where the Kiosk is looking for objects. You should adjust this up and down until the line bisects the objects that need to be monitored (What you're seeing on the left hand side of the screen in the Kinect Sensors Depth stream. Each pixel color represents a different distance from the Kinect).

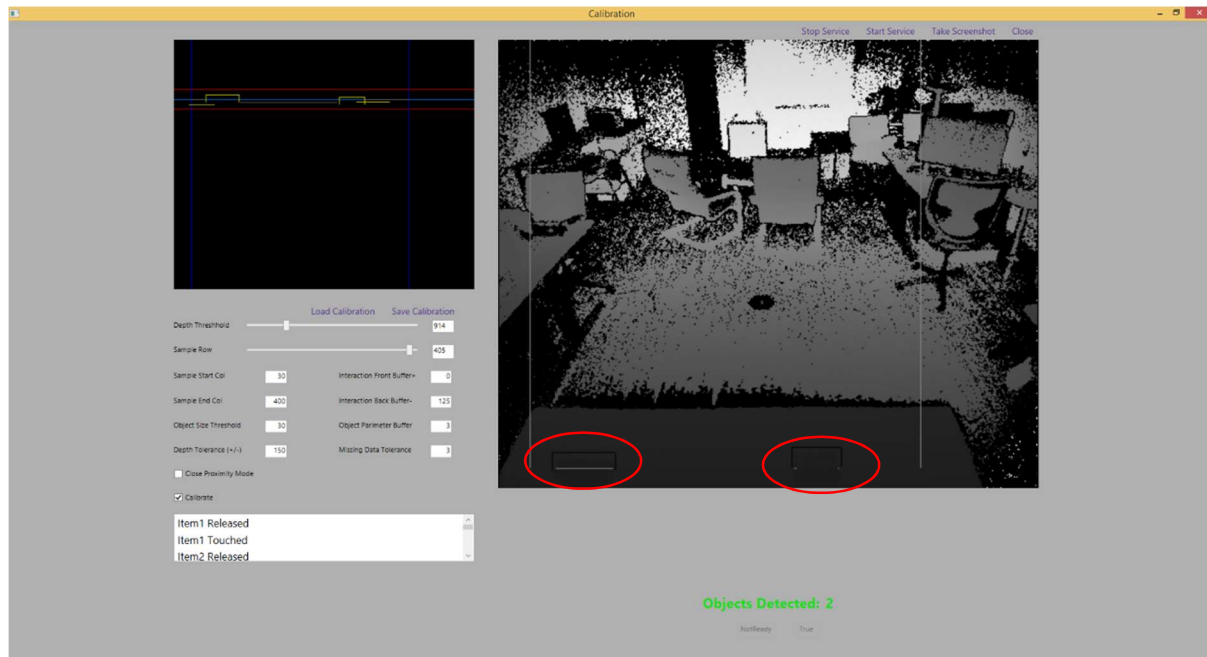


Now we adjusted the Depth Threshold to the point that we're highlighting the objects (You should see a white line spanning the object(s) with no line visible between each)



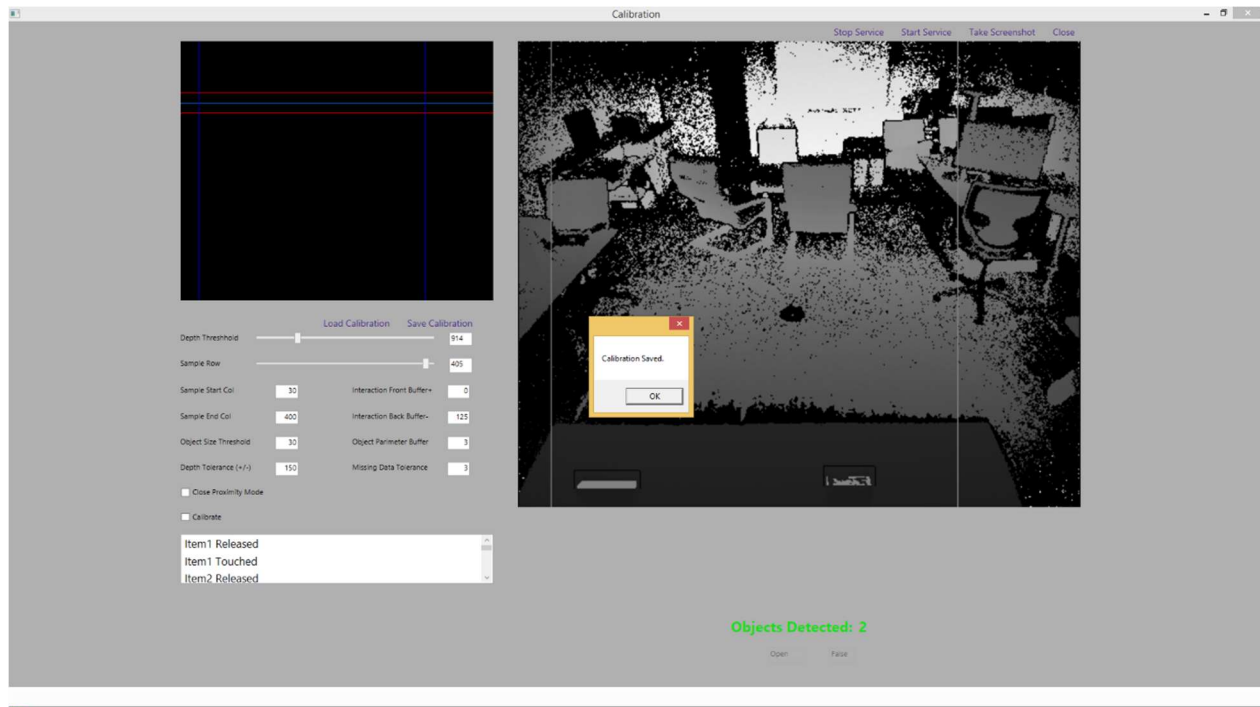
Now we change the Object Threshold to something smaller than 300 telling the service that things at this size and larger are objects we're interested in... You'll see that the app has put a bounding frame around the objects now. The bounding frame is based on the object size detected by the app and the "Object Parameter Buffer" setting. The app also uses the "Depth Tolerance" and "Missing Data Tolerance" settings to help find the objects since there can be a lot of noise in the signal.

"Depth Tolerance" is the number of millimeters the distance can vary +/- from the "Depth Threshold" and still be considered part of the same object.



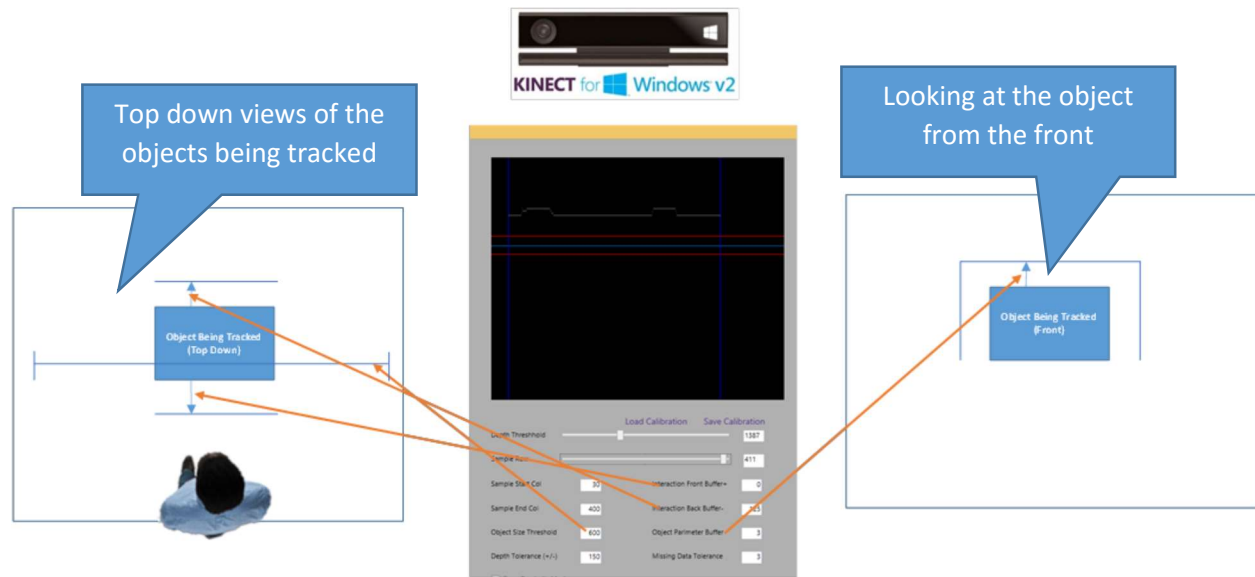
Here we have "uncheck" the "Calibrate" and the app is running as it would in normal mode. This allows you the reach out and touch objects and see if the app processes the events as you expect. We have also clicked on the "Save Calibration" button to persist our configuration.

You note that not only do we have the object bounding lines present but that parts of the object are now highlighted indicating that the sensor "sees" them. If you were to remove them, the highlighting would disappear and you should receive an "Object Removed" event. Placing the object back in the same location would result in it being highlighted again and you receiving a "Object Replaced" event. (Yes object placement is a limitation in this approach. If you don't put it back exactly the system may still think its missing.)



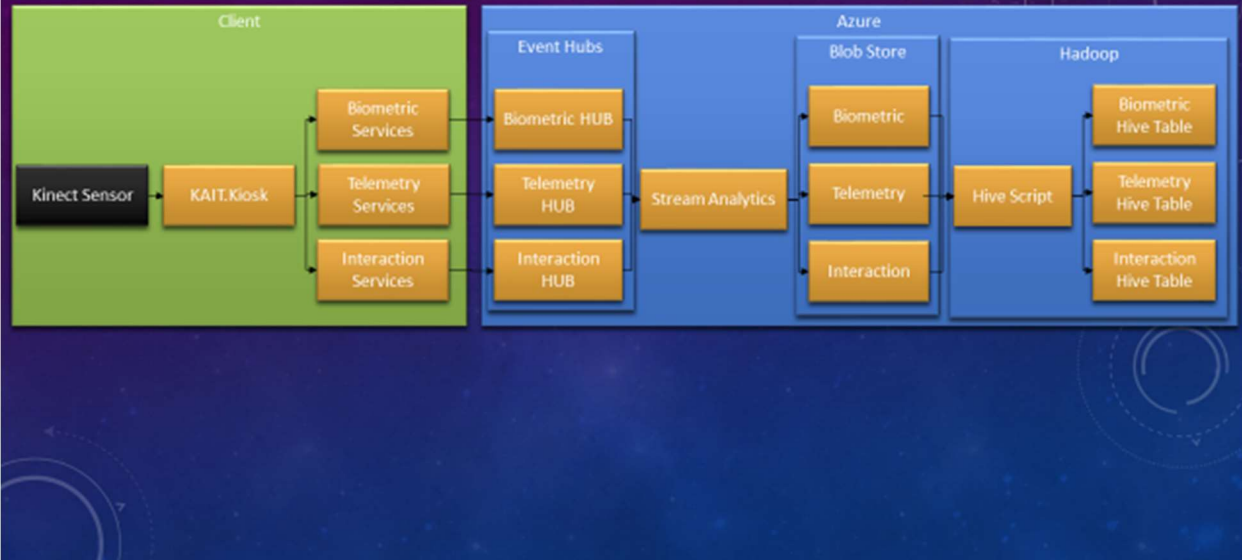
When the calibration is saved the file it creates is called "[ObjectDeteetionConfig.json](#)" and it's persisted to the install location for the app. The app also supports loading different configs from the Calibration screen.

The other setting in the calibration window can be a bit esoteric so perhaps a picture will help.

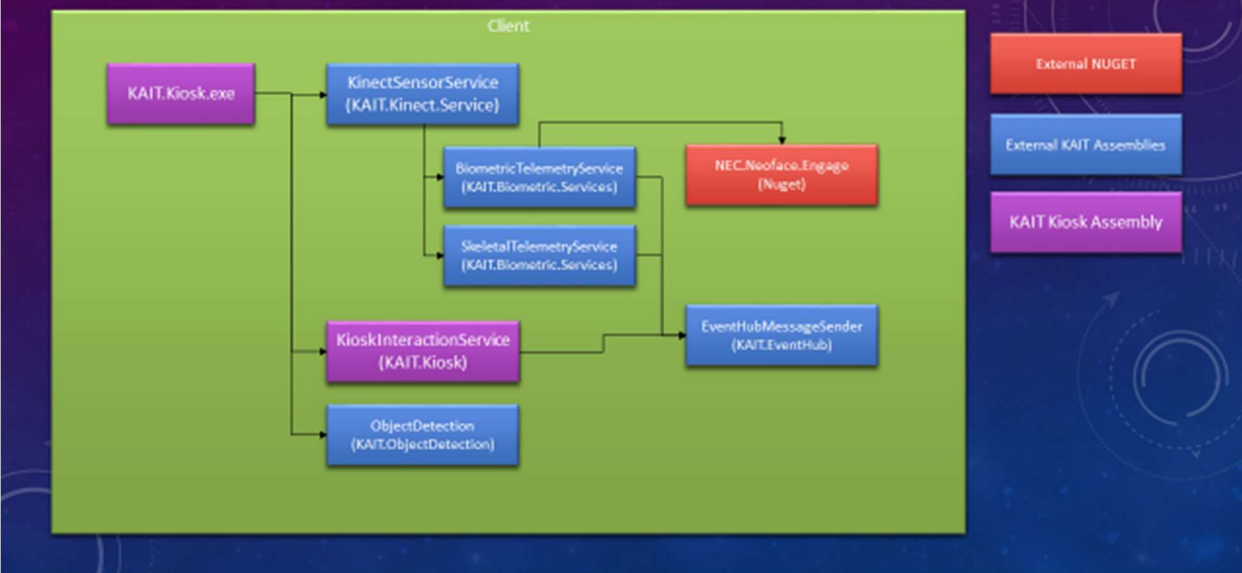


KAIT.Kiosk Architecture

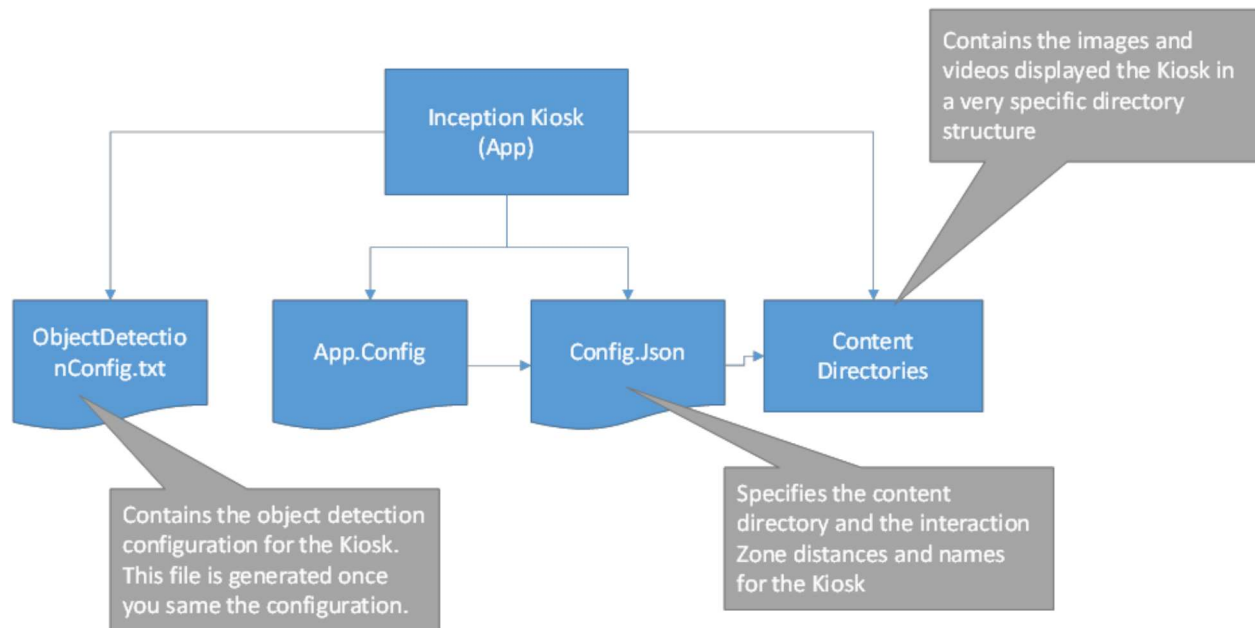
KAIT KIOSK HIGH LEVEL ARCHITECTURE



KAIT KIOSK APPLICATION ARCHITECTURE

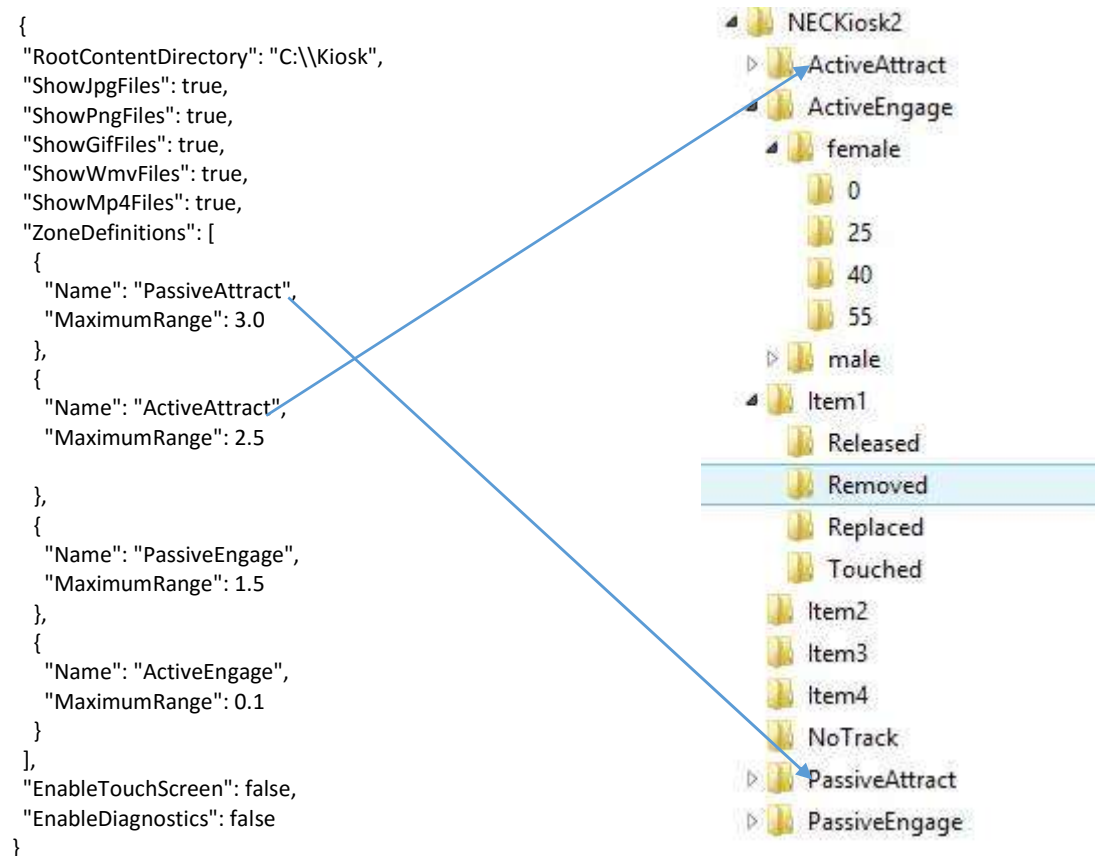


Support and Configuration Files



Config.json with mapping to content directory

*The Config.json file is managed by the Kiosk app and does not need to be edited directly



Trouble Shooting

If you've successfully compiled the application and the app isn't working let's start with the basics.

1. Run Kinect SDK Configuration Checker
2. Run Kinect SDK Body Tracking Sample
3. Run Kinect SDK Depth Data Sample
4. Run Kinect Speech Sample

Biometrics don't seem to be captured or you're getting an Exception at Startup

5. Run Biometric Reference App

If you're running the Biometric sample application and you're not getting any data returned. Couple things to check: In the BiometricTelemetryService.cs

1. Put a break point in where it has (see if you're getting an image) If you reach this point just let it run:

```
if (!TestMode)
    result = NEC.NeoFace.Engage.NECLabs.DetectFace(bmp);
else
    result = null;
```

1. Put a break point at the catch block for the ProcessFace method – This will tell us if we have problems with NEC

```
catch(Exception ex)
{
    _IsNECInFaultCondition = true;
    OnDemographicsProcessingFailure(ex.Message + " " + ex.InnerException);
    return false;
}
```

2. Oversampling: Remember we default to requiring 4 good samples before we return resolution. You can change this and the Sampling Range (The distance at which we take the sample) in the App.config

```
<add key="Azure.Hub.SkeletalHub" value="Endpoint=sb://inceptioningess-
ns.servicebus.windows.net/;SharedAccessKeyName=SendPolicy;SharedAccessKey=dgncIvTYT6
KrJSPzcfHfEnHswE8SoDxP8g/bn0+SPj8=;EntityPath=telemetry"/>
```

```
<add key="Azure.Hub.Biometric" value="Endpoint=sb://inceptioningess-
ns.servicebus.windows.net/;SharedAccessKeyName=SendPolicy;SharedAccessKey=VJrXuJaw60
Og5saHUBDm8bdKKceUbq9B39KuRhIa90k=;EntityPath=demographics"/>
```

```
<add key="BiometricTelemetryService.OversamplingThreshold" value="4"/>
```

```
<add key="EventHubMessengerSender.RetryPeriodInSeconds" value="60"/>
```

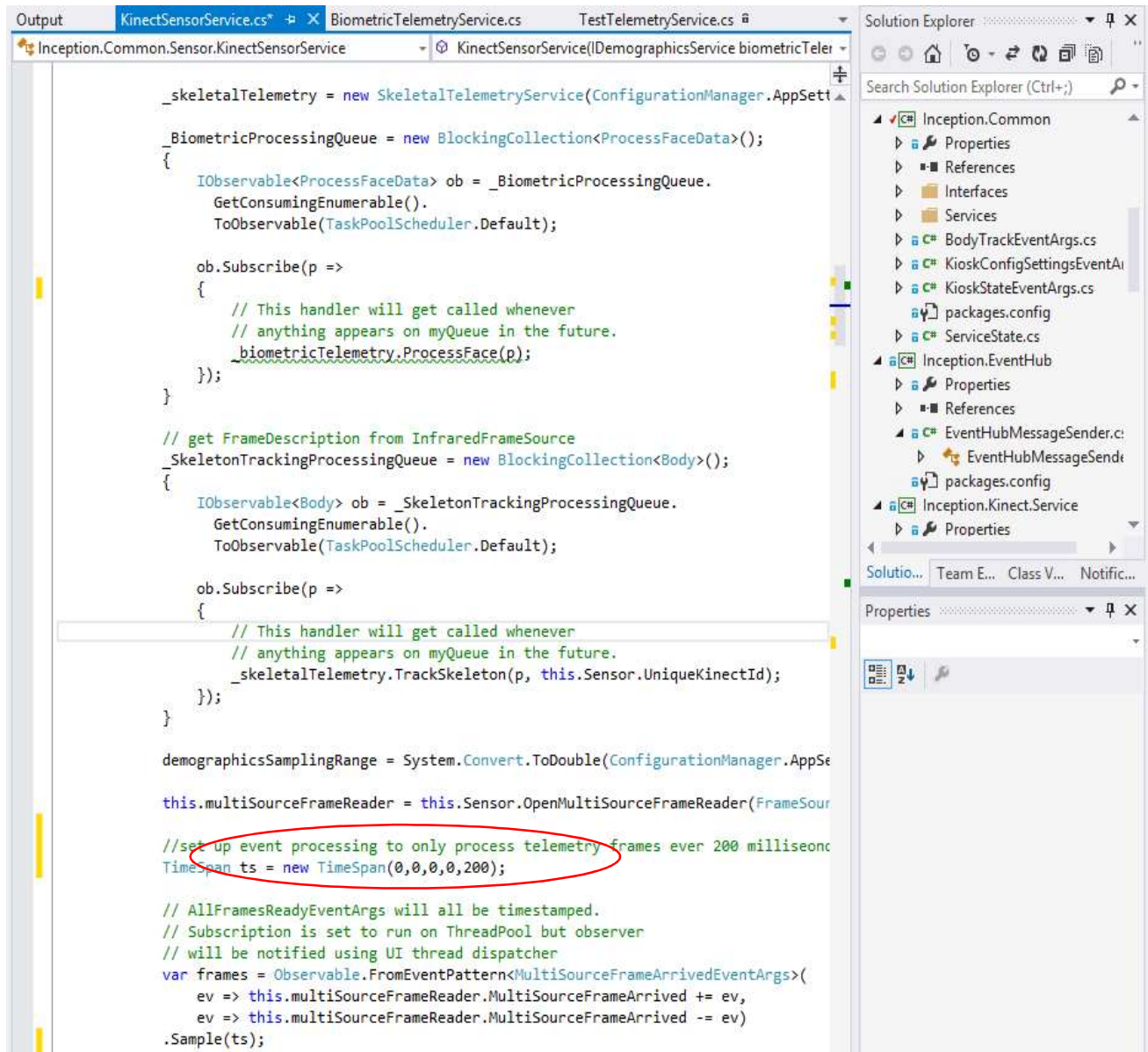
```
<add key="Demographics.Sampling.Range" value="3.2"/> (Meters)
```

Finally if all of this is working sometimes it can take the NEC stuff a min or two (really) to get initialized on the first call. After that it's screaming fast.

Developer Notes

Change the rate at which KAIT samples and sends telemetry

KinectSensorService.cs



```
Output  KinectSensorService.cs*  BiometricTelemetryService.cs  TestTelemetryService.cs
Inception.Common.Sensor.KinectSensorService  KinectSensorService(IDemographicsService biometricTele...

_skeletalTelemetry = new SkeletalTelemetryService(ConfigurationManager.AppSettings

_BiometricProcessingQueue = new BlockingCollection<ProcessFaceData>();
{
    IObservable<ProcessFaceData> ob = _BiometricProcessingQueue.
        GetConsumingEnumerable().
        ToObservable(TaskPoolScheduler.Default);

    ob.Subscribe(p =>
    {
        // This handler will get called whenever
        // anything appears on myQueue in the future.
        _biometricTelemetry.ProcessFace(p);
    });
}

// get FrameDescription from InfraredFrameSource
_SkeletonTrackingProcessingQueue = new BlockingCollection<Body>();
{
    IObservable<Body> ob = _SkeletonTrackingProcessingQueue.
        GetConsumingEnumerable().
        ToObservable(TaskPoolScheduler.Default);

    ob.Subscribe(p =>
    {
        // This handler will get called whenever
        // anything appears on myQueue in the future.
        _skeletalTelemetry.TrackSkeleton(p, this.Sensor.UniqueKinectId);
    });
}

demographicsSamplingRange = System.Convert.ToDouble(ConfigurationManager.AppSe

this.multiSourceFrameReader = this.Sensor.OpenMultiSourceFrameReader(FrameSour

//set up event processing to only process telemetry frames ever 200 milliseonc
TimeSpan ts = new TimeSpan(0,0,0,0,200);

// AllFramesReadyEventArgs will all be timestamped.
// Subscription is set to run on ThreadPool but observer
// will be notified using UI thread dispatcher
var frames = Observable.FromEventPattern<MultiSourceFrameArrivedEventArgs>(
    ev => this.multiSourceFrameReader.MultiSourceFrameArrived += ev,
    ev => this.multiSourceFrameReader.MultiSourceFrameArrived -= ev)
.Sample(ts);
```

Solution Explorer

- ✓ Inception.Common
 - Properties
 - References
 - Interfaces
 - Services
 - BodyTrackEventArgs.cs
 - KioskConfigSettingsEventArgs.cs
 - KioskStateEventArgs.cs
 - packages.config
 - ServiceState.cs
- Inception.EventHub
 - Properties
 - References
 - EventHubMessageSender.cs
 - EventHubMessageSendi
 - packages.config
- Inception.Kinect.Service
 - Properties

Solution... Team E... Class V... Notific...

Properties