Extreme Motion SDK C# API and Unity – Hands On

Extreme Reality

4/10/2014

Table of Contents

1 Overview

The ExtremeMotion SDK uses the Singleton pattern. It is a thread safe, event driven mechanism which requires registration and setting up handlers. Callbacks occur in the context of the thread that was registered.

2 Initializing

First acquire an instance:

```
GeneratorSingleton.Instance.Initialize(PlatformType platform, ImageInfo imageInfo);
```

Choose platform type and image format using the following enums:

```
Xtr3D.Net.ImageInfo.ImageFormat
Xtr3D.Net.BaseTypes.PlatformType
```

3 Start

In order to receive data the engine must be started:

```
GeneratorSingleton.Instance.Start();
```

Start function acquires the camera and starts processing the video streamed from the camera. Please note: Before calling Start() you should make sure the camera is connected and not in use by another process.

Possible exceptions that might be thrown:
No camera was found: `Xtr3D.Net.Exceptions.CameraAbsentException`
Camera is used by another process: `Xtr3D.Net.Exceptions.CameraBusyException`

4 Stop

To stop acquiring the camera and processing the video streamed from it:

```
GeneratorSingleton.Instance.Stop();
```

If your application supports OS suspend/resume states, call Start/Stop on these events.

5 Reset

Reset can be explicitly called when the application needs to re-calibrate the user. There is no need to call it unless there is a different user. Re-calibration will measure the body proportions again.

6 Shutdown

To close the engine:

```
GeneratorSingleton.Instance.Shutdown();
```

Initialize and Shutdown should be called once in the application's lifetime.

7 Image Acquisition

Extreme Motion SDK captures images directly from the camera and processes them. Developers may change this behavior and capture images on their own by modifying the “Source” field in the supplied configuration file, ConfigBase.xml:

```
<param name="Source" value="External" />
```

Once the configuration file is modified, the developer is responsible for supplying images to Extreme Motion SDK by calling `DirectImageFeed.HandleCapturedImage` method. Calling `DirectImageFeed.HandleCapturedImage` method without setting the configuration file as shown above will result with an error.

When using an external image source, the developer can control the way injected images are handled using the parameter "SyncExternalSource" in the configuration file ConfigBase.xml:

- Setting the parameter to “false” (which is the default mode):
  `<param name="SyncExternalSource" value="false" />`
  Will configure `DirectImageFeed.HandleCapturedImage` method to run in a non-blocking mode, meaning the method will return immediately but the frame might be dropped.
- Setting the parameter to true:
  `<param name="SyncExternalSource" value="true" />`
  Will configure `DirectImageFeed.HandleCapturedImage` method to run in a blocking mode, meaning the method will return only when Extreme Motion SDK has started processing the injected image.

8 RGB image event

To display an RGB image you should register to the ColorImageFrameReady event.

Registration:

```
GeneratorSingleton.Instance.ColorImageFrameReady +=
new EventHandler<ColorImageFrameReadyEventArgs>(OnColorImageReceived);
```

Handler setup:

```
void OnColorImageReceived(object sender, ColorImageFrameReadyEventArgs e) { };
```

Receiving image data:

```
ColorImageFrame colorImageFrame = e.OpenFrame();
colorImageFrame.ColorImage.Image; // the new image
```

Receiving RGB image warnings:

To receive the warnings from RGB image event:

```
var currWarnings = colorImageFrame.Warnings
```

For possible warnings, check the enum:

```
Xtr3D.Net.ImageWarnings
```

9 Gesture Data Event

To receive the Gesture data you should do the following:

1. Create gestures file using *Superpose*.
2. Set gesture file by calling GeneratorSingleton.Instance.SetGestureRecognitionFile
3. Register to the GesturesFrameReady event.

Registration:

```
GeneratorSingleton.Instance. GesturesFrameReady +=
new EventHandler< GesturesFrameReady EventArgs>(On GesturesFrameReady);
```

Handler setup:

```
void On GesturesFrameReady (object sender, GesturesFrameReady EventArgs e) { };
```

Gesture Types:

`DownGesture` - Received when a user moves downwards
`HeadPositionGesture` - Received each frame and specifies the index of the head region among a predefined set of regions.
`SequenceGesture` - Received when a user sequentially performs a predefined set of gestures
`StaticPositionGesture`- Received when a user poses a predefined pose

`SwipeGesture`- Received when a user moves his/her hand from left to right or right to left
`UpGesture` - Received when a user moves upwards
`WingsGesture` - Received when a user spreads his hands widely. The class holds the angle of the user hands

Receiving gesture data:

```
GesturesFrame gesturesFrame = e. OpenFrame();
foreach (BaseGesture gesture in gesturesFrame.FirstSkeletonGestures())
{
    // Update messages for gesture
    if (!gestureMessages.ContainsKey(gesture.ID))
    {
        gestureMessages.Add(gesture.ID, new GestureMessage(gesture.Description));
    }
    gestureMessages[gesture.ID].timeToLiveCounter = gestureTypeToDelay[gesture.Type];
    switch (gesture.Type)
    {
        case BaseGesture.GestureType.HEAD_POSITION:
        {
         HeadPositionGesture headPositionGesture = gesture as HeadPositionGesture;
         int regionIndex = headPositionGesture.RegionIndex;
         break;
        }
        // Other gestures
        case BaseGesture.GestureType...:
         {

         }
    }
}
```

10 Skeleton Data Event

To receive the Skeleton data you should register to the DataFrameReady event.

Registration:

```
GeneratorSingleton.Instance.DataFrameReady +=
new EventHandler<DataFrameReadyEventArgs>(OnDataFrameReady);
```

Handler setup:

```
void OnDataFrameReady (object sender, DataFrameReadyEventArgs e) { };
```

Receiving skeleton data:

```
DataFrame dataFrame = e. OpenFrame();
JointCollection mySkeleton = dataFrame.Skeletons[0].Joints;
```

JointCollection type holds all the skeleton joints data.
To receive data on a certain Joint (for instance Head X position):

```
float headX = mySkeleton.Head.skeletonPoint.X
```

Receiving skeleton warnings:

To receive the skeleton warnings:

```
var currWarnings = dataFrame.Skeletons[0].ClippedEdges
```

For possible warnings check enum:

```
Xtr3D.Net.ExtremeMotion.Data.FrameEdges
```

11 AllFramesReady Event

Registration:

```
GeneratorSingleton.Instance.AllFramesReady  +=
new EventHandler<AllFramesReadyEventArgs>(OnAllFramesReady);
```

Handler setup:
```
void OnAllFramesReady (object sender, AllFramesReadyEventArgs e) { };
```

Receiving skeleton data and RGB image:
```
AllFramesFrame allFrames = e.OpenFrame() as AllFramesFrame;
foreach (var args in allFrames.FramesReadyEventArgs)
{
   InvokeHandler(sender, args);
}
```

Invoke handler will be:
```
void InvokeHandler(object sender, EventArgs eventArgs)
```

Save each event args:
```
var dataFrameReady = eventArgs as DataFrameReadyEventArgs;
var colorImageFrameReady = eventArgs as ColorImageFrameReadyEventArgs;
```

Then call each event separately if data received is not null:
For example:
```
if (null != dataFrameReady)
{
   this.MyDataFrameReady(sender, dataFrameReady);
   return;
}
```

Please note:
Each frame holds a frame ID/number and timestamp which can be used for manual synchronization or improving usability.

To access frame ID:
```
dataFrame.FrameKey.FrameNumberKey
```

To receive frame timestamp:
```
dataFrame.TimeStamp
```

12 Skeleton Proximity

To get the skeleton proximity (distance from camera):
```
float proximity = dataFrame.Skeletons[0].Proximity.SkeletonProximity;
```

13 Skeleton Tracking State

Holds the current Skeleton state, you can use it to react differently to each Skeleton state in your application.

Receiving the skeleton state:

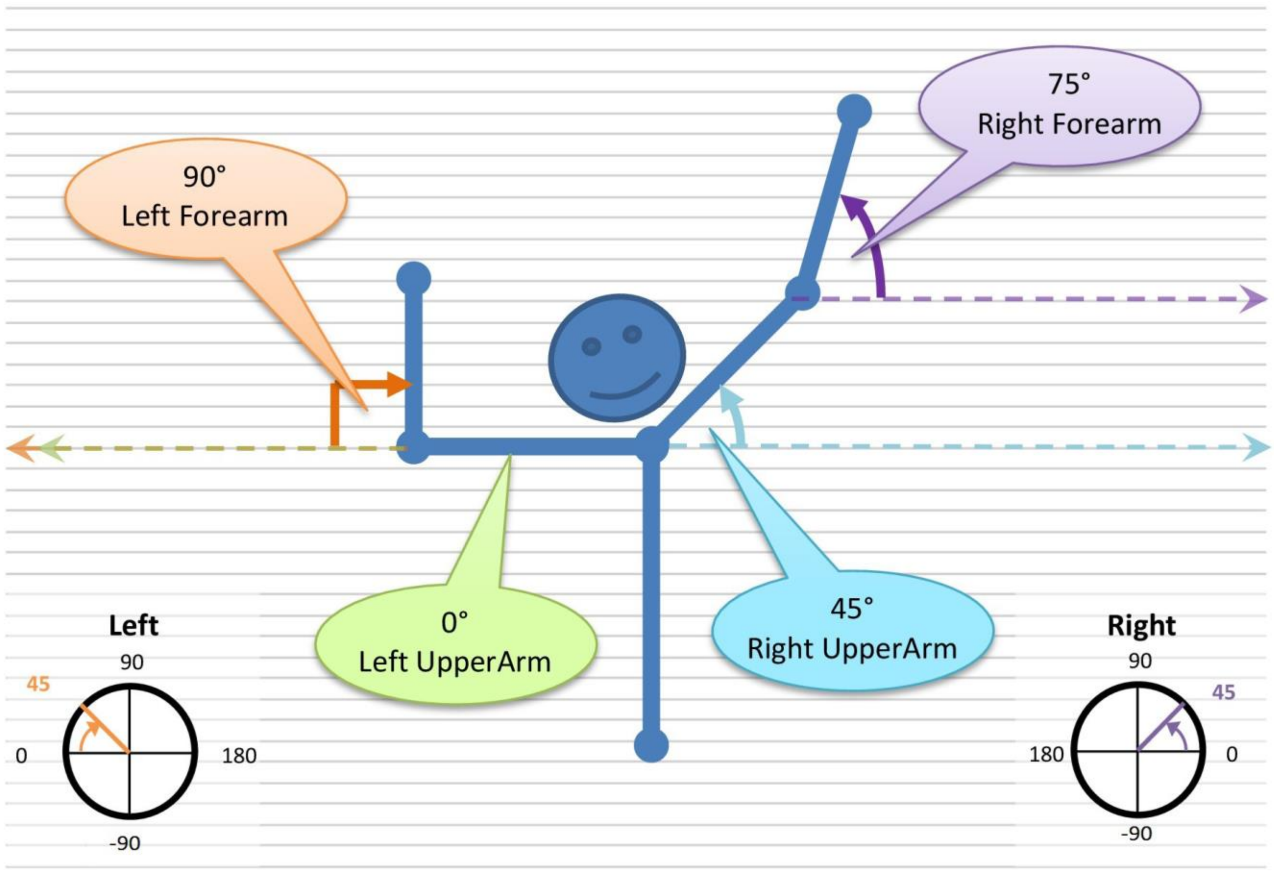
```
TrackingState skeletonState = dataFrame.Skeletons[0].TrackingState;
```

Skeleton states are: Initializing, Calibration, Not Tracked, and Tracked.

14 Detecting Predefined Gestures

The Extreme Motion SDK contains the ability to also detect predefined gestures, via the **GESTURES** stream. This is achieved by supplying a predefined set of static gestures via an xml file. The xml is can be edited and tested in real-time using *SuperPose* GUI Tool available at xtr3d site under http://www.xtr3d.com/developers/resources/.

In order to use this stream, you must perform the following tasks:
Make sure the xml file resides in the application's public external directory.
Invoke the `GeneratorSingleton.Instance.SetGestureRecognitionFile` method to set the xml file, which contains the description of the static gestures to be detected. You may safely call this several times, however, you should note the following The filename given in the latest invocation of this method is the one that will be used. An example for the xml file can be found below along with a diagram reference to gestures.



Registration:

```
GeneratorSingleton.Instance.GesturesFrameReady
```

Handler setup:

```
void OnGesturesFrameReady (object sender, GesturesFrameReadyEventArgs e) { };
```

Receiving skeleton data:

```
GesturesFrame dataFrame = e. OpenFrame();
BaseGesture[] gestures = gesturesFrame.FirstSkeletonGestures();
```

Static Gesture XML File, created using the *SuperPose* GUI tool at http://www.xtr3d.com/developers/resources/

Project_name.xml

This is the file you will need to integrate into the main project.
This file holds the data for the Static Gesture that you have created.

Example:

```
<Xtr3DGesturesProfile>                 (Default)
  <XmlVersion>1</XmlVersion>           (Default)
  <Gesture>
   <Id>1</Id>                          (ID number of the gesture.  manually assigned by the developer.)
   <Name>Both_Hands</Name>             (Gesture name, as named by the developer)
   <State>Enabled</State>              (Default)
   <Side>Both</Side>                   (This field decide which hand will be used for position detection. possible values: Both, Left, Right)
   <Type>1</Type>                      (Default)
   <RightHand>                         (Parameters for the right hand)
     <UpperArm>
       <Angle>180</Angle>              (Angle of the Upper Arm. the angle is displayed in figure 1.1)
      <Tolerance>30</Tolerance>        (How loose or tight the recognition of the upper arm will be. possible values: 10[tight]-90[loose] degrees)
     </UpperArm>
      <Forearm>
       <Angle>90</Angle>               (Angle of the Forearm. the angle is displayed in figure 1.1)
      <Tolerance>30</Tolerance>        (How loose or tight the recognition of the forearm will be. possible values: 10[tight]-90[loose] degrees)
     </Forearm>
    </RightHand>
    <LeftHand>                         (parameters of the left hand)
     <UpperArm>
       <Angle>176</Angle>
       <Tolerance>30</Tolerance>
      </UpperArm>
      <Forearm>
       <Angle>93</Angle>
       <Tolerance>30</Tolerance>
      </Forearm>
    </LeftHand>
  </Gesture>
  <Gesture>                            (new Static gesture)
   <Id>2</Id>
    <Name>Right Hand</Name>
    <State>Enabled</State>
    <Side>Right</Side>                 (if only one hand is chosen, then the parameters for the other hand are disabled and will not show)
   <Type>1</Type>
    <RightHand>
      <UpperArm>
       <Angle>33</Angle>
       <Tolerance>30</Tolerance>
      </UpperArm>
      <Forearm>
       <Angle>129</Angle>
       <Tolerance>30</Tolerance>
      </Forearm>
    </RightHand>
  </Gesture>                           (only right hand was chosen. the parameters for the left hand will not be displayed.)
  <Gesture>
</Xtr3DGesturesProfile>
```