Kinemoto user guide

Introduction

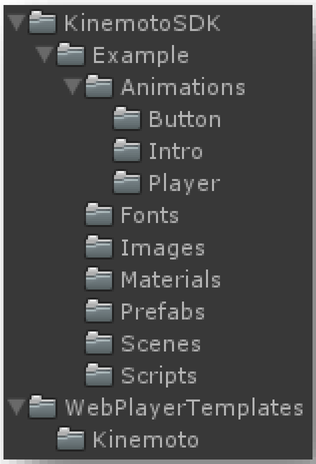
Thank you for purchasing the KinemotoSDK, there are a few simple steps to get started with the KinemotoSDK which will be explained in this guide.

The current version of the KinemotoSDK is v2.2.0.0.
The minimal required KinemotoServer version is v3.0.0.0.

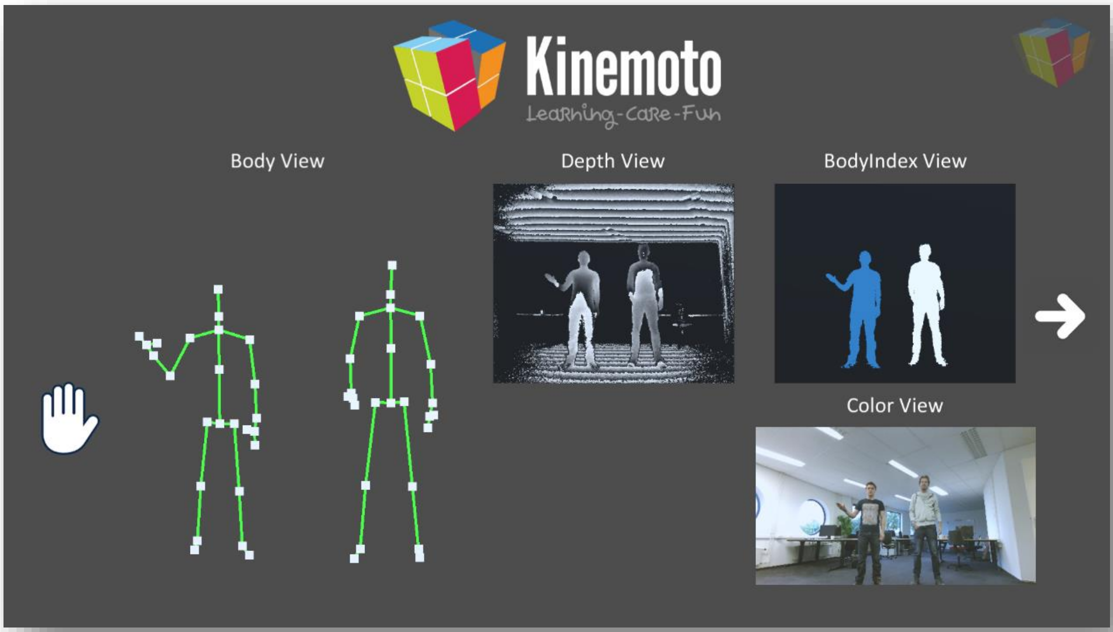
If you have any questions be sure to send us an e-mail at: support@kinemoto.com

1 Kinemoto Package

When you import the Kinemoto Package from the asset store two folders will be added to your project: "*KinemotoSDK*" and "*WebPlayerTemplates*". Everything you need to set up Kinemoto in your project is located inside the KinemotoSDK folder but Unity3D still needs to have the "*WebPlayerTemplates*" folder in the Assets directory to be able to find the Build template.



The KinemotoSDK folder contains this guide, the Library and an Example folder. The example folder contains some assets that are visible in the scene and if you need an example of one of the elements that are explained in the rest of the guide you can take a look at the example.



1.1 Web Player Template:

To build for web you need to set up the web player template. You can do this by going to File>Build Settings. In the Build Settings window select Web Player as target platform and press Switch Platform. Now that you're building for web press Player Settings and select the Kinemoto Web Player Template.

Kinemoto web applications are designed to run at a Full HD resolution so if the current "**Player Settings**" are set to a different resolution, be sure to change it to 1920*1080.

1.2 Kinemoto Logging System:

If you set up your project to build for Web with the Kinemoto Web Player Template you can use the Kinemoto Logging System as an alternative to the Unity3D logging system. You can send a Debug log to Kinemoto with:

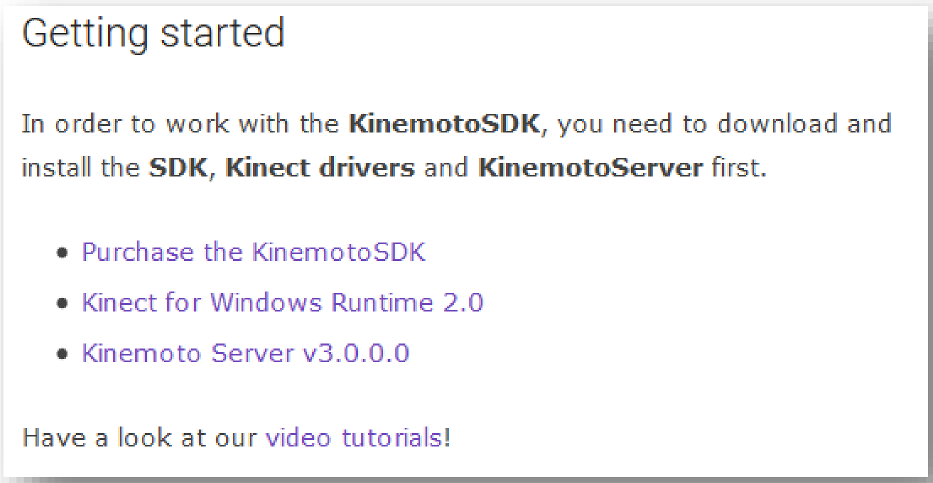
```
Kinemoto.Log("Your Debug log message here");
```

When using the Kinemoto Logging System you can see your debug messages in the browser window as well. This allows you to debug from the web player more easily than with the standard debugger.

2 Preparation

To use Kinemoto for our Kinect Game we need to have two things installed aside from the package you just downloaded. Namely: **The Kinemoto Server** and if you haven't installed it already **Microsoft's Kinect for Windows v2.**

You can find both of these at our developer site: developer.kinemoto.com
Just scroll down to "Getting started" and you will find both of the download links.



Getting started

In order to work with the **KinemotoSDK**, you need to download and install the **SDK**, **Kinect drivers** and **KinemotoServer** first.

- Purchase the KinemotoSDK
- Kinect for Windows Runtime 2.0
- Kinemoto Server v3.0.0.0

Have a look at our video tutorials!

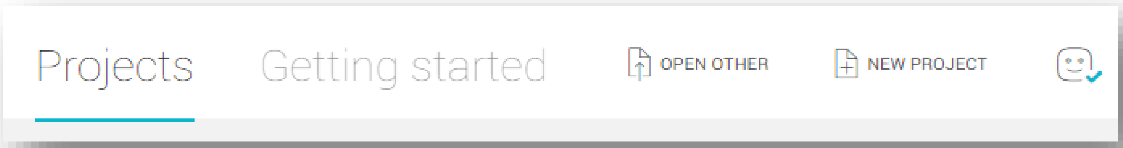
After downloading these you will need to install the Kinemoto Server and Kinect for Windows v2. We will talk about the Kinemoto SDK in the next chapter where we talk about implementing Kinemoto in Unity.

3 Unity Implementation

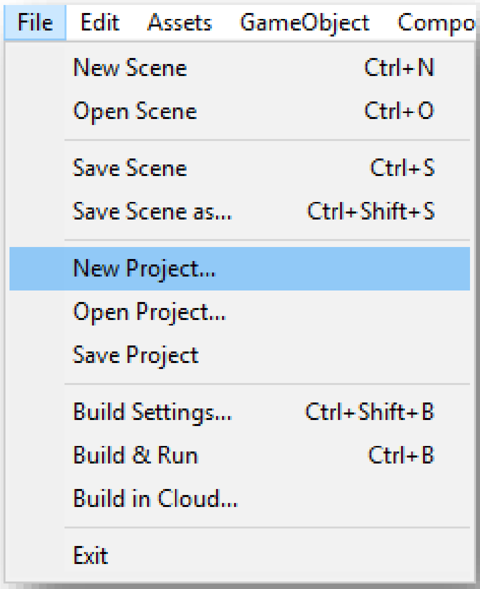
In this Chapter we're going to create a *new Unity project*, we're going to load in the Unity SDK and we are going to implement it. If you already have a Unity project you can skip **Step 1** and go directly to **Step 2**.

Step 1: Create a new unity project

Launch Unity and press "**New Project**"



Or if you already have a project open and want to start from scratch press **File > New Project**



Pick a name for your project and choose the standard assets you want to use (none in our case, since you can always add them later).

Step 2: Turn on the Kinemoto Server

Remember the Kinemoto Server we installed in the previous chapter? Locate the program or its shortcut and make sure its running. Kinemoto won't work without the server so every time you wish to test or play your Kinemoto application make sure the Kinemoto server is running.

Step 3: Load the KinemotoSDK Package into your project

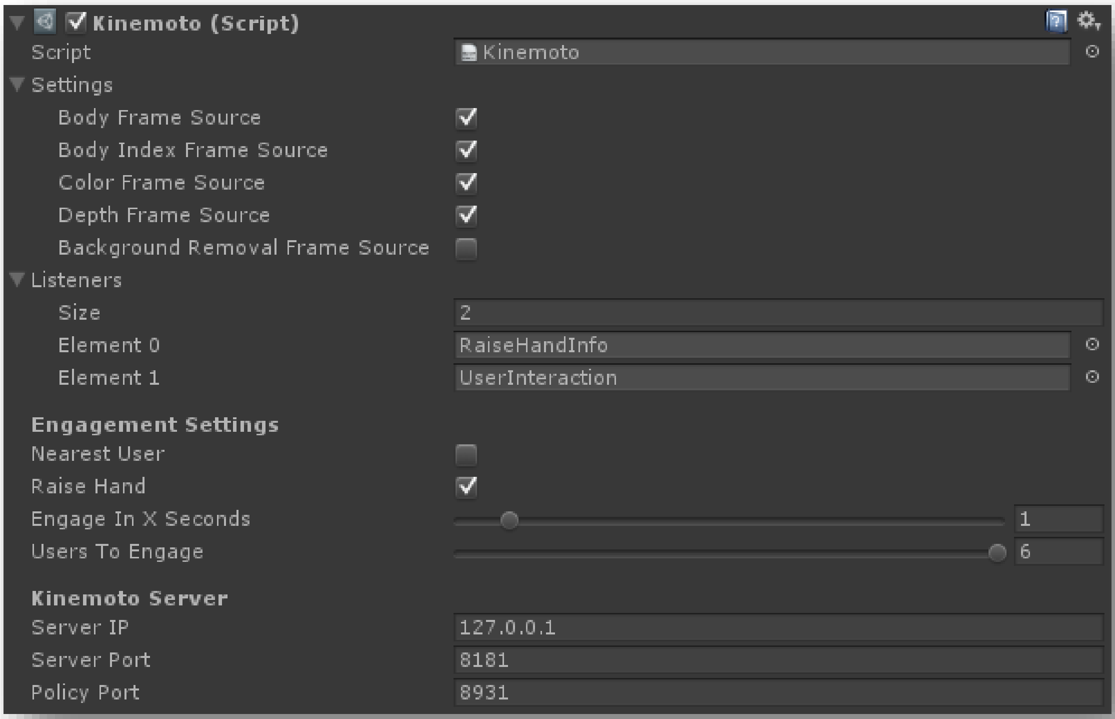
Now we need to get the KinemotoSDK Package into our Unity project. Double-click *KinemotoSDKvx.x.x.x*. This will load all KinemotoSDK files into Unity.

Note: The KinemotoSDK folder now contains a complete example overview with all possible streams and prefab already loaded and configured.

Step 4: Implement the base Kinemoto Script

Create an empty game object and add the Kinemoto script to it.

The Kinemoto Script on the GameObject will look like this.



✓ **Kinemoto (Script)**	
Script	Kinemoto
Settings	
Body Frame Source	✓
Body Index Frame Source	✓
Color Frame Source	✓
Depth Frame Source	✓
Background Removal Frame Source	
Listeners	
Size	2
Element 0	RaiseHandInfo
Element 1	UserInteraction
Engagement Settings	
Nearest User	
Raise Hand	✓
Engage In X Seconds	1
Users To Engage	6
Kinemoto Server	
Server IP	127.0.0.1
Server Port	8181
Policy Port	8931

There are quite some options visible but don't worry we will check them out one by one.

Settings:

Here you (now) have 5 options you can turn on or off, by default they'll all be unchecked.

- *Body Frame Source:* Retrieves the Body data off all users seen by the Kinect
- *Body Index Frame Source:* Retrieves the colored outlines of all users seen
- *Color Frame Source:* Retrieves a basic camera image seen by the Kinect
- *Depth Frame Source:* Retrieves the distance in mm off the Kinect for every pixel
- *Background Removal Frame Source*: Retrieves the camera image outlines of every seen user in a Green Screen like effect.

More info can be found in Chapter **Error! Reference source not found.**.

Listeners:

Change the size of Listeners to the amount of GameObjects Kinemoto needs to reach and drag and drop them into the fields. Next chapter we're going to implement the **UserInteraction** class so we can control a cursor on the screen. So to reach the *GameObject* with the **UserInteraction** class we would drop it onto a Listener in the Kinemoto class.

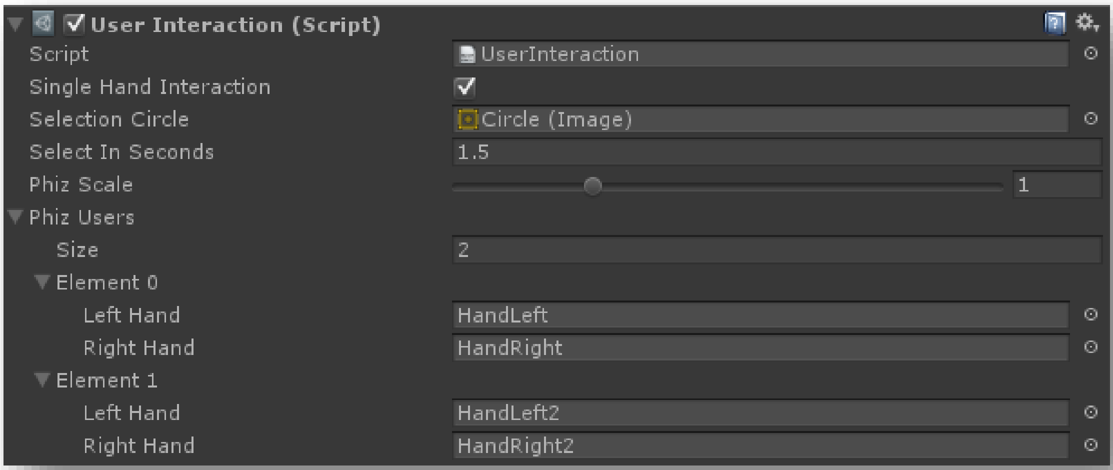
<u>Engagement Settings:</u>
In most cases you will need some way to tell which person in front of the Kinect is going to be interacting with the game. Kinemoto has two ways to determine which person that will be ("*Nearest User*" will be checked by default).

- *Nearest User*: the bodies closest to the Kinect will become engaged;
- *Raise Hand*: the first users that hold their hand above their head for as long as mentioned in: "Engage in X Second" will become engaged;
- *Engage in X Seconds*: the amount of time it takes for a user to become engaged;
- *Users to Engage*: the maximum amount of players that can play the game.

<u>Kinemoto Server</u>:
The information here should point towards the Kinemoto Server. Its best to keep the information as is.

4 Adding the UserInteraction

In this chapter we're going to add the *"UserInteraction"* script to our project. The *"UserInteraction"* controls the Phiz which allows you to easily add cursor interaction with your Hands. Phiz stands for **Ph**ysical **I**nteraction **Z**one.
To add the *"UserInteraction"*-script, create an empty GameObject that will handle your UserInteraction or locate an object you want to use for it and press **Add New Component**. Find the *"UserInteraction"*-script and add it. After adding the component it should look like this:



User Interaction (Script)

Script	UserInteraction
Single Hand Interaction	✓
Selection Circle	Circle (Image)
Select In Seconds	1.5
Phiz Scale	1
Phiz Users	
Size	2
Element 0	
Left Hand	HandLeft
Right Hand	HandRight
Element 1	
Left Hand	HandLeft2
Right Hand	HandRight2

As you may have noticed by now, the *UserInteraction* class is almost completely renewed. It is now possible to add all 6 Kinect users with their own Phiz and each hand will become a complete cursor from now on (which is done automatically by adding the "*CursorController*"-script in the background). A listener to the Kinemoto-Object will also be added automatically from now on. Let's go through all the options.

Single Hand Interaction:
If true, it allows only the most active hand to act as a cursor and disables the other hand. The most active hand is defined by the hand which you used to engage and can be switched by dropping it below your hip and raising the other hand.

Selection Circle:
A sprite image used to visualize the "selecting"-state of the cursor.

Select In Seconds:
When a cursor (*read: active hand*) is hovering on a button, this is the amount of time it will take before the button is selected. It is also used for animating the Selection Circle mentioned above.

Phiz Scale:
This option determines how much movement is needed to move the *GameObject* around the Screen. The higher the number the more movement is needed to move the cursor. The Phiz Scale caps out at 3.

Phiz Users:

- *Size*: The amount of users you want to have their own Phiz for. For every user you add here, you have to connect their own set of left and right hands (Note: if you want the same cursor for all hands, you can duplicate them so you have multiple *GameObjects*).
    - **Left Hand**: Add a *GameObject* to use as a left hand cursor here
    - **Right Hand**: Add a *GameObject* to use as a right hand cursor here

Note: The Phiz needs to have a Main Camera in the scene to function and the Phiz only works in Orthographic Camera mode for now.

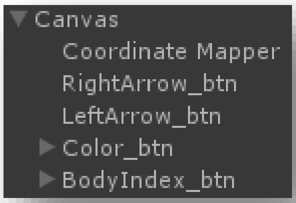
5 CursorController

As mentioned above in the *UserInteraction* section. Each hand is referred to as a cursor and automatically has the "*CursorController*"-script attached. This feature makes it possible for you as a developer to get a Kinect enabled, interactive menu / GUI system.

First, you have to create a **Canvas** with one or more buttons in it. Setup the buttons to your own needs and add an *OnClick* event. Your button should work completely with the mouse before heading on with the Kinect interaction. If you don't know how to setup the Canvas and button interactions, please read through the Unity documentation first.

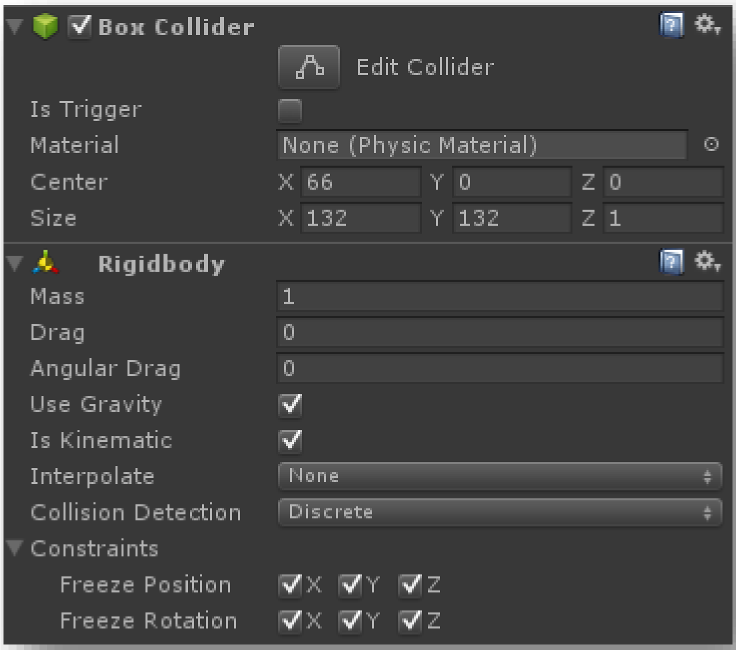
To be able to select your buttons with your hands follow the following steps:

1. Add "*_btn*" or "*_button*" to your button name in the **Hierarchy view** (e.g. "*yourButtonName_btn*").



```
▼ Canvas
    Coordinate Mapper
    RightArrow_btn
    LeftArrow_btn
  ▶ Color_btn
  ▶ BodyIndex_btn
```

2. Click "**Add Component**" -> "**Box Collider**"
3. Resize the collider to match the size of your button
4. Click "**Add Component**" -> "**Rigidbody**"
5. Check "**Use Gravity**" and "**Is Kinematic**"
6. Freeze all *Constraints*



Box Collider ✓

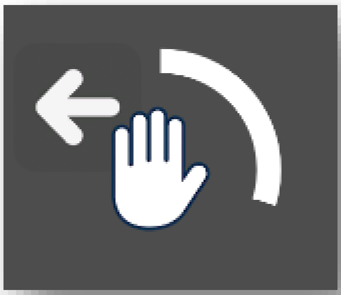
Edit Collider

Is Trigger	☐
Material	None (Physic Material)
Center	X 66 Y 0 Z 0
Size	X 132 Y 132 Z 1

Rigidbody

Mass	1
Drag	0
Angular Drag	0
Use Gravity	✓
Is Kinematic	✓
Interpolate	None
Collision Detection	Discrete
Constraints	
Freeze Position	✓X ✓Y ✓Z
Freeze Rotation	✓X ✓Y ✓Z

Now, when you move your hand cursor over the button, the selection circle will start to fill and after the “*Select In Seconds*” value it will click on the button like a normal mouse click.



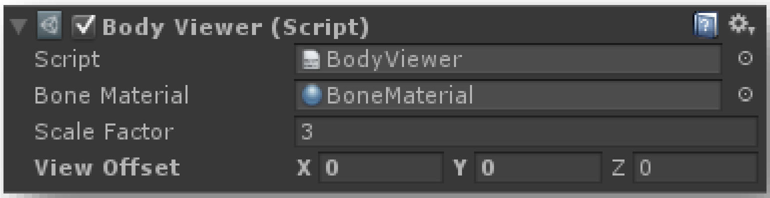
6 Kinect Stream Viewers

Since there are multiple Kinect Streams like the Body, BodyIndex, Color, Depth, etc., the KinemotoSDK has complete viewers for all of these implemented. Adding these viewers is really simple and doesn’t need any coding at all. To add the different viewers the following steps are all the same except for the Body Viewer.

Body Viewer

[Example scene: BodyViewer.unity]

Create an Empty GameObject and Add a Component to it name “*BodyViewer*” which should look like this.



Body Viewer (Script)

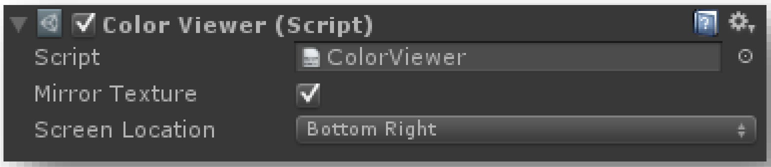
Script	BodyViewer
Bone Material	BoneMaterial
Scale Factor	3
View Offset	X 0 Y 0 Z 0

You can select a material for the bones, set the size for the *BodyViewer* with the Scale Factor and define an offset with a Vector3 for where the Body will be displayed.

Color / BodyIndex / Depth Viewer

[Example scenes: ColorViewer.unity / BodyIndexViewer.unity / DepthViewer.unity]

Find an object you want to add the *Viewer* to or create a *GameObject* like a Cube or a Quad (just make sure it has a renderer and your preferred size). Press **Add Component** and look for the Viewer to add it (*ColorViewer*, *BodyIndexViewer*, *DepthViewer*). Your new component should look like this (with the according Script of course):


Color Viewer (Script)

Script: ColorViewer

Mirror Texture: ✓

Screen Location: Bottom Right

You can use the check box for Mirror Texture to make sure that the viewer shows the image in the correct order. In other words this options makes sure the view works like a mirror.

[Example scene: MultiStreamGUIViewer.unity]
When you drag the viewer on the Main Camera, you can use the Screen Location to choose where you want the viewer to be located. The possible locations are: Bottom Right, Bottom Left, Top Right and Top Left.

Background Removal Viewer
[Example scene: BackgroundRemovalViewer.unity]
This Green screen like effect works exactly like the viewers mentioned above but it doesn't have any settings. It simply renders an outline of the ColorViewer. If you combine this with an image background you get the green screen effect.

7 Scripting references

As you may have noticed from setting up the *UserInteraction* or the different views. Everything in the KinemotoSDK works from the **Kinemoto** script.
So the first step to programming with the KinemotoSDK is pointing a listener in the Kinemoto class to the GameObject that contains the script you want to influence.
After that make sure you imported the Kinemoto library. You can do this by adding the following reference to your script:

```
Using KinemotoSDK.Kinect;
```

```
void Kinect_BodyFound(ulong TrackingID)
```

This function gets called as soon as a user walks into the Kinect. Every new Body gets a unique TrackingID which is sent over to this function.

```
void Kinect_BodyLost(ulong TrackingID)
```

This function gets called as soon as a user walks out of the Kinect's vision. Every Body lost this way won't get recognized by the Kinect anymore and this function gets called with the corresponding TrackingID.

```
void Kinect_EngagedUserFound(ulong TrackingID)
```

This function gets called as soon as a Body (recognized earlier by Kinect) carries out the engagement method defined by the Kinemoto class (*hand raised / nearest user*). Every engaged Body sends over its TrackingID to this function.

```
void Kinect_EngagedUserLost(ulong TrackingID)
```

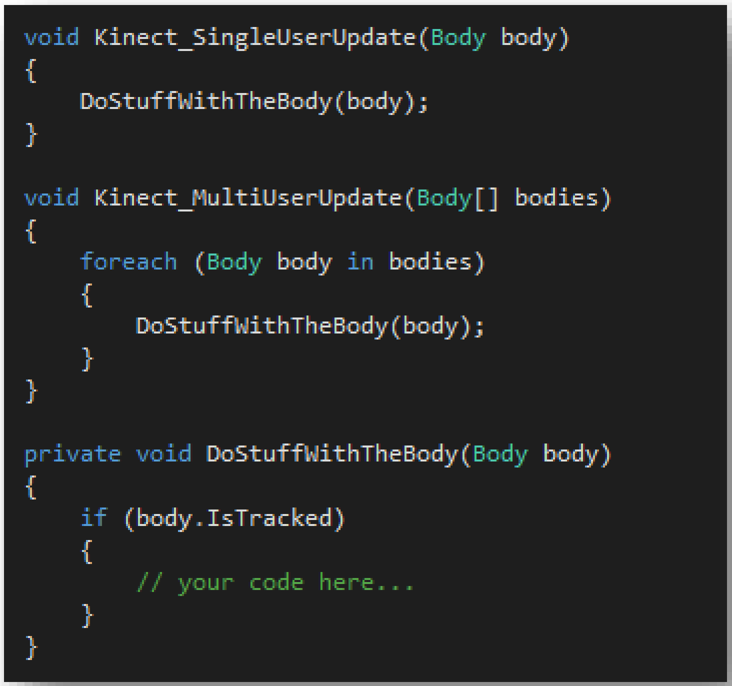
This function is quite similar to the BodyLost function as it gets called when Kinect is unable to recognize an EngagedUser. However not every Body visible by Kinect can be engaged. Every Body lost this way won't get recognized by the Kinect anymore and this function gets called with the corresponding TrackingID.

```
void Kinect_SingleUserUpdate(Body body)
```

If the Kinemoto Class was configured to only engage one user then the SingleUserUpdate will be called every frame. Therefore this function is useful to check for certain motions or gestures from your user to interact with your game. The Body sent to this function will always be the currently engaged user

```
void Kinect_MultiUserUpdate(Body[] Bodies)
```

If the Kinemoto Class was configured to engage two or more users then the MultiUserUpdate will be called every frame. Therefore this function is useful to check for certain motions or gestures from users to interact with your game. Even though this update is very similar to the SingleUserUpdate be sure to use the correct Body from the array with bodies or to loop through all of the bodies.



```
void Kinect_SingleUserUpdate(Body body)
{
    DoStuffWithTheBody(body);
}

void Kinect_MultiUserUpdate(Body[] bodies)
{
    foreach (Body body in bodies)
    {
        DoStuffWithTheBody(body);
    }
}

private void DoStuffWithTheBody(Body body)
{
    if (body.IsTracked)
    {
        // your code here...
    }
}
```

Working with the Body[] array:

The Body[] array contains all of the **Engaged Users** but in a randomly order. Be

careful with linking the motion from a Body in this array to actions in the game because the order of the bodies will get changed if an EngagedUser gets lost and engaged again.
Try to check each body their TrackingID with *Body.TrackingId* so you're certain the correct Body carries out the correct behavior in your game.

8 Implementing Joints, Handstates & Spacepoints

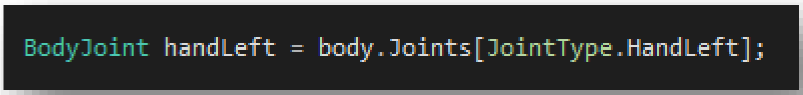
This chapter will explain how to use data from the Body. The data you can obtain from the Body is neatly organised in enums.

Body Joints:
The Body contains a dictionary of BodyJoint's called *Joints*. Aside from the type of joint it contains some more information about each joint like the *CameraSpacePoint* which is a *Vector3* containing the location of the joint in a 3D space and the *TrackingState* which checks if the joint is **Tracked**, **NotTracked** or **Inferred**.
The most important thing to know about the joints is of course what kind of joint it is and for that you can use the *JointType*-enum.

Joint Type:

0	SpineBase	12	HipLeft
1	SpineMid	13	KneeLeft
2	Neck	14	AnkleLeft
3	Head	15	FootLeft
4	ShoulderLeft	16	HipRight
5	ElbowLeft	17	KneeRight
6	WristLeft	18	AnkleRight
7	HandLeft	19	FootRight
8	ShoulderRight	20	SpineShoulder
9	ElbowRight	21	HandTipLeft
10	WristRight	22	ThumbLeft
11	HandRight	23	ThumbRight
24	ThumbRight		

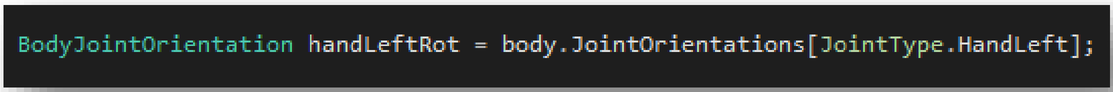
You can access the *BodyJoint* you need by referring to:



```
BodyJoint handLeft = body.Joints[JointType.HandLeft];
```

You can use the information about the *HandState* from the Body or the position of one of the joints with *CameraSpacePoint* to check if the player is moving their body in the required way.

Body Joint Orientation:
Aside from the *BodyJoints* the Body also contains *JointOrientations* which contains information about the rotation of a joint. This information is stored as a Quaternion. You can access this information with:



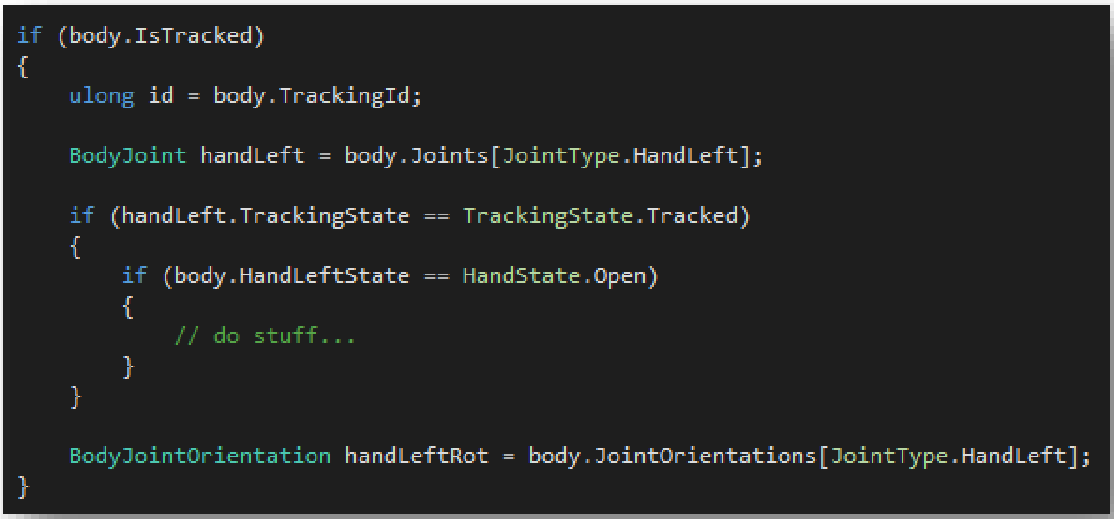
```
BodyJointOrientation handLeftRot = body.JointOrientations[JointType.HandLeft];
```

You can use this information to check which way a certain joint is rotated. This is quite interesting for the Spine, Neck & Head so you can track which way the player is looking to add an extra dimension to your game.

HandState:
With this Enum you can check what kind of Hand movement is detected by the Kinect. There are five different possibilities: Unknown, NotTracked, Open, Closed & Lasso. Just like the BodyJoints the hand state are members of the body object.

0	Unknown
1	NotTracked
2	Open
3	Closed
4	Lasso

The Body contains a HandRightState and a HandLeftState which you can use to check the state of the hands.



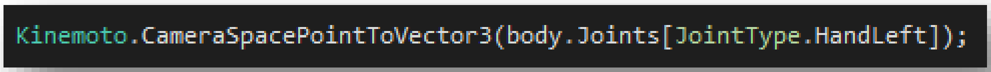
```
if (body.IsTracked)
{
    ulong id = body.TrackingId;

    BodyJoint handLeft = body.Joints[JointType.HandLeft];

    if (handLeft.TrackingState == TrackingState.Tracked)
    {
        if (body.HandLeftState == HandState.Open)
        {
            // do stuff...
        }
    }

    BodyJointOrientation handLeftRot = body.JointOrientations[JointType.HandLeft];
}
```

CameraSpacePoint:
The *CameraSpacePoint* was mentioned earlier along with the *BodyJoints*. This variable contains the location of the joint mapped to the world in Unity. It has an X, Y & Z float. You can pass it through a static Kinemoto method to convert it to a Vector3 with:



```
Kinemoto.CameraSpacePointToVector3(body.Joints[JointType.HandLeft]);
```

9 Extra Information (F.A.Q.)

If you have trouble implementing Kinemoto check out developer.kinemoto.com. Here you can find a series of video tutorials where you can follow the implementation step by step.

You can also send an e-mail to support@kinemoto.com or check out the questions below.

How can I keep the Kinemoto Server up to date?
Every time you start the Kinemoto Server it should automatically update to the latest version.
The Kinemoto SDK will be updated through the asset store in the client.

How do I program my own behaviors?
Use either the Kinect_SingleUserUpdate or the Kinect_MultiUserUpdate to check the states of your joints every frame. Check if the joints are in the required position and execute your own behavior if they are.

Why doesn't X work?
Make sure that KinemotoSDK.Kinect is implemented at the top of your script. And check the guide again to check if the required functions are correctly implemented.

Is the KinemotoSDK compatible with the older Kinect?
No, it isn't. Simply because it runs on completely different libraries.

Can I use the KinemotoSDK to build Standalone applications?
Yes you can! As long as the Kinemoto Server is running you should be fine.

10 Changelog

KinemotoSDK 2.2.0.0, 2015-08-26

[Minimal required Kinemoto Server version: v3.0.0.0]

- Added DepthStream support, including a sample scene and DepthViewer-Prefab
- Extended the KinemotoSDK Example with a fully Kinect controlled carousel through all possible Streams and views
- Bug fix: Stream settings will be send again when reloading the same scene (when the WebSocket connection already occurred)
- Added "CursorController.ButtonSelectedByUser"-property to determine which user selected the button
- Bug fix: When a user who is the only one allowed to control the scene leaves the view, the controls shift to another still existing user
- Disable cursors of users who are disengaged
- Added a public "WebSocketHandler.IsOpen"-property to check if a connection with the KinemotoServer is currently open
- Optimized some Images and according scripts
- Removed the "MenuHotCorner"-script and reprogrammed the KinemotoMenu to use it as with every other button ("..._btn")
- Added different Loading screen images to the Kinemoto WebPlayerTemplate

KinemotoSDK 2.1.6.0, 2015-07-07

[Minimal required Kinemoto Server version: v2.3.0.7]

- Update to newer KinemotoMenu
- Changed mechanics of the CursorController in UserInteraction. You can now simply add a new button with a name that ends
- with "_btn" or "_button", add a collider and you can handle all cursor interactions on the button itself
- Added a new EngagementScreen which is a scene you can use to let your user engage before the game begins
- Null reference error bug fix in the BodyViewer by opening the scene when already engaged

KinemotoSDK 2.1.5.0, 2015-06-17

- Implementation of a BackgroundRemovalStream which you can use to create Greenscreen effects
- Complete rewrite of the KinectSettings class to a singleton
- Added a Constants class for often referenced strings
- Changed the OnHandClosed, OnHandOpen and OnHandLasso events to use 3 new parameters
  - PlayerID (the position of the engaged Phiz user)
  - TrackingID (the real TrackingId of the body)
  - Operating side ("left" or "right")

    - When using a default viewer (eg. BodyIndex or Color) the stream will now automatically be enabled
    - Before a binary stream is used, a header will be sent containing the stream name and length of the byte array

KinemotoSDK 2.1.0.0b, 2015-06-05

- Completely revisited the EngagementHandler and the UserInteraction (Phiz)
    - New class structures
    - New logics
    - It is now possible to get "a real player" with EngagementHandler.EngagedPlayers (player1, player2, etc)
- Changed Menu script according to change in the Phiz
    - Menu works with button colliders instead of its own Phiz implementation

KinemotoSDK 2.0.4.0, 2015-05-19

- Added CoordinateMapper Class
    - Added MappedPosition to the Joint Data
- Added new eventlistener for when a connection has been made with the Kinemoto Server (WebSocketHandler.OnConnectionOpened)