# Unity to Arduino Connector (Proof of Concept)

## Getting Started:

In order to utilise an Arduino board and associated sensor data in a Unity Project, this serial connector project must be implemented. This guide will explain the set-up, use of and expansion of the Unity to Arduino Connector.

### Requirements:

1. Arduino Board
2. USB to Serial Cable
3. Unity Project
4. Unity to Arduino Scripts
5. Arduino Sketch
6. Serial Command Dependency Package.
7. Associated Software for each part (i.e. Arduino IDE, Unity and Visual Studio).

### Set Up:

1. Connect the Arduino Board to the PC using the USB to Serial Cable
2. Download the UnityArduinoConnector Project from GitHub (Found At: https://github.com/TimothyGAnderson/UnityArduinoConnector)
3. Open the OSC Test Project folder and navigate to Scenes, run the TestingScene.unity file.

### Using the Connector:

Figure : The Arduino IDE with upload button highlighted.

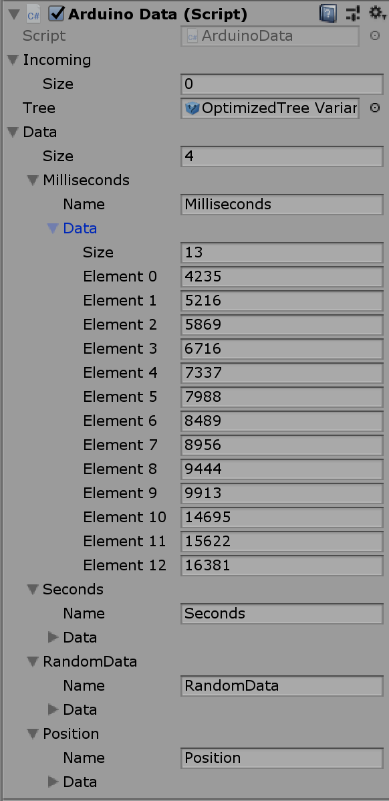
In order to use the connector, the Arduino must be plugged in and working. The Arduino Sketch must be loaded to the Arduino Board.

Once the Sketch has been uploaded then the Arduino Board is ready to use.

 If starting from a blank Unity project, you will need to create an empty game object and attach the ArduinoConnector.cs script to it.

Figure : The Arduino Connector Inspector Component

If successful, you will see this in the inspector in you editor. The Port field will automatically be assigned once you run project. The baud rate is the maximum rate of information that can be transferred, in this case the baud rate is 115200 bits per second, this has to be manually specified. D is the data. In order to fill this field, you must create a game object and attach the ArduinoData.cs script to the game object. Then select the game object you just created and drag it into the ‘D’ field on the Arduino Connector component.

The ArduinoData.cs file contains the data that can be requested from the Arduino Board. The Fields in the inspector will be filled automatically as data is received by Unity.

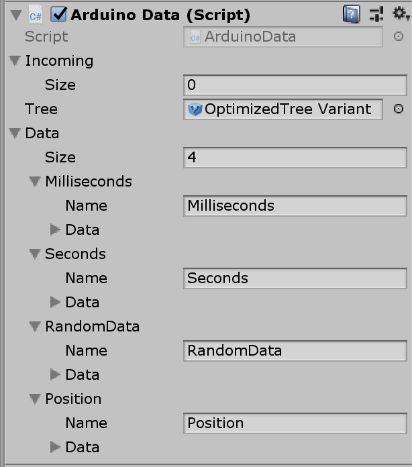


Figure 3a: The Arduino Data Inspector Component with no data from the Arduino Board.

Figure b: The Arduino Data inspector component with fields that have been filled from the board.

In the example above, the number of Milliseconds has been echoed and has added an element to the Millisecond list. This happens with each of the lists contained within the ArduinoData.cs script.

## Extending the Connector:

By virtue of design, the connector is able to be easily modified to take send data from the Arduino Board to Unity. This can include sensor data from the various sensors available for Arduino Boards, or even sending commands *to* Arduino boards for controlling components such as lights and valves.

In order to extend the Connector, several things have to happen.

First: The command that the Arduino Board receives has to be added to the list of commands that the board can recognise. This is done in the void Setup() function call.

void setup() {

Serial.begin(115200);

while(!Serial);

sCmd.addCommand("PING", pingHandler);

sCmd.addCommand("ECHO", echoHandler);

sCmd.addCommand("SEND", sendHandler);

sCmd.addCommand("SENDALL", sendallHandler);

//Add your command here.

sCmd.addCommand(“IRCOMM”, irHandler);

sCmd.addDefaultHandler(unrecognized);

}

Secondly: You will need to implement the code for the custom event handler that you will be using with your new command. In the example above, a new command called ‘IRCOMM’ has been added, as well as a new event handler called ‘irHandler’. In Unity, upon pressing a specific key (for arguments sake say ‘I’) this will send the IRCOMM command to the Arduino board which will then respond with the code contained within the irHandler event handler.

An example of this would be:

void irHandler() {

//Add Your Code Here. For example if the IRSensor reads no IRLight, then turn an LED bulb off, otherwise the LED bulb is on. Assume lights and pins have been correctly setup and the modes have been defined in the setup() function.

val analogRead(0);

Serial.println(val);

if(val < 1000){

digitalWrite(LED,LOW);

//This is what can be sent to Unity, this is just a simple echo example. But can be modified for other data.

Serial.println(“Light Off”);

}else{

digitalWrite(LED,HIGH);

Serial.println(“Light On”);

}

Delay(10);

}

Once this is done, the only part left is to modify the Unity scripts to send commands to the Arduino board, these commands will then return data sent by the Arduino board for use in Unity.

In the ArduinoConnector.cs script, find the Update function and add in a way to send the command to the Arduino. If wanting to call it via keyboard it can be done like this below.  
  
if (Input.GetKeyDown(KeyCode.I))

{

if (stream != null)

{

if (stream.IsOpen)

{

WriteToArduino("IRCOMM");

}

}

}

This sends the command to the Arduino board to interact with the IR Sensor (for example). This is by no means the only way to send commands to the Arduino board, these commands can be sent by OnTriggerEnter()/OnTriggerExit() calls (when objects collide), or when game objects are added or deleted.

In order to read data from the Arduino Board, it gets a little trickier. In order to do this, we are required to alter the data node. The data node takes the data from the Arduino board as a string (s), this string is run through several loops in order to parse the data into correct types that can be used by Unity.

public List<float> ConvertToFloat()

{

List<float> list = new List<float>();

foreach(string s in data)

{

float f = 0;

if (float.TryParse(s, out f))

{

list.Add(f);

}

else

Debug.LogError("Unable to parse value in " + name);

}

return list;

}

The above function creates an empty list of floating point numbers. Then iterates throughout each string in the data string that has been output by the Arduino board. If the string (s) can be parsed to a floating point number, then it is added to the list, and the list is returned. Similar code is used for parsing the data to an integer.

The other function included in the ArduinoData.cs file is ConvertToVector3() function. This converts the data string s into a series of Vector3 Coordinates that can be used to place objects based on the Vector3 (x,y,z) system. This is achieved by utilising the same methodology as the previous two functions but also splits the string whenever there is a ‘,’ character. This denotes the change from x pos->y pos -> z pos.

In order to utilise different data types, you can use similar methods to parse the data that you need. Create a list of the type of data you need (int, char, string, Vector3, float, etc), then parse the data and add it to a list, then return the list. This list can then be accessed and manipulated by other C# scripts and Unity objects.