

Unity3D Documentation

Links to Basic unity tutorial

Unity Basics: <https://www.youtube.com/watch?v=UWXVbZgXNA4>

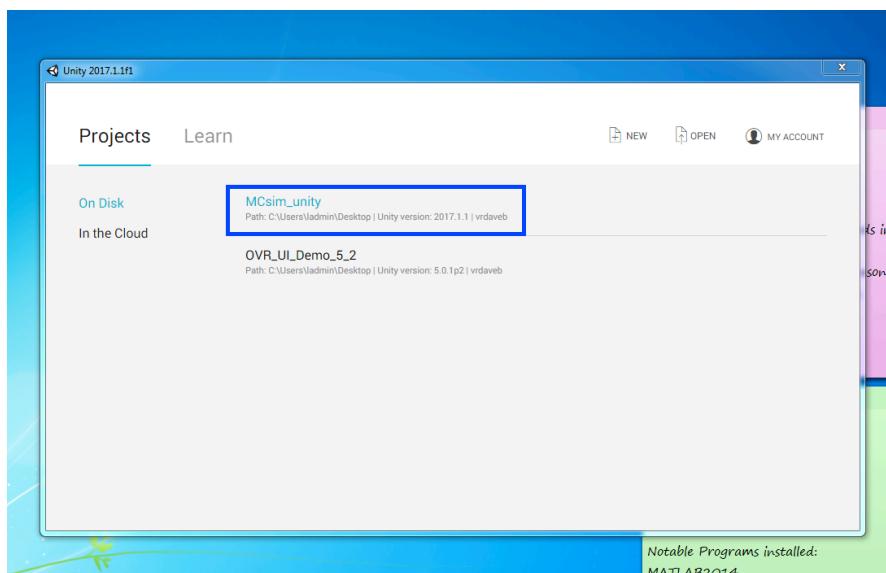
C# scripting in unity: <https://www.youtube.com/watch?v=tzq-DJeNIrU&t=525s>

Starting Unity

1. Open the **unity.exe** application by clicking the symbol like in the image below.



2. Open the project **MCsim_unity**:



3. The scene (**race_track_lane**) should now be opened and you will see a window with all the *GameObjects* at the left pane and the game view in the middle. **Press the Play button to start** the game. Make sure that the VR-headset is **plugged in** with USB-3.0 and the HDMI-cable. **OBS! Do not start the game before the raspberry pi's ROSCORE server has started!**

Author: Thibault Helle

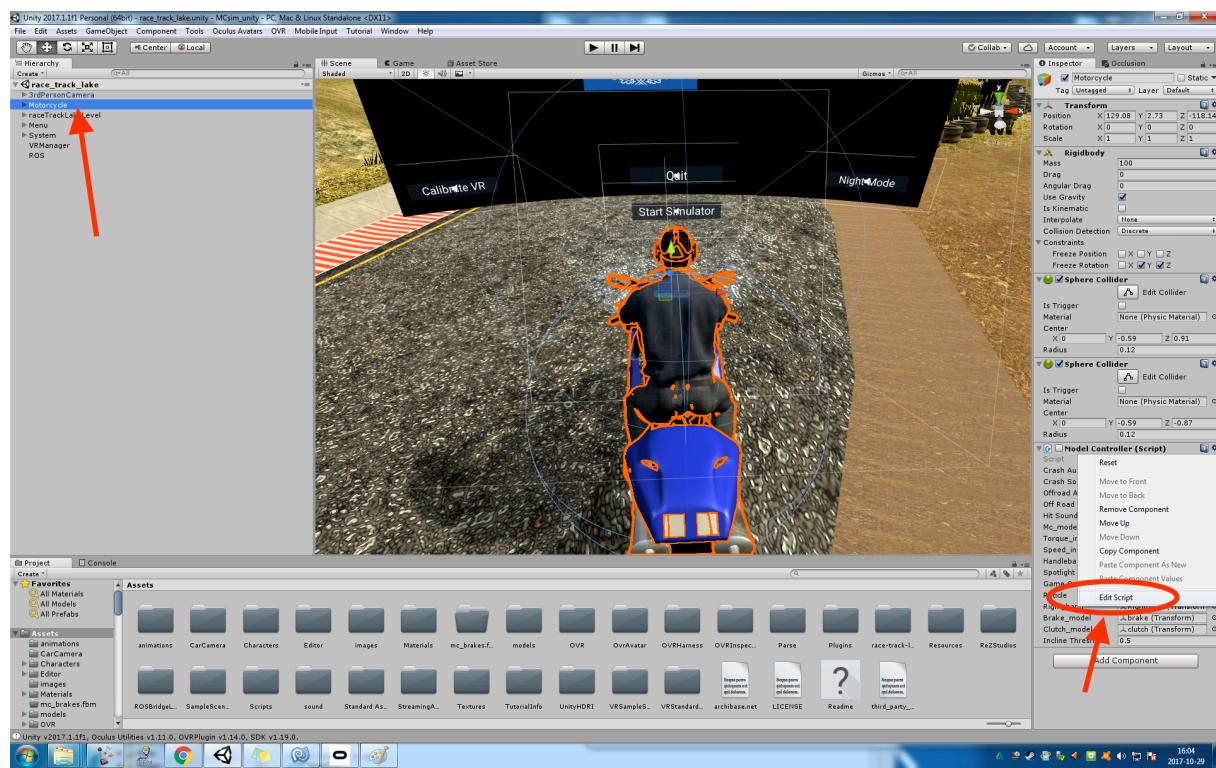


4. To stop the game, just press the **Stop button** at the same place.

Edit motorcycle behavior in C#

All C#-scripts must be attached to a GameObject, i.e. the “**Motorcycle**” game-object has the most important script called **modelController.cs**. This is where most of the dynamics happens and where the public state variables of the model are (x, y, z, rotations, throttle, onRoad, speed etc...) and can be accessed from other scripts as they are public. Edit the script by **right-clicking** on the **attached** script’s in the Game Object’s **inspector**.

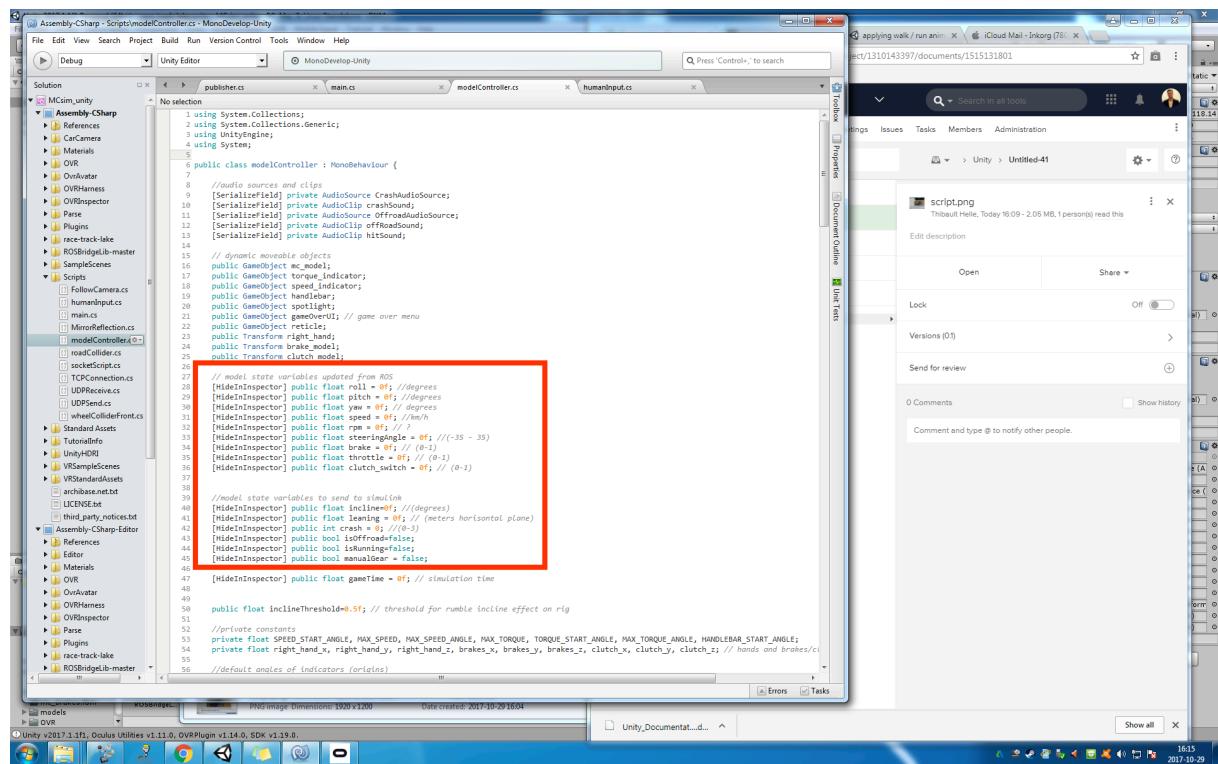
Author: Thibault Helle



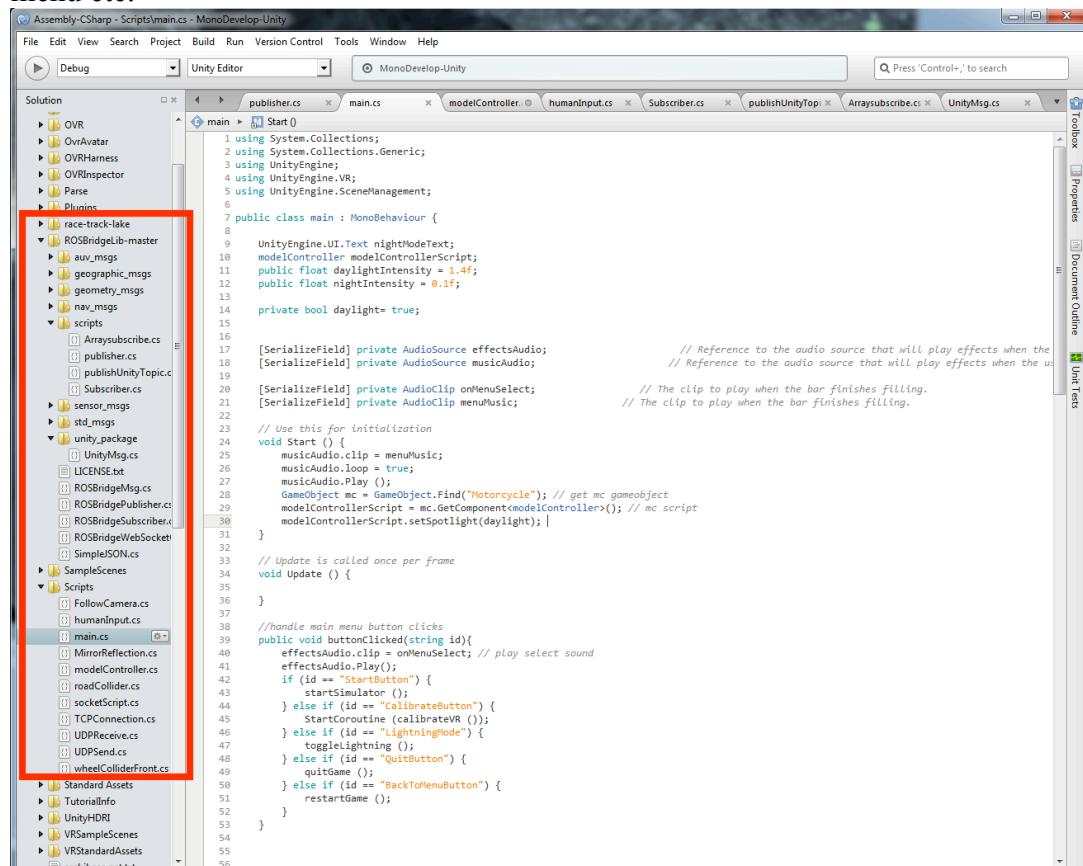
Another application, **MonoDevelop** will open where you can open and debug the code live, even when the game is running. In the **modelController.cs** as shown below, you can find all the variables that is important.

Variables needs to be **public** for it to be read/written by the ROS scripts (later in this guide). If you don't put [HideInInspector] in front of the code, you will be able to dynamically edit the constant value in the GUI inspector in Unity. The void **Start()** function is the first function to be started when the game starts, and **Update()** is run every frame update, if the script is enabled (checkbox in inspector).

Author: Thibault Helle

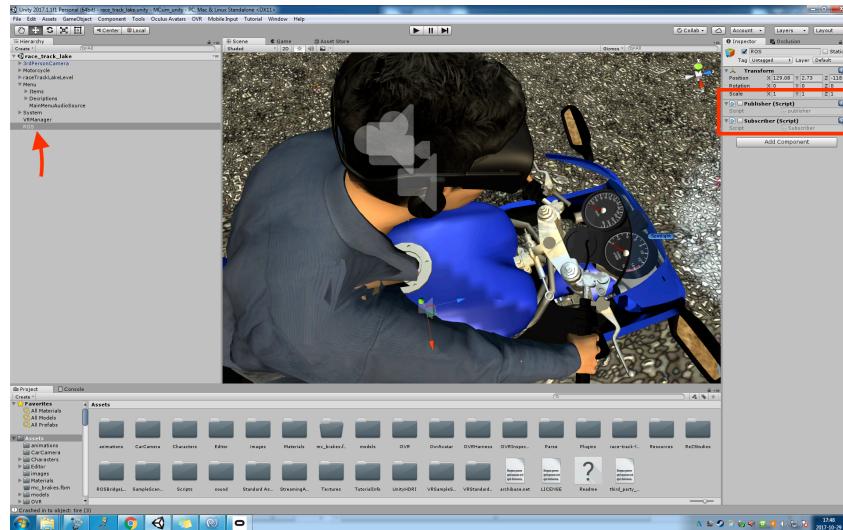


You can find the other script here in the folders as shown in the image below. The main.cs script handles the main events, such as pressing button at the menu, playing music, hiding menu etc.

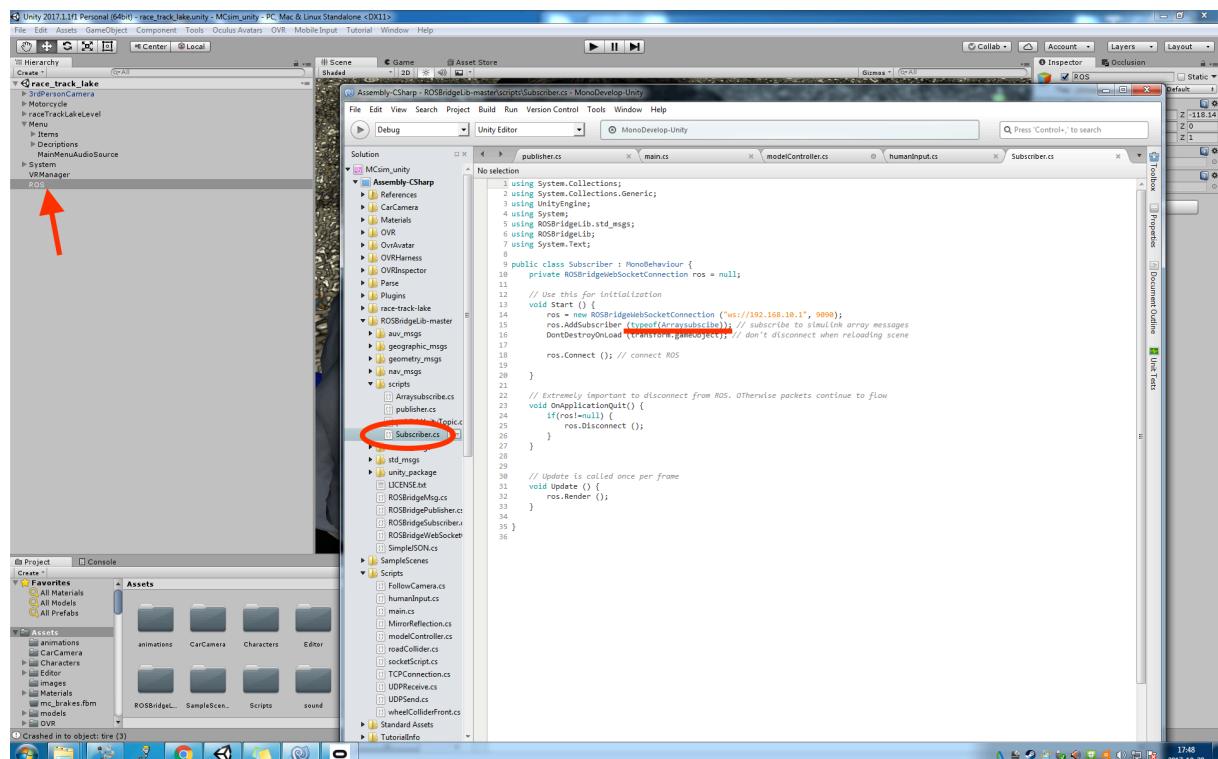


Edit incoming/outgoing data

Click on the ROS gameobject to see the attached ROS scripts like in the image below.

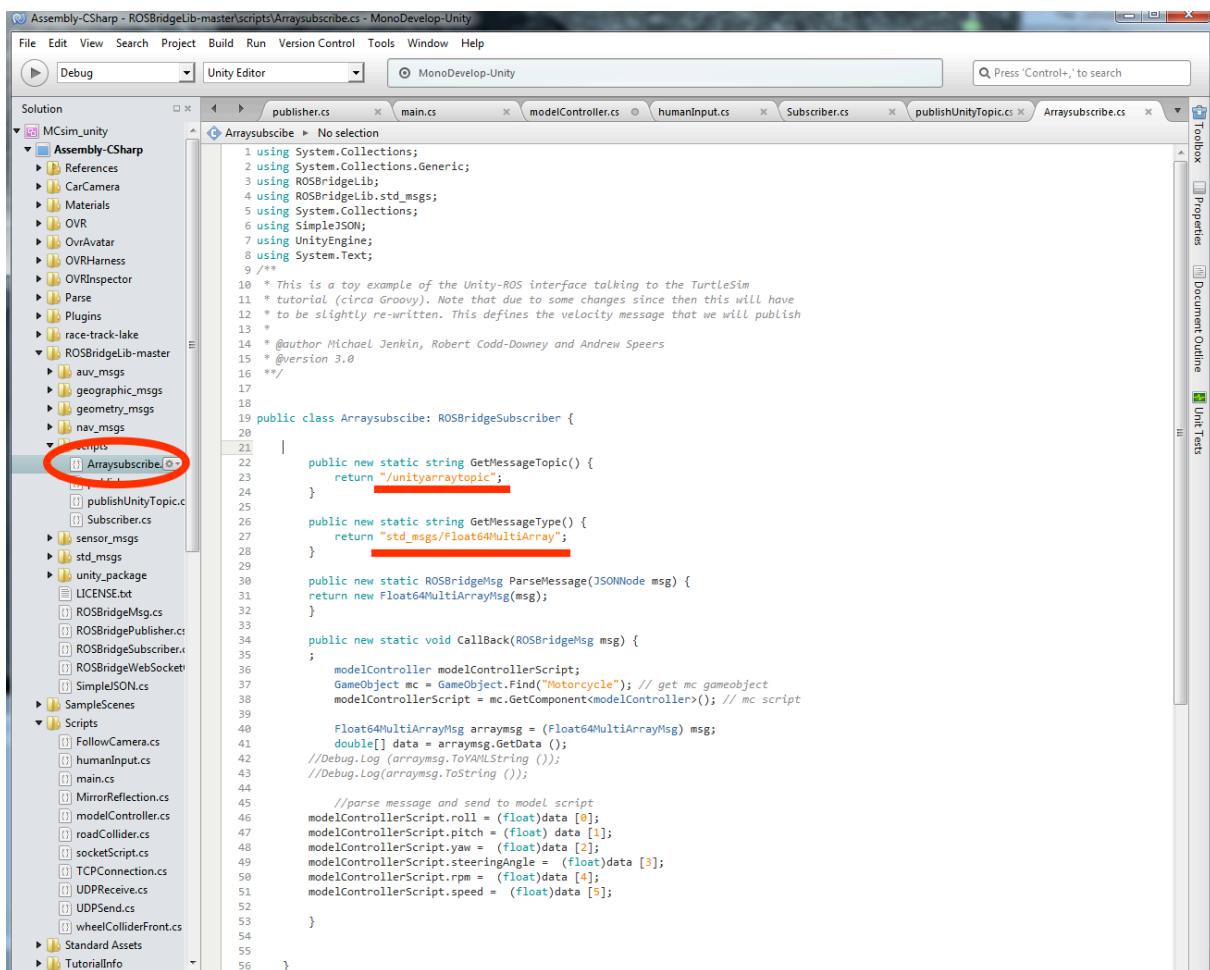


In the **Subscriber.cs** script we can add topics to be subscribed to, and of what type of message it is. Like in the image below, we are adding a subscription to **Arrayssubscribe**, which is a class type.



To see or edit the type, open up the script of the same name, **Arrayssubscribe.cs**. In the image below, you can see the topic name **"/unityarraytopic"** and the message type **"std_msgs/Float64MultiArray"**, which is a standard ROS message type.

Author: Thibault Helle



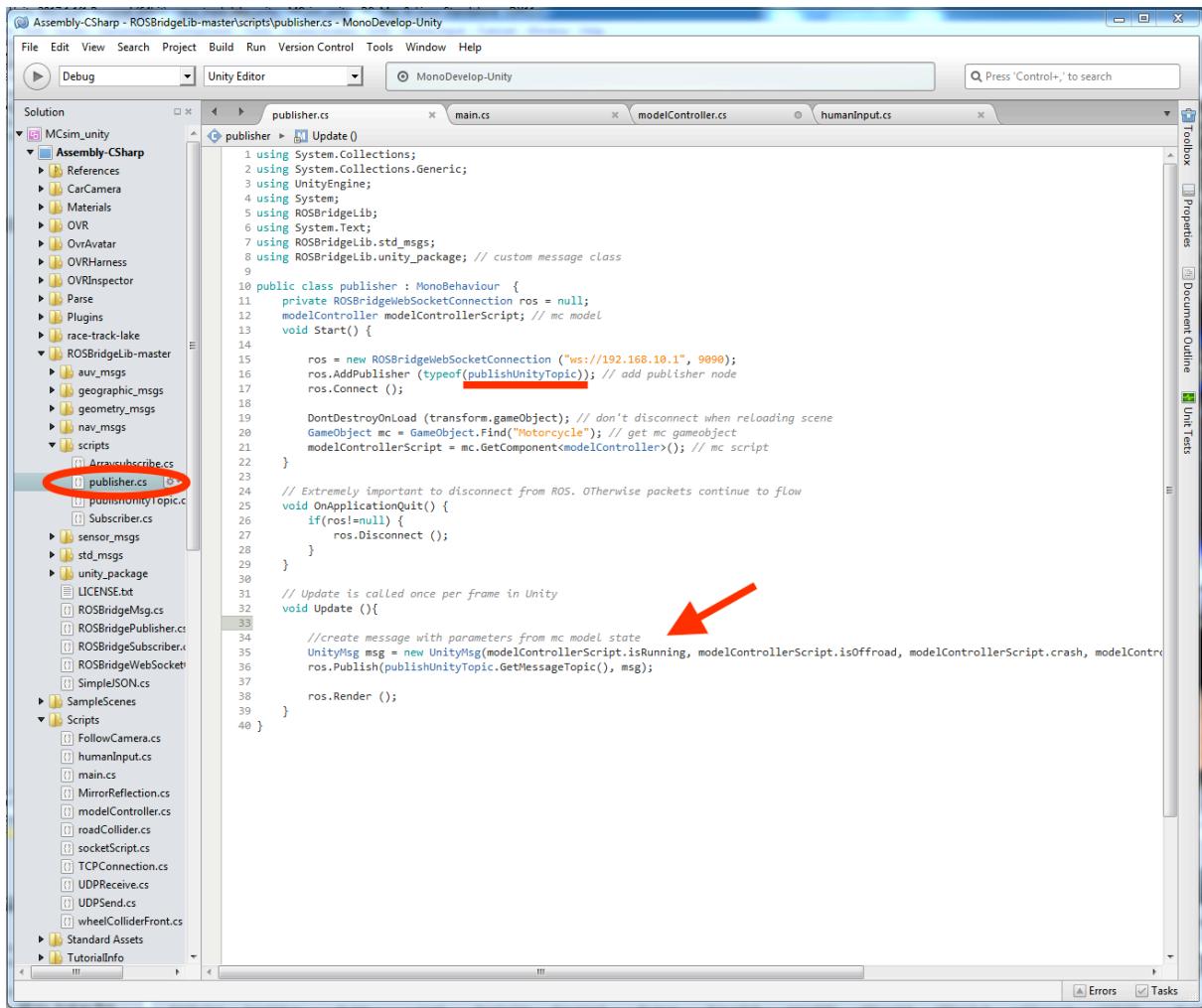
The screenshot shows the MonoDevelop-Unity interface with the project 'Assembly-CSharp' open. The 'Scripts' folder contains several C# files, including 'Arraysubscribe.cs'. This file is currently selected and displayed in the code editor. The code is a ROSBridgeSubscriber implementation for a topic named '/unityarraytopic'. It uses a 'Float64MultiArray' message type and parses the incoming message to update a 'Motorcycle' GameObject's state variables via a 'modelControllerScript' component.

```
1 using System.Collections;
2 using System.Collections.Generic;
3 using ROSBridgeLib;
4 using ROSBridgeLib.std_msgs;
5 using System.Collections;
6 using SimpleJSON;
7 using UnityEngine;
8 using System.Text;
9 /**
10 * This is a toy example of the Unity-ROS interface talking to the TurtleSim
11 * tutorial (circa Groovy). Note that due to some changes since then this will have
12 * to be slightly re-written. This defines the velocity message that we will publish
13 *
14 * @author Michael Jenkin, Robert Codd-Downey and Andrew Speers
15 * @version 3.0
16 */
17
18
19 public class Arraysubscribe: ROSBridgeSubscriber {
20
21     public new static string GetMessageTopic() {
22         return "/unityarraytopic";
23     }
24
25     public new static string GetMessageType() {
26         return "std_msgs/Float64MultiArray";
27     }
28
29     public new static ROSBridgeMsg ParseMessage(JSONNode msg) {
30         return new Float64MultiArrayMsg(msg);
31     }
32
33     public new static void CallBack(ROSBridgeMsg msg) {
34         ;
35         modelController modelControllerScript;
36         GameObject mc = GameObject.Find("Motorcycle"); // get mc gameobject
37         modelControllerScript = mc.GetComponent<modelController>(); // mc script
38
39         Float64MultiArrayMsg arraymsg = (Float64MultiArrayMsg) msg;
40         double[] data = arraymsg.GetData();
41         //Debug.Log (arraymsg.ToString ());
42         //Debug.Log (arraymsg.ToString ());
43
44         //parse message and send to model script
45         modelControllerScript.roll = (float) data [0];
46         modelControllerScript.pitch = (float) data [1];
47         modelControllerScript.yaw = (float) data [2];
48         modelControllerScript.steeringAngle = (float) data [3];
49         modelControllerScript.rpm = (float) data [4];
50         modelControllerScript.speed = (float) data [5];
51
52     }
53
54 }
55
56 }
```

We then update the GameObject “Motorcycle” state variables to by accessing the script as seen above in the code. Where all data from the incoming messages are passed to the corresponding variable to the model script.

Author: Thibault Helle

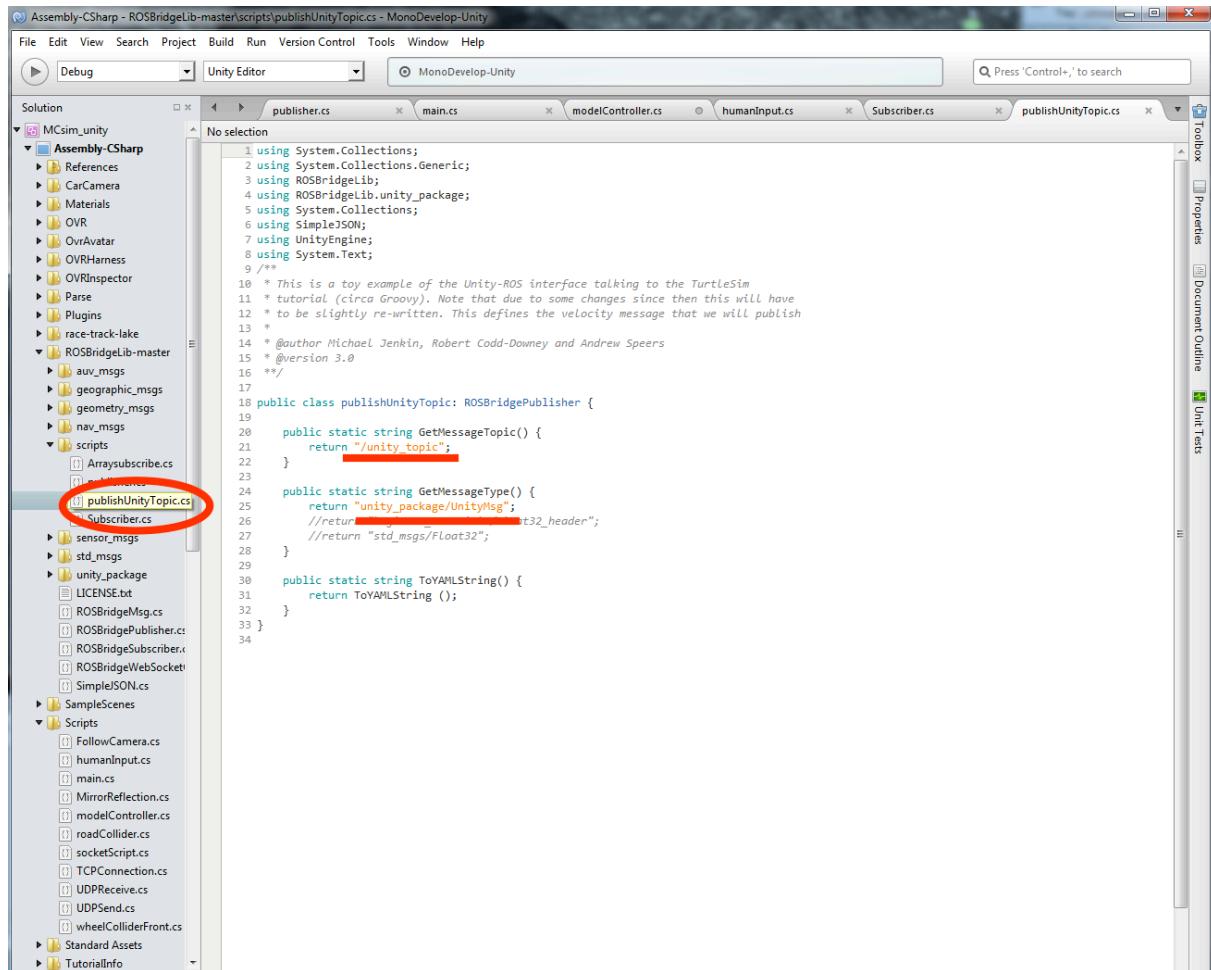
Now for publishing a message using ROS, we use the **publisher.cs** script to add publishers with a certain type. In this script below we have a topic type of **publishUnityTopic**. In the `Update()` function you can edit what message type the topic needs and pass what variables to send from the running modelcontroller script.



Here (above image) we send a custom made message with variable names (instead of an array) which is easier to handle. To see the message class declaration and to add/remove data fields, open **UnityMsg.cs** script in the **unity package** folder.

Author: Thibault Helle

Now to see the topic declaration type, open **publishUnityTopic.cs**. Here you define what message type that will be sent.



The screenshot shows the MonoDevelop-Unity IDE interface. The title bar reads "Assembly-CSharp - ROSBridgeLib-master\scripts\publishUnityTopic.cs - MonoDevelop-Unity". The menu bar includes File, Edit, View, Search, Project, Build, Run, Version Control, Tools, Window, Help. The toolbar has Debug, Unity Editor, and MonoDevelop-Unity buttons. A search bar says "Press 'Control+, to search". The main area shows a code editor with the following content:

```
1 using System.Collections;
2 using System.Collections.Generic;
3 using ROSBridgeLib;
4 using ROSBridgeLib.unity_package;
5 using System.Collections;
6 using SimpleJSON;
7 using UnityEngine;
8 using System.Text;
9 /*
10 * This is a toy example of the Unity-ROS interface talking to the TurtleSim
11 * tutorial (circa Groovy). Note that due to some changes since then this will have
12 * to be slightly re-written. This defines the velocity message that we will publish
13 *
14 * @author Michael Jenkin, Robert Codd-Downey and Andrew Speers
15 * @version 3.0
16 */
17
18 public class publishUnityTopic: ROSBridgePublisher {
19
20     public static string GetMessageTopic() {
21         return "/unity_topic";
22     }
23
24     public static string GetMessageType() {
25         return "unity_package/UnityMsg";
26         //return "std_msgs/Float32";
27         //return "std_msgs/Float32_header";
28     }
29
30     public static string ToYAMLString() {
31         return ToYAMLString();
32     }
33 }
```

The project tree on the left shows the solution structure:

- MCsim_unity
- Assembly-CSharp
 - References
 - CarCamera
 - Materials
 - OVR
 - OvrAvatar
 - OVRHarness
 - OVRInspector
 - Parse
 - Plugins
 - race-track-lake
 - ROSBridgeLib-master
 - auv_msgs
 - geographic_msgs
 - geometry_msgs
 - nav_msgs
 - scripts
 - Arraysubscribe.cs
 - publishUnityTopic.cs
 - Subscriber.cs
 - sensor_msgs
 - std_msgs
 - unity_package
 - LICENSE.txt
 - ROSBridgeMsg.cs
 - ROSBridgePublisher.cs
 - ROSBridgeSubscriber.cs
 - ROSBridgeWebSocket.cs
 - SimpleJSON.cs
 - SampleScenes
 - Scripts
 - FollowCamera.cs
 - humanInput.cs
 - main.cs
 - MirrorReflection.cs
 - modelController.cs
 - roadCollider.cs
 - socketScript.cs
 - TCPConnection.cs
 - UDPReceive.cs
 - UDPSend.cs
 - wheelColliderFront.cs
 - Standard Assets
 - TutorialInfo

If you have other questions on how to add/edit things just ask me! ☺