AirConsole Unity Plugin

The Unity-Plugin is a C# wrapper for the AirConsole Javascript API. With the plugin, Unity serves you the needed screen.html file inside the Editor and creates a screen.html file after an WebGL build export.

> **Note:** The plugin comes with an embedded webserver / websocket-server for the communication between the AirConsole backend and the Unity-Editor. You don't need to install any other webserver or services. The plugin was tested with Unity 5.1 for the Windows and OSX-Editor.

Download

Download from Github

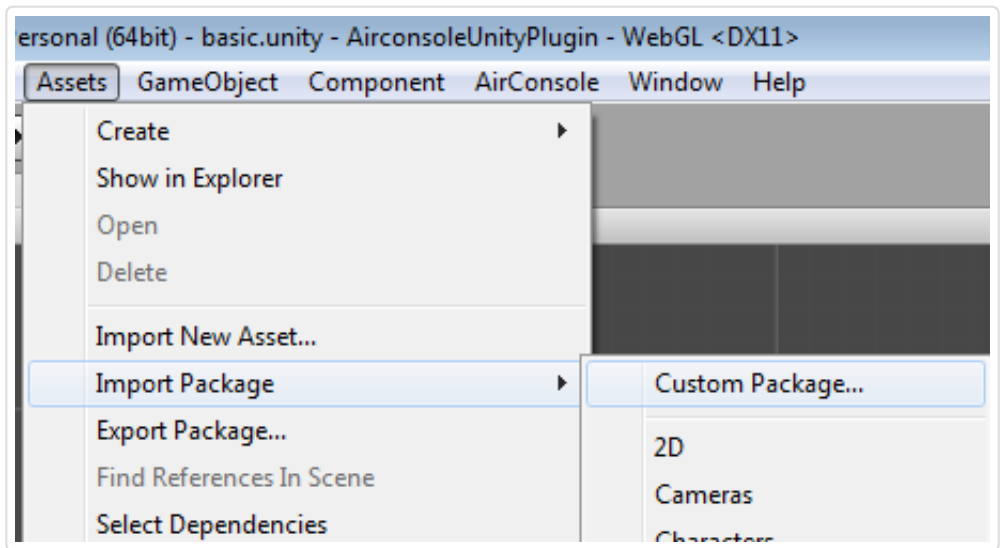
Download the newest AirConsole-Unity-Plugin on our GitHub repository (http://github.com/AirConsole/airconsole-unity-plugin). You can find the newest ".unitypackage" file in the "Builds" folder.

Download from Assetstore

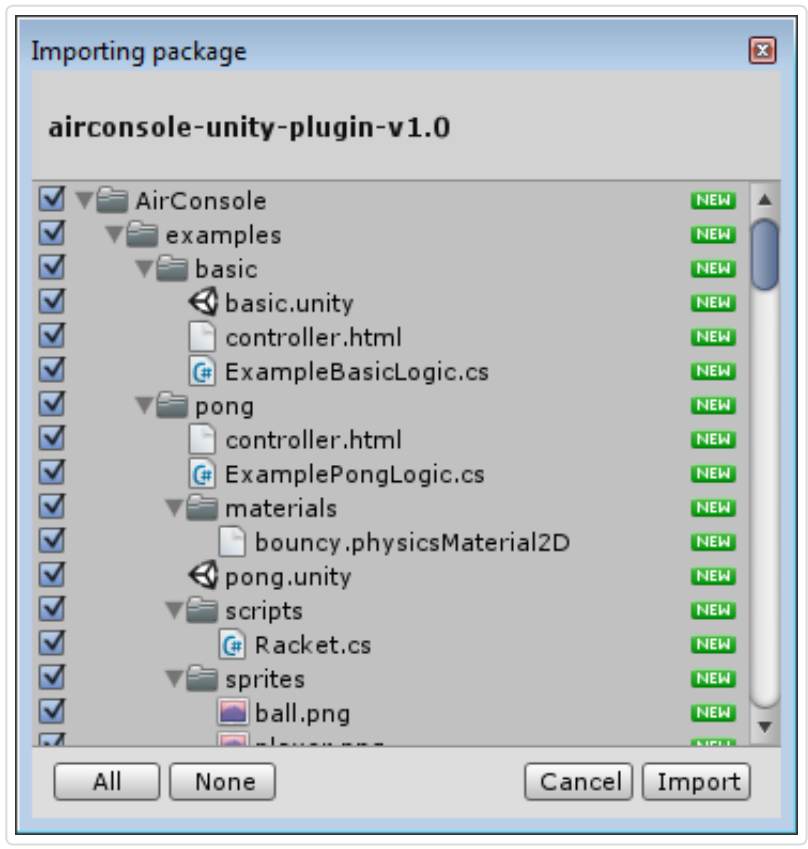
Comming soon!

Installation

To import the plugin, click on `Assets > Import Package -> Custom Package` and choose the downloaded `*.unitypackage` file.

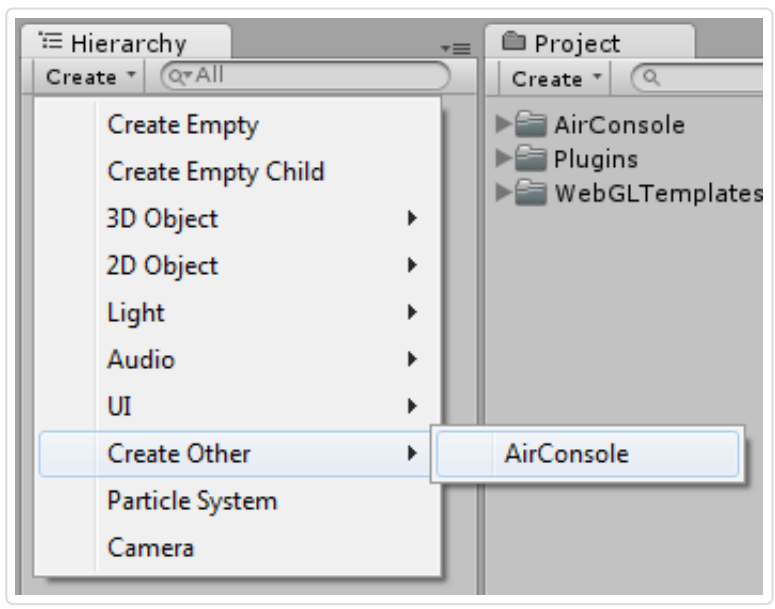


Make sure that all assets are selected and click on "Import".



Instantiate AirConsole object

To set up the communication inside a scene, you have to instantiate the AirConsole object. In the Hierarchy window, click on `Create > Create Other > AirConsole`.



The AirConsole object offers you the following options:

Option	Description
Browser Start Mode	As soon as you press the “Play” button in Unity, your default browser will open the AirConsole simulator in the chosen mode.
Controller Html	Link your controller.html file to this inspector variable, otherwise you can’t start the AirConsole simulator and you will get an error message. (Note: if your controller needs additional files like images or fonts, put them in the folder “Assets/WebGLTemplates/AirConsole”)
Auto Scale Canvas	This option is enabled by default and scales your final webgl canvas automatically to the window size. The aspect ratio will be kept during the scaling.

Note: the AirConsole object survives a scene change by default, so you don't have to worry about the connections and device states when changing a level inside Unity)

Quick example

Here is a simple example where the controller sends "How are you?" to the screen and the screen replies with "Full of pixels!"

Include the AirConsole Javascript API in your controller.html file:

```
<script type="text/javascript" src="http://www.airconsole.com/api/airconsole-1.2.js"></sc
ript>
```

/controller.html

```
var air_console = new AirConsole();
air_console.message(AirConsole.SCREEN, "How are you?");
```

Unity

To be able to receive a message in Unity, create a new Unity MonoBehaviour class and include these namespaces:

```
using NDream.AirConsole;
using Newtonsoft.Json.Linq;
```

Add an event handler in the Start method of the MonoBehaviour class:

```
void Start() {
    AirConsole.instance.onMessage += OnMessage;
}

void OnMessage(int from, JToken data) {
    AirConsole.instance.Message(from, "Full of pixels!");
}
```

Note: make sure to only send JSON supported data types between the controller.html file and Unity. The AirConsole-Unity-Plugin exchanges data via a modifed version of the popular JSON.NET library. The plugin delivers received data inside Unity as JToken objects. You can check the documentation of JSON.NET here: **http://www.newtonsoft.com/json/help/html/N_Newtonsoft_Json_Linq.htm (http://www.newtonsoft.com/json/help/html/N_Newtonsoft_Json_Linq.htm)**

JToken parse example

When you send an object like this one on the controller.html side:

```
// Javascript
var message = {
  'action': 'move',
  'info': { 'amount': 5, 'torque': 234.8 }
};

air_console.message(AirConsole.SCREEN, message);
```

You can parse the object in Unity like this:

```
// C#
void OnMessage(int from, JToken data) {
  string action = (string) data["action"];
  int amount = (int) data["info"]["amount"];
  float torque = (float) data["info"]["torque"];
}
```

Note: when you receive JToken data in Unity, you have to cast the object to get the right data type.

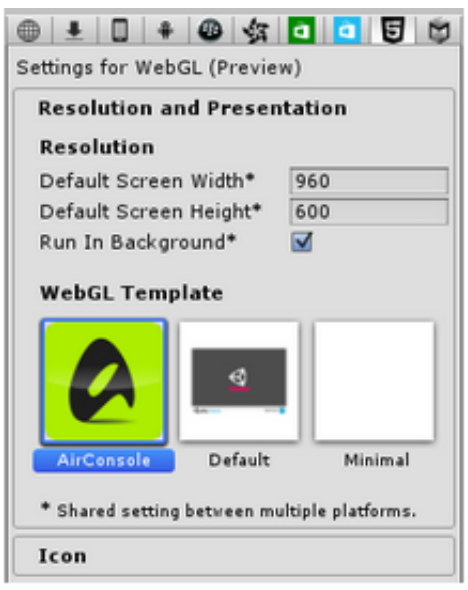
To send the same data from Unity you can use dynamic vars:

```
// C#
var message = new {
    action = "move",
    info = new { amount = 5, torque = 234.8f }
} ;

AirConsole.instance.Message(device_id, message)
```

Test your game

To test your game, just hit the “Play” button inside the Unity-Editor. If you want to test the final WebGL-Build, choose first the AirConsole Template in the Playersettings:



Now you can export your Unity project in the Build-Settings (choose “Build” instead of the “Build and run” button). As soon as your build is exported, the button “Open Exported Port” on the AirConsole object in your scene will appear. By pressing this button, the AirConsole simulator will start with your final build.

Note: the "Open Exported Port" button is only available as long as you don’t change the last target location of your exported build

IMPORTANT: before you can build the WebGL port, you have to start the game at least once in the Editor to generate all important metafiles for the final build.

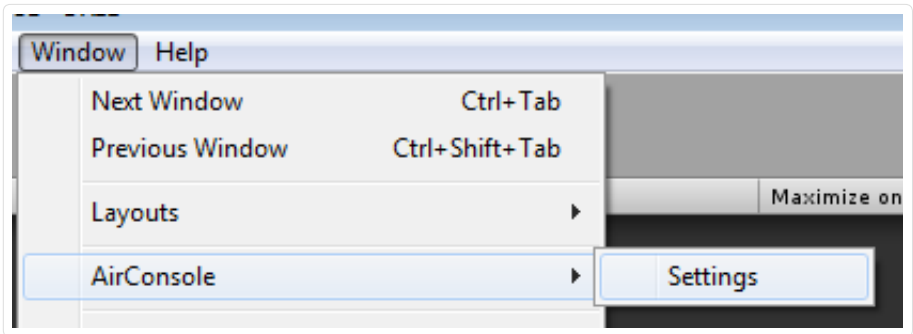
Examples

- AirConsole/Examples/Basic/basic.scene:
  In this example all possible API functions are listed & used.

- AirConsole/Examples/Pong/pong.scene:
  This example contains a simple pong game with a specific controller.

Settings

You can open the settings window by clicking on the "Settings" button on the inspector of the AirConsole object or by navigating over the menu bar `AirConsole > Settings` .



In the "Settings" window, the ports for the internal web socket and web server can be changed, should the default ports already be occupied.

> **Note:** These ports are only needed for the AirConsole simulator inside the Unity Editor. The final WebGL port will not use any of these ports

You can also stop and restart the internal running webserver if the AirConsole simulator should not work properly." ersetzen mit "You can stop and restart the internal web server in case the AirConsole simulator isn't working properly