Unity Asset Documentation

Thank you for your interest in our assets!

Please note, that our assets always include PDF-Files as this is a requirement from Unity.

We have an up-to-date online documentation that is much more convenient to use, available here.

About this asset

Support

Thank you for buying our asset.

Unfortunately sometimes customers leave negative ratings and complain about things not working without contacting us so we can not help or fix the issue. We want to provide good support and have our customers having a great time developing their content, so:

If you happen to encounter any issues, please write an email to assets@firesplash.de and we will be happy to help!

Also you would do us a big favor, if you'd leave a review on the asset store, once you got some experience with the asset. Thank you!

Setup and architecture

This asset is a Unity-Native implementation of the Socket.IO protocol version 4 (it is also compatible to version 3) Setting up the system is as simple as adding the component **SocketIOClient** to a GameObject in your scene. You can then access it from your scripts. The Client holds the connection to the server and it can create connections to multiple namespaces per client instance. The Client connects to the default namespace ("/") right after you call SocketIOClient.Connect(...) and you can access additional namespaces using SocketIOClient.GetNamespace(...). On a SocketIONamespace instance, you will find all commonly required methods to exchange events with the server.

Version compatibility

	Server v1	**Server v2**	**Server v3**	**Server v3.1**	**Server v4**
Socket.IO Client for Unity v2	No	Yes	No	Partial*	No
Socket.IO Client for Unity v3/v4 BASIC	No	No	Yes	Yes	Yes
Socket.IO Client for Unity v4 PLUS	No	No	Yes	Yes	Yes

*) Partial means that a compatibility mode must be enabled on server side which actually makes the server act like a V2 server. This means it has no V3 benefits. It would be the same as if you ran a v2 server. This is only meant for transitioning to V3.

This table does only cover our assets. There are many other solutions available, some don't work on WebGL at all, some use deprecated APIs. Some are over-aged. Our assets are made FOR Unity, not only

WITH Unity. This means every feature was designed having Unity in mind regarding thread safety, saving framerate, WebGL support etc.

Differences between Basic and Plus Client

Our two current Socket.IO assets are completely independed code bases. While both provide a way to connect to Socket.IO Servers (Version 3 or higher), their implementation is completely different. For simple applications, the BASIC variant might be enough, while the PLUS variant provides the maximum flexibility and a way larger feature set and an implementation which is near 100% consistent between WebGL and Native as we entirely re-implemented the whole protocol on top of plain Websockets.

The main difference is, that the Basic variant only supports a single payload, no binary messages and always returns a JSON string for events. The PLUS variant works with objects, supporting even binary payloads and more than one payload per event. It does not deliver a JSON representation but always parsed objects.

Code difference

A simple fictive case would be to receive an event, containing a position of an object and then updating the location locally.
The following example shows the simplest possible implementation with the respective asset, without using special or higher features.

For the example, we assume we got a handle to the object in "remoteObj", all server settings are configured in the components using the inspector, and...

...using the BASIC asset...

...we got a reference to the SocketIOCommunicator (from the basic asset) in "sioCom":

```
void Start() {
        sioCom.Instance.On("ObjectMoved", (payloadJson) => {
                Vector3 newPos = JsonUtility.FromJson<Vector3>(payloadJson);
                remoteObj.transform.position = newPos;
        });

        sioCom.Instance.Connect();
}
```

...using the PLUS asset...

...we got a handle to the SocketIOClient (from the PLUS asset) in "io":

```
void Start() {
        io.D.On<Vector3>("ObjectMoved", (newPos) => {
                remoteObj.transform.position = newPos;
        });

        io.Connect();
}
```

Feature Matrix

	Socket.IO Client V2	**Socket.IO Client V3/V4 BASIC**	**Socket.IO Client V4 PLUS**
TRANSPORT FEATURES			
SSL/TLS Support	Yes	Yes	Yes
Websocket Transport	Yes	Yes	Yes
Long-Polling	No	No	No
Set custom path	Yes	Yes	Yes
Custom Parser/Adapter	No	No	Experimental (Must be implemented by yourself. Not recommended)
Multiple Instances (Connect to multiple servers)	Yes	Yes	Yes
SUPPORTED PLATFORMS			
Windows, Linux, MacOS	Yes	Yes	Yes
Android, iOS	Yes	Yes	Yes
WebGL	Yes	Yes	Yes
Console platforms	Assumed yes[1]	Assumed yes[1]	Assumed yes[1]
SOCKET.IO FEATURES			

	Socket.IO Client V2	**Socket.IO Client V3/V4 BASIC**	**Socket.IO Client V4 PLUS**
Connect Default Namespace ("/")	Yes	Yes	Yes
Connect Additional Namespaces (Multiplexing)	No	No	Yes
Event-Specific Callbacks (On/Off)	Yes	Yes	Yes
CatchAll-Callbacks (OnAny/OffAny)	No	Yes[2]	Yes
Authentication payloads	No	Yes, static	Yes, per Namespace (using a delegate)
Binary Payloads	No	No	Yes
Object Payloads	Manually serialized	Manually serialized	Yes[3]
Multiple Payloads	No	No	Yes
Acknowledgements	No	No	Yes
Volatiles & Reliability	Similar to volatile	Similar to volatile	Volatile or with enhanced reliability[4]
IMPLEMENTATION FEATURES			
Delay-Free, threaded delegate	No	No	Yes[5]
Using UnityEvent for common Callbacks	No (delegate)	No (delegate)	Yes

[1]) We don't see any reason why it should not work but we are not able to verify this. Consoles might add legal or technical restrictions. Referr to your agreements and technical descriptions.
[2]) Implementation can slightly differ from the Socket.IO standard as well as between WebGL and Native

builds.
[3]) Only objects, that can be serialized using Json.Net
[4]) In plain JS Socket.IO, all packets are guaranteed to be delivered. When using "volatile", sending is tried once and the packet is dequeued in any case. Our basic assets work similar to the volatile operation. Our PLUS asset works the same way by default but is can be toggled. If volatileMode is disabled, the system tries to guarantee the delivery of a packet. However this only means it will stay queued while the client is disconnected, and requeued upon most networking errors. This is not a 100% guarantee. [5]) Normal EventHandlers are called using a dispatcher loop. This is required to safely run them on the main thread. This causes a delay of usually one frame between receiption and callback. "Delay-Free callbacks" are delegates that get invoked immediately when an event comes in on the socket, but in native builds they get called from the receiver thread so the delegate is NOT executed on the main thread. This is only for advanced programmers, because you need to mind thread safety. In WebGL builds they are also delayed technically. This can not be changed.

# Correct Seat License Count Officially you need to buy a seat license in the asset store for any unity user working with the asset. As this is not an editor asset but a "game asset" it is hard to determine how many seats you need, right? We are always trying to make our assets affordable and still rock solid. We ask our customers to be fair players and buy licenses according to the actual usage and revenue. Here is a guideline: ##For independent developers up to 5 team members working on your game (in total) and in compliance to the revenue terms of Unity "Free"... ...we allow using this asset with a single seat license for unlimited projects as long as you **comply with the revenue terms for Unity "Free"** (no matter if you are actually uisng Plus for some reason, Pro and Enterprise excluded). We do only count people working on the game (artists, developers, game designer, ...). If you hire IT specialists, marketing etc they do not count. ##For bigger independent dev teams, if you don't comply to the Unity Free license terms as a small team or if you are using Unity Pro or Enterprise... ...we ask you to buy one license per team member who works on your project (in Unity). ##For big studios and any enterprise licensee ...we assume you are making a lot of money with your games. Great! While the license requirements still are met with one license per unity user, we **kindly request** you to buy one seat per unity user or per project using our asset depending on what count is higher.

Frequently Asked Questions

Please click here

FIRESPLASH ENTERTAINMENT

Namespace Firesplash.GameDevAssets.Socket IOPlus

Classes

DataTypes

Contains simple DataTypes used for Socket.IO communication and states

DataTypes.SocketIOErrorPayload

A Payload template to mimic JS "Error" events. Error messages are contained in the "message" property

DataTypes.SocketIOException

This is the base class all Socket.IO-Exceptions derive from

DataTypes.SocketIOProtocolViolationException

This exception is thrown, when the application is trying to violate a protocol constraint. There are only rare possibilities to do so, though. This exception is NOT thrown for incoming packages!

DefaultParser

This implemented the default Socket.IO parser to parse string typed EngineIO messages into Socket.IO and encode packets vice versa.

Parser

This is a skeleton class for writing Socket.IO / Engine.IO transcoders

SocketIOClient

This MonoBehavior derives from EngineIOClient and implements the main Socket.IO Manager logic on top of Engine.IO. It implements Socket.IO protocol version 5 (which is used by Socket.IO v3 and v4)

SocketIOEvent

SocketIONamespace

This class represents a (usually connected) Socket.IO namespace and implements the EventEmitter and the EventReceiver. We try to keep our API as near to the official Socket.IO V4 API as possible within this class. https://socket.io/docs/v4/emitting-events/https://socket.io/docs/v4/listening-to-events/#eventemitter-methods

SocketIOPacket

This class represents a lowlevel SocketIO packet in its parsed state.

Enums

DataTypes.ConnectionState

DataTypes.PacketType

Delegates

DataTypes.SocketIOAuthPayloadCallback

This delegate gets called when a Socket.IO namespace is being connected. Your function must return null if not auht payload is required for the namespace, or an object that can be serialized by Json.Net if a payload must be provided

DataTypes.ThreadedSocketIOEvent

This delegate gets called on a namespace for any Socket.IO event (received or internally generated). The delegate is invoked from a Thread so it is not safe to access Unity functions from it.