Table of Contents

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 an implementation of the Socket.IO protocol version 3 and 4 designed for Unity It works quite straight-forward: Just create a GameObject and add the "Socket IO Communicator" component to it. Enter all details used for connecting into the fields. The Address has to be provided **without protocol prefix** and in most cases without any folders. **Socket.IO-Namespaces and Acknowledgement are not supported by the BASIC library.** See more information in the feature matrix below.

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.

Backwards Compatibility Note / Warning

One might ask: If your Socket.IO asset v2 is compatible to Socket.IO Server v4, why was this still a good investment? Why the higher price? The Socket.IO v2 asset only covers the very basic portions of the protocol in native mode. For example reconnects are not automated, most of the status events are not emitted to your application and after all the compatibility layer is not a future-safe investment. This Asset version implements the most important events (reconnect, connect_error, reconnect_error and so on) as well as auto reconnecting. Further Socket.IO v3/v4 has a way better timeout/ping handling and is considered more stable. This is only the case when both sides are v3/v4 The higher price is applied as v3/v4 support needed huge changes in the code base and lots of research. The undelaying protocol has significantly changed since v2 which is also the reason why v2 and v3/v4 are not compatible (without the compatibility flag which actually degrades v3/v4 servers to v2) After all we provide our customers a discounted upgrade/downgrade path, so at any time customers will only pay the extra work, and do not have to buy a complete second asset.

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 this 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			
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
IMPLEMENTATION FEATURES			
Delay-Free, threaded delegate	No	No	Yes[4]

	Socket.IO Client V2	Socket.IO Client V3/V4 BASIC	Socket.IO Client V4 PLUS
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]) 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

Project / Asset Set Up

Our asset works quite straight-forward. Just create a GameObject and add the "Socket IO Communicator" component to it. Enter all detail used for connecting into the fields. The Address has to be provided without protocol prefix or any folders. Socket.IO-Namespaces are not supported by this library.

The Socket.IO-Address has to be entered in an URI-Format without protocol: *If using standard ports (Insecure: tcp/80, secure: tcp/443) you do not need to enter a port. If using the standard socket.io path (/socket.io/) you do not need to specify a path. WARNING: Paths are not the same as namespaces! A path is what you enter in path-option on server side.*

Whenever possible you should not use the "Auto Connect" feature as most likely you will want to **setup the listeners** ("On") **before connecting**. See the example for a best practice setup.

Connecting at runtime / from code

Instead of configuring the connection parameters in the component using the inspector, you can also specify the target address at runtime using the Connect() method: void Instance. **Connect** ([string targetAddress, bool enableReconnect,][SIOAuthPayload authPayload])

This is described later in this documentation.

Multiple Socket.IO Connections

If you want to connect to more than one Socket.IO server, you have to use one GameObject per connection and you need to name them different. If you use two GameObjects identically named with SocketIOCommunicator Component in WebGL builds, it will result in unexpected behavior because the Socket.IO system depends on unique names from javascript (JSLib) side.

Json.NET for better Json parsing

Why Json.NET

Unfortunately Unity's own JSON "Skills" (JsonUtility) are very limited and everything except reliable when it comes to complex objects – but that's fine as that is not the scope of those methods. There is a very great alternative out there: Newtonsoft Json / JSON.NET – The latter is a fork which has been customized to support Unity's IL2CPP builds (e.g. WebGL).

How do I integrate Json.NET? To utilize that and keep the ease of use of "Emit(eventName, data)" without the third parameter, install the JSON-NET package as described here: https://github.com/jilleJr/Newtonsoft.Json-for-Unity#installation

Afterwards head to your project's player settings and set the "Scripting Define Symbol" HAS_JSON_NET:

Do not set the flag, if you did not install JSON.NET into the project. This would cause compiler errors.

Recent Unity versions bring their own copy of Json.Net. Their package is identical to the ,,official" Json.Net implementations so you can use whichever your prefer´but installing Json.Net while having unity's variant in the project will cause compiler errors regarding duplicate assemblies.

Unity's variant is currently delivered with the perforce / collab plugin which is enabled by default.

https://docs.unity3d.com/Packages/com.unity.nuget.newtonsoft-json@2.0/manual/index.html

Using the asset / Code Referene

We provide a well-documented example in our asset. Please review the code to get a closer understanding.

The SocketIOCommunicator Component provides access to a singleton "Instance" which contains the actual Socket.IO implementation specific to the platform. You use this to interact with the library. Further in runtime it will create an object called "SIODispatcher" which is a helper for running everything thread safe. This component is used to enqueue actions to be run on the main thread.

The "Instance" provides the API required to interact with Socket.IO. Click here to view it's reference.

Frequently Asked Questions

General issues

Does this asset support visual scripting?

No. This asset is made for developers working with code.

Why are we not creating Units for it?
Creating universal nodes for Socket.IO would cost lots of time and only target very few people. You can however create your own Visual Scripting Units for implementing Socket.IO communications. Implementing specialized units is much easier than creating universal nodes because our asset for example supports variable payload counts and types.

The game is not connecting to my server…

…not even locally (but the example works with the Firesplash Entertainment hosted demo server)

In most cases this is caused by one of the following reasons:

1. **You enabled the "Secure Connection" option in the component but did not correctly configure SSL on your server side – or vice versa!** Try to access the Socket.IO server using a Webbrowser: http(s)://your.server-address.here/socket.io/ Don't forget the trailing "/" It **should** say {"code":0,"message":"Transport unknown"} If any browser errors appear, this is your issue. Be sure to use the exact same address as in the unity component at the red position, and set **http** for *unchecked* secure connect and **https** for *checked*.
2. **You are using the wrong port** Socket.IO assumes port 80 (insecure) or 443 (secure). Make sure to enter the correct port if you are using some other. Test access like in solution1 to make sure everything is right

Now I was able to run a server locally and connect the game with it but… after uploading to the server it is no longer working again…

Well…

1. **Again: SSL might not be configured correctly (you won't communicate unencrypted in production, right?)** Check SSL as done in the last question, possible solution 1. But this time using the actual server address.
2. **Your server does not accept the transport "websocket" or some layer in between drops the requests** This is a whip of rocket science for most developers: Socket.IO for Unity depends on the "websocket" transport and is NOT able to fallback to long-polling. This is why a standard socket.io application might work while the unity asset is not working. Load Balancers, Reverse Proxies, Firewalls and any device/solution between the client and the server could potentially cause issues here. Usually the issue sits on the server side. Most likely – if using one – your load balancer or reverse proxy (like nginx/apache proxying requests through your servers port 80/443 down to a nodejs based server on a different port) is not configured correctly. Here are some hints for reverse proxy configuration: https://socket.io/docs/v3/reverse-proxy/

This Rocket-Science can easily be tested by writing a web based test application. Have a look at our example ServerCode zip archive. It contains such a test html file. The important thing about it is to only try the websocket transport using the options: https://socket.io/docs/v3/client-initialization/#transports

I am using a transport adapter on my server side…

Can I connect to this server using your asset?

Usually no. Transport adapters need to be implemented on both sides. You would have to rework our code to implement the transport adapter.

How do I Serialize/Deserialize complex objects?

Unity's builtin "JsonUtility" does not support complex objects. This means it only supports objects and only a single layer of complexity.

Job	Builtin JsonUtility variant / workaround	Better variant (requires Json.Net)
Employee data	{ "firstname": "John", "surname": "Baker", "job": "Dentist" }	{ name: { "first": "John", "last": "Baker" }, "job": "Dentist" }
A list of numbers	{ "a": 12, "b": 15, "c": 20}	[12, 15, 20]
A dynamic list of employees	*Not possible at all*	[{ name: { "first": "John", "last": "Baker" }, "job": "Dentist" },{ name: { "first": "Martina", "last": "Carrey" }, "job": "Student" }, …]

The good news: All those "bad" situations can be solved using Json.Net (see Instructions)

Callbacks (On) firing twice after a reconnect

When a reconnect happens, my listeners are invoked twice. After another reconnect three times, and so on. Why?

This is likely an issue in how - or better when - you are calling "On" to register your callbacks.
Registered Callbacks do survive a reconnect, as this is a client side registration. You **must not** register your callbacks in a "connect" event, except you add some kind of check that it is not being registered twice - or of course if this behaviour is wanted.

See also the official Socket.IO documentation.

How do I send request headers, for example to provide an oauth token?

Websockets - the underlaying technology of this socket.io implementation - do not support request headers (see also this stackoverflow discussion) so this is not directly possible.
Instead, socket.io implements it's own mechanism to send authentication data on connect time, using the method SocketIOClient.SetAuthPayloadCallback - There is an example in our provided example script. The Server-side API does **not** access a header, but accesses this data using a call like `socket.handshake.auth.whatever` where whatever is the fieldname in your authentication payload.

This is the **only** officially supported way to provide data before the actual socket.io connection is established. Another (hacky) workaround could be to provide the token within the connect URI using query parameters.

See also the official Socket.IO documentation

WebGL / IL2CPP Build Issues

I am using WebGL (on an SSL-secured site)…

…and the game does not connect to the server.

The most common issue is an SSL-misconfiguration. Please make sure that you can access **https:// <*your-socketio-address*> /socket.io/** from your browser without any SSL errors or warnings. (A json formatted transport error is fine) Also remember that in modern browsers, a WebGL game delivered via HTTPS may not contact a server using HTTP/WS but only using HTTPS/WSS so this means SSL everywhere or nowhere.

My IL2CPP-Build (WebGL is always IL2CPP) using Json.Net is showing errors…

…like Attempting to call method 'System.Collections.Generic.List[…]::.ctor' for which no AOT code was generated

For some reason using arrays in structures does not work on IL2CPP. Try to use Lists instead. struct Foo {public int[] bar;}could break, struct Foo {public List<int> bar;} works.

My WebGL (or other IL2CPP) build using Json.Net is crashing…

…telling me that a constructor or type has not been found

Most likely the linker stripped code that is used by Json.Net. You should add a link.xml file to your project which restricts the linker from doing so. Its content depends on your project but here is an example:
https://github.com/SaladLab/Json.Net.Unity3D/blob/master/src/UnityPackage/Assets/link.xml

I imported Json.Net and now I am getting a lot of "duplicate assembly" errors

Sometime around mid of 2021 Unity started to deliver their "own" Json.Net package with some components. If your project already contains that package, installing Json.Net will cause this behavior. It is up to you to decide what to do. Unity officially states that their package is not meant to be used but on the other hand you got no real choice. Seen from a dev's perspective it should not cause any issues so if your project includes unity's version, remove your imported one and use unity's. https://docs.unity3d.com/Packages/com.unity.nuget.newtonsoft-json@2.0/manual/index.html

Mobile Platforms

The asset should work on mobile platforms without any issues if you configure your project correctly.

Android build does not connect to the server

Please make sure, you set this to required in project settings: