Table of Contents

Overview Overview

How do I... How do I...

Troubleshooting / FAQ Troubleshooting / FAQ

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			
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]
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

How do I...

... Connect to a server?

This is an example how your script could look to connect to the default namespace and reconize when the connect succeeded:

```
SocketIOClient sio; //Assigned via inspector

void Start()
{
    sio.D.On("connect", (p) =>
    {
        Debug.Log("Connected!");
    });

    sio.Connect();
}
```

Connect to a specific server

You can set the target server address in the inspector or pass it while connecting:

```
[...]
    sio.Connect("https://backend.example.com");
[...]
```

Change the Path to the server

If your server is running on a non-standard path, you can pass it using the server address. Please do not confuse this with namespaces! This is the client implementation for this specific feature.

```
[...]
    sio.Connect("https://backend.example.com/mysocket.io/");
[...]
```

Use a different port

If your server is running on a non-standard port, you can pass it using the server address.

```
[...]
    sio.Connect("https://backend.example.com:1234");
[...]
```

Use HTTP without encryption

Please don't do that. Its not 19XX anymore. However this is totally possible:

```
[...]
    sio.Connect("http://backend.example.com");
[...]
```

All server address options

Above are some examples, but there is more: The serverAddress allows to specify the protocol, hostname, port, path and query parameters in one string.

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 into the path-option on the server side.

Possible valid values / combinations could be:

- https://some.server.org
- http://some.server.org
- https://some.server.org:1234
- http://some.server.org?someLoadbalancerHint=abc
- https://some.server.org:1234/otherpath/
- https://some.server.org:1234/otherpath/?someLoadbalancerHint=abc

... Access a namespace

Socket.IO allows to connect to multiple namespaces using one connection. There is always the default namespace ("/"), and you can connect additional namespaces.
Our PLUS-Asset also implements this, which is also the reason that you need to access "D" (for DefaultNamespace) instead of directly subscribing / emitting events on the client instance.

Default Namespace

As simple as `sio.DefaultNamespace` or even shorter `sio.D`

Other namespaces

This is not much more of a hassle: `sio.GetNamespace("/otherNamespace")`

Other ways to work with namespaces

```
SocketIOClient sio; //Assigned via inspector

void Start()
{
    SocketIONamespace nsAdmin = sio.GetNamespace("/admin");

    nsAdmin.On("connect", (sioEvent) =>
    {
        Debug.Log("Connected to admin namespace!");

        //you can also send to other - e.g. the default - namespaces form here
        sio.D.Emit("ServerWillShutdown");
        //or
        sio.D.GetNamespace("/whatever").Emit("ServerWillShutdown");

        //now emit to the admin namespace
        nsAdmin.Emit("ShutdownServer");

        //The coolest thing is, you can create universal callbacks because you will always have a reference back to the namespace from the event itself
        sioEvent.Namespace.Emit("AnotherEvent");
    });

    sio.Connect();
}
```

... Subscribe to events

For these examples, we assume you already connected and the Client is referenced in "sio".

Subscribe to events

The code example is assumed to run in Start()

sio.D is a shorthand to sio.GetNamespace("/")

```
//Single payload events can be subscribed using generics
sio.D.On<string>("SinglePayloadEventName", (payload) => {

});


//If you got multiple payloads, this is also possible but requires a slightly different API. Now assuming, payload #1 is a string and #2 is an integer:
//This is what you receive, if the server does this: socket.Emit("MultiPayloadEventName", "some string", 135);
sio.D.On("MultiPayloadEventName", (sioEvent) => {
    Debug.Log("Payload 1: " + sioEvent.GetPayload<string>(0));
    Debug.Log("Payload 2*2= " + sioEvent.GetPayload<int>(1) * 2);
});


//Maybe the event is an acknowledgement? This is now also possible:
sio.D.On("SomeACKEventName", (sioEvent) => {
    if (sioEvent.callback != null) {
        object[] response = new string[] { "Some example text" };
        e.callback.Invoke(response); //this sends the acknowledgement back to the server
    }
});
```

Unsubscribe from an event

If you register a callback without using a lambda expression and not through a generic "On" call, you can unsubscribe again:

```
//Create a clalback function
void ProcessSomeEvent(object[] payload) => {
    if (payload != null) {
        foreach(var p in payload) Debug.Log(p.toString());
    }
}

//Register the callback
sio.D.On("SomeEvent", ProcessSomeEvent);

//unregister this specific callback again
sio.D.Off("SomeEvent", ProcessSomeEvent);
```

You can also unregister all callbacks form an event - This also removes the generic registrations:

```
//Let's register using a generic "On"
sio.D.On<string>("SinglePayloadEventName", (payload) => {
    Debug.Log(payload);
});

//And then clear all registrations on the event
sio.D.RemoveAllListeners("SinglePayloadEventName");
```

Subscribe for single execution

If you only need to wait for a specific event and fire the callback once, you can make this even easier:

```
sio.D.Once<string>("SinglePayloadEventName", (payload) => {
    //This callback will only fire once
    Debug.Log(payload);
});
```

... Emit events

Simple events (Single Payload)

The following code can be run wherever you need to emit events.

A payload must always be serializable by Json.Net or a binary value in form of a `byte[]`

If you hand Emit an array as payload, the elements will be sent as individual payloads! See next section.

```
//Some simple examples
sio.D.Emit("EventWithoutPayload");
sio.D.Emit("EventWithSimplePayload", "A string payload");
sio.D.Emit("EventWithSimpleIntPayload", 1234);
```

Complex events (multiple payloads)

Multiple payloads are also possible:

```
List<string> list = new List<string>();
list.Add("a string as first payload"); //string1
list.Add("another string as second payload"); //string2
sio.D.Emit("EventWithMultiplePayloads", list.toArray());
```

The server side using Javascript will access the payloads like

```
io.on("EventWithMultiplePayloads", (string1, string2) => {
    [... ]
});
```

... work with complex data structures

This is where the PLUS asset gets interesting: You can use any object, which can be serialized by Json.Net and pass it to a generic "Emit" or enclosed in a payload array. You can also receive any of these objects using a generic "On".

Just make sure, that your data type can be serialized.

For example classes with recursions, like Transforms, can not be serialized. Create your own data classes or structs if required.

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.

There are assemblies missing

If not already done, please import Json.Net: https://github.com/jilleJr/Newtonsoft.Json-for-Unity/wiki/Install-official-via-UPM

I replaced your Basic asset by the Plus and now I get compiler errors

The PLUS asset has a different architecture and API. You will have to refactor your code.

I replaced your Basic asset by the Plus and now things stopped working

The PLUS asset has a different architecture and API. You will have to refactor your code. The callbacks do no longer receive json strings but objects from now on.
Depending on your code, you might not get compile errors because the library might now be treating your callbacks as generic subscriptions accepting a string.

Game does not connect, not even locally

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

1. You are using a secure URL (https/wss) 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

3. Your server (software) does not support the websocket transport
This asset does not support Long-Polling by design. This would not make sense for modern apps and games. Make sure that your server supports websockets.

After uploading, the server it is no longer working but it worked locally

1. Could be an SSL issue
Again: SSL might not be configured correctly (you won't communicate unencrypted in production, right?) Check SSL as done in the first question, possible solution 1. But this time using the actual server address.

2. The Websocket transport not supported by the server or a proxy in between
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

Transport Adapters

I am using a transport adapter on my server side. Can I connect to this server using your asset?

Easy answer: **No.**

Complete answer: Under some circumstances **yes** but you must implement the parser and encoder on your own. We can't assist you here. It will also only work if the basic transport logic stays in place (Engine.IO via websockets). It is a Case-By-Case thing. You can derive a class from Parser and inject it using SocketIOClient.SetParser

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: https://socket.io/docs/v4/client-api/#event-connect

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: https://socket.io/docs/v4/middlewares/#sending-credentials

WebGL / IL2CPP builds

Connecting fails from SSL-Secured website

Question:I am using WebGL (on an SSL-secured site) and the game does not connect to the server. Why?
Answer: The most common issue is an SSL-misconfiguration. Please make sure that you can access https:///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.**

IL2CPP build throwing missing AOT code errors

Problem: My IL2CPP-Build (WebGL is always IL2CPP) is showing errors like "Attempting to call method 'System.Collections.Generic.List[…]::.ctor' for which no AOT code was generated"
Solution: 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 bar;} works.

Build is crashing at runtime

Problem: My WebGL (or other IL2CPP) build using Json.Net is crashing telling me that a constructor or type has not been found.
Solution: 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

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: