
# Socket.IO V3 / V4 Client Asset Documentation


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 have any issues, please write an email to assets@firesplash.de and we will be happy to help!**

This asset is used to connect a unity project to a server (or multiple servers) using Socket.IO middleware. The library provides the most important methods for this. It was not and will never be a try to implement every single aspect but the library is extendable as we provide the full source code.

What we sell and deliver is everything you need for most applications.

After all we would love to get your honest review on this asset on the asset store when you got some experience with it. This helps us out a LOT!

## Socket.IO Protocol version compatibility

For general version compatibility please referrer to this link: https://socket.io/docs/v4/client-installation/index.html
**Compatibility matrix from our side:**

| Server Version -> v Asset Version v | v1.x | v2.x | v3.0.x | v3.1.x | v4.x |
|---|---|---|---|---|---|
| Asset V2 | | | | | |
| Asset V3/V4 | | | | | |

Green = Full compatibility (within the general limitations of this asset according to this documentation)

Yellow = Compatible via 'allowEIO3' flag and **NOT officially supported** – Might break at any time.

Red = incompatible

### 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 (Note: a **connect_error** causes **NO reconnect** while any connection drop during a running

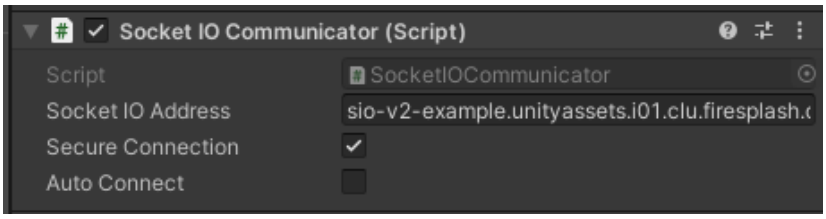
session does) Further Socket.IO v3/v4 has a way better timeout/ping handling and is considered more stable. This is only the case when both sides are v3/v4

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.

## 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.
S.IO-Namespaces are not supported by this library.



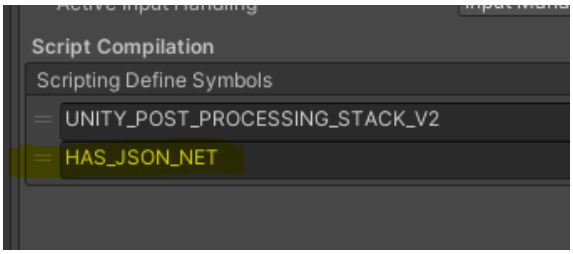

Whenever possible you should not use the “Auto Connect” feature as most likely you will want to setup the listeners (“On”) before connecting. See the example for a best practice setup.

**If you want to connect to more than one Socket.IO server, you have to use on 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.**
*This is a Unity limitation.*

## Why JSON.NET and how do I integrate it?

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).
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.**

## Usage

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 following methods are available:

```
void Instance.On(string eventName, Callback(string payloadData));
```

Used to subscribe to a specific event. The callback will be executed everytime when the specific event is received. The callback contains a string. This is the data sent from the server, eighter a stringified JSON object (if the data was a json object) or a plain text string.
If the server sent no payload, the string will be null.

*Example:*

```
sioCom.Instance.On("WhosThere", (string payload) =>
  {
    Debug.Log("Data received: " + payload);
  }
);
```

```
void Instance.Off(string eventName, Callback);
```

Used to remove the subscription to an event.

*Example:*

```
sioCom.Instance.Off("WhosThere", WhosthereHandlerMethod);
```

`void Instance.Emit(string eventName [[, string payloadData], bool DataIsPlainText]);`

Used to send an event to the server containaing am optional payload.
**With** third parameter: If `DataIsPlainText` is set true, the data will be delivered as a string. Else it will be delivered as a JSON object. If JSON object is sent (DataIsPlainText=false) and the string is not a valid stringified object, unexpected errors might occur. The third parameter is a hard override.
**Without** third parameter: If the payload is a valid JSON stringified object, the server will receive it as a JSON object. The automatic detection (JSON or PlainText) only works reliably in conjunction with JSON.NET as described above. If you don't use JSON.NET (or if you forgot to set the flag), omitting the third parameter will cause a deprecation warning.

*Examples:*

```
sioCom.Instance.Emit("ItsMe", "Hello World", true);
sioCom.Instance.Emit("ItsMe", "{“msg”: “Hello World”}", false);
sioCom.Instance.Emit("ItsMe", "Hello World");
```

`void Instance.Connect()`

When Auto-Connect is disabled (best practice), this call connects to the server.

*Example:*

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

`void Instance.Close()`

Closes the connection to the server

*Example:*

```
sioCom.Instance.Close();
```

`bool Instance.IsConnected()`

Returns a Boolean which is true if the library is currently connected to the server.

*Example:*

```
while (sioCom.Instance.IsConnected()) {
  sioCom.Instance.Emit("KnockKnock");
  //Please don’t do this. That’s evil 😉
}
```