Socket.IO V2 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 support AT firesplash DOT 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.

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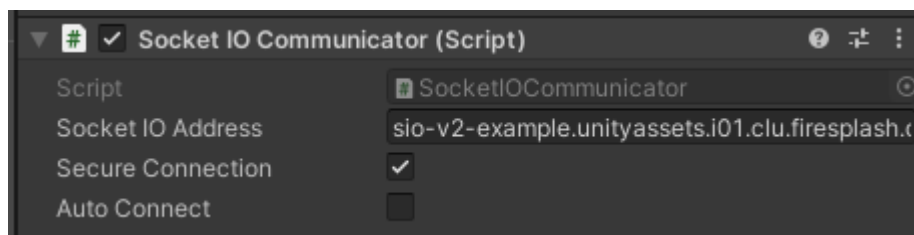
Whenever possible you should not use the "Auto Connect" feature as most likely you will want to setup the listeners ("On") before connecting. See the example for a best practice setup.

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.

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[, Callback(string payloadData)]);
```

Used to send an event to the server containaing am optional payload.
If the payload is a valid JSON stringified object, the server will receive it as a JSON object.

Example:

```
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 😉
}
```