websocket-sharp for Unity

Provides the **WebSocket** protocol client and server.

Before Starting

If you succeed in downloading and importing **websocket-sharp for Unity**, Could you see whether **websocket-sharp** menu exists on the menu bar in Unity Editor?

And if exists, Could you try to do both **websocket-sharp/Echo Back Test** and **websocket-sharp/About websocket-sharp** menu items?

If you obtain same results as screenshots of **websocket-sharp for Unity** on Unity Asset Store, **websocket-sharp for Unity** is available.

But if does not exist, Could you see whether you succeeded in downloading and importing **websocket-sharp for Unity**?

Getting Started

WebSocket Client

```
using System;
using UnityEditor;
using UnityEngine;
using WebSocketSharp;

namespace WebSocketSharp.Unity.Editor
{
  public class MenuExtension : MonoBehaviour
  {
    [MenuItem ("websocket-sharp/Echo Back Test")]
    private static void EchoBack ()
    {
      string res = null;
      using (var ws = new WebSocket ("ws://localhost:4649/Echo"))
      {
        var ver = Application.unityVersion;
        ws.OnOpen += (sender, e) =>
        {
          ws.Send (String.Format ("Hello, Unity {0}!", ver));
        };

        ws.OnMessage += (sender, e) =>
        {
          res = e.Data;
        };

        ws.OnError += (sender, e) =>
        {
          Debug.LogError (e.Message);
        };

        ws.Connect ();
      }

      if (!res.IsNullOrEmpty ())
        EditorUtility.DisplayDialog ("Echo Back Successfully!", res, "OK");
    }
  }
}
```

Step 1 Required namespace.

```
using WebSocketSharp;
```

The `WebSocket` class exists in the `WebSocketSharp` namespace.

Step 2 Creating a instance of the `WebSocket` class with the specified WebSocket URL to connect.

```
using (var ws = new WebSocket ("ws://example.com"))
{
  ...
}
```

The `WebSocket` class inherits the `IDisposable` interface, so you can use the `using` statement.

Step 3 Setting the `WebSocket` events.

WebSocket.OnOpen Event A `WebSocket.OnOpen` event occurs when the WebSocket connection has been established.

```
ws.OnOpen += (sender, e) =>
{
  ...
};
```

`e` has come across as `EventArgs.Empty`, so you don't use `e`.

WebSocket.OnMessage Event A `WebSocket.OnMessage` event occurs when the `WebSocket` receives a WebSocket data frame.

```
ws.OnMessage += (sender, e) =>
{
  ...
};
```

`e.Type` (`WebSocketSharp.MessageEventArgs.Type`, its type is `WebSocketSharp.Opcode`) indicates the **Frame Type** of a received WebSocket frame. So by checking it, you determine which item you should use.

If `e.Type` equals `Opcode.TEXT`, you use `e.Data` (`WebSocketSharp.MessageEventArgs.Data`, its type is `string`) that contains a received **Text** data.

If `e.Type` equals `Opcode.BINARY`, you use `e.RawData` (`WebSocketSharp.MessageEventArgs.RawData`, its type is `byte[]`) that contains a received **Binary** data.

```
if (e.Type == Opcode.TEXT)
{
  // Do something with e.Data
  return;
}

if (e.Type == Opcode.BINARY)
{
  // Do something with e.RawData
  return;
}
```

WebSocket.OnError Event A `WebSocket.OnError` event occurs when the `WebSocket` gets an error.

```
ws.OnError += (sender, e) =>
{
  ...
};
```

`e.Message` (`WebSocketSharp.ErrorEventArgs.Message`, its type is `string`) contains an error message, so you use it.

WebSocket.OnClose Event A `WebSocket.OnClose` event occurs when the WebSocket connection has been closed.

```
ws.OnClose += (sender, e) =>
{
  ...
};
```

`e.Code` (`WebSocketSharp.CloseEventArgs.Code`, its type is `ushort`) contains a status code indicating the reason for closure and `e.Reason` (`WebSocketSharp.CloseEventArgs.Reason`, its type is `string`) contains the reason for closure, so you use them.

Step 4 Connecting to the WebSocket server.

```
ws.Connect ();
```

Step 5 Sending a data.

```
ws.Send (data);
```

The `Send` method is overloaded.

The types of `data` are `string`, `byte []` and `FileInfo` class.

Step 6 Closing the WebSocket connection.

```
ws.Close (code, reason);
```

If you want to close the WebSocket connection explicitly, you use the `Close` method.

The `Close` method is overloaded.

The types of `code` are `ushort` and `WebSocketSharp.CloseStatusCode`, the type of `reason` is `string`.

In addition, the `Close ()` and `Close (code)` methods exist.

WebSocket Server

```
using System;
using UnityEditor;
using UnityEngine;
using WebSocketSharp.Server;

namespace WebSocketSharp.Unity.Editor
{
  public class ServerMonitor : EditorWindow
  {
    WebSocketServer _server;

    ServerMonitor ()
    {
      _server = new WebSocketServer (4649);
      _server.AddWebSocketService<Echo> ("/Echo");
      _server.AddWebSocketService<Chat> ("/Chat");
      _server.OnError += (sender, e) =>
      {
        Debug.LogError (e.Message);
      };

      _server.Start ();
    }

    void OnDestroy ()
    {
      if (_server != null)
        _server.Stop ();
    }

    void OnGUI ()
    {
      GUILayout.Label ("WebSocket Server started!", EditorStyles.boldLabel);
      if (GUILayout.Button ("Close", GUILayout.Width (100))) {
        Close ();
      }
    }
  }
}
```

Step 1 Required namespace.

```
using WebSocketSharp.Server;
```

The WebSocketServer, WebSocketServiceHost<T> and WebSocketService classes exist in the WebSocketSharp.Server namespace.

Step 2 Creating a class that inherits the WebSocketService class.

For example, if you want to provide an echo service,

```
using System;
using WebSocketSharp;
using WebSocketSharp.Server;

public class Echo : WebSocketService
{
  protected override void OnMessage (MessageEventArgs e)
  {
    Send (e.Data);
  }
}
```

Or if you want to provide a chat service,

```
using System;
using WebSocketSharp;
using WebSocketSharp.Server;

public class Chat : WebSocketService
{
  protected override void OnMessage (MessageEventArgs e)
  {
    Broadcast (e.Data);
  }
}
```

If you override the OnMessage method, it is bound to the server side WebSocket.OnMessage event.

In addition, if you override the OnOpen, OnError and OnClose methods, each of them is bound to the WebSocket.OnOpen, WebSocket.OnError and WebSocket.OnClose events.

Step 3 Creating a instance of the WebSocketServiceHost<T> class if you want the single WebSocket service server.

```
var wssv = new WebSocketServiceHost<Echo> ("ws://example.com:4649");
```

Creating a instance of the `WebSocketServer` class if you want the multi WebSocket service server.

```
var wssv = new WebSocketServer (4649);
wssv.AddWebSocketService<Echo> ("/Echo");
wssv.AddWebSocketService<Chat> ("/Chat");
```

You can add any WebSocket service with a specified path to the service to your `WebSocketServer` by using the `WebSocketServer.AddWebSocketService<T>` method.

The type of `T` inherits `WebSocketService` class, so you can use a class that was created in **Step 2**.

If you create a instance of the `WebSocketServer` class without port number, the `WebSocketServer` set the port number to **80** automatically.

Step 4 Setting the events.

WebSocketServiceHost<T>.OnError Event A `WebSocketServiceHost<T>.OnError` event occurs when the `WebSocketServiceHost<T>` gets an error.

```
wssv.OnError += (sender, e) =>
{
  ...
};
```

`e.Message` (`WebSocketSharp.ErrorEventArgs.Message`, its type is `string`) contains an error message, so you use it.

WebSocketServer.OnError Event Same as the `WebSocketServiceHost<T>.OnError` event.

Step 5 Starting the server.

```
wssv.Start ();
```

Step 6 Stopping the server.

```
wssv.Stop ();
```

HTTP Server with the WebSocket

I modified the `System.Net.HttpListener`, `System.Net.HttpListenerContext` and some other classes of Mono to create the HTTP server that can upgrade the connection to the WebSocket connection when receives a WebSocket connection request.

You can add any WebSocket service with a specified path to the service to your `HttpServer` by using the `HttpServer.AddWebSocketService<T>` method.

```
var httpsv = new HttpServer (4649);
httpsv.AddWebSocketService<Echo> ("/");
```

Secure Connection

As a **WebSocket Client**, creating a instance of the `WebSocket` class with the WebSocket URL with **wss** scheme.

```
using (var ws = new WebSocket ("wss://example.com"))
{
  ...
}
```

If you want to set the custom validation for the server certificate, you use the `WebSocket.ServerCertificateValidationCallback` property.

```
ws.ServerCertificateValidationCallback = (sender, certificate, chain, sslPolicyErrors) =>
{
  // Do something to validate the server certificate.
  return true; // The server certificate is valid.
};
```

If you set this property to nothing, the validation does nothing with the server certificate, always returns valid.

As a **WebSocket Server**, creating and setting a instance of the WebSocket server with some settings for the secure connection.

```
var wssv = new WebSocketServer (4649, true);
wssv.Certificate = new X509Certificate2 ("/path/to/cert.pfx", "password for cert.pfx");
```

Logging

The `WebSocket` class includes own logging functions.

The `WebSocket.Log` property provides the logging functions.

If you want to change the current logging level (the default is `LogLevel.ERROR`), you use the `WebSocket.Log.Level` property.

```
ws.Log.Level = LogLevel.DEBUG;
```

The above means that the logging outputs with a less than `LogLevel.DEBUG` are not outputted.

And if you want to output a log, you use some output methods. The following outputs a log with `LogLevel.DEBUG`.

```
ws.Log.Debug ("This is a debug message.");
```

The `WebSocketServiceHost<T>`, `WebSocketServer` and `HttpServer` classes include the same logging functions.

Source and Examples

- **GitHub: sta/websocket-sharp**

Documentation

- **websocket-sharp Library Reference**

Required Environment

C# **3.0**, .NET **3.5** compatible or later.

Supported WebSocket Specifications

websocket-sharp supports **RFC 6455**.

Supported WebSocket Extensions

Per-message Compression

websocket-sharp supports **Per-message Compression** extension. (But, does not support with extension parameters.)

If you want to enable this extension as a WebSocket client, you should do like the following.

```
ws.Compression = CompressionMethod.DEFLATE;
```

And then your client sends the following header in the opening handshake to a WebSocket server.

```
Sec-WebSocket-Extensions: permessage-deflate
```

If the server supports this extension, responds the same header. And when your client receives the header, enables this extension.

WebSocket References

- **The WebSocket Protocol**
- **The WebSocket API**
- **Compression Extensions for WebSocket**