# Namespace WebSockets.Core

## Classes

ClientHandshake

The client side of the WebSocket handshake.

Handshake

The base protocol class providing functionality shared by both clients and servers.

MessageProtocol

The base protocol class providing functionality shared by both clients and servers.

ServerHandshake

The server side of the WebSocket handshake.

WebRequest

A class modelling the required values of a WebSocket HTTP request.

WebResponse

## Enums

HandshakeState

The state of the handshake.

ProtocolState

The state of the protocol.

# Class ClientHandshake

Namespace: WebSockets.Core

Assembly: WebSockets.Core.dll

The client side of the WebSocket handshake.

```
public class ClientHandshake : Handshake
```

### Inheritance

object ← Handshake ← ClientHandshake

### Inherited Members

Handshake.State , Handshake.SelectedSubProtocol , Handshake.ReadData(byte[], ref long, long) , Handshake.WriteData(byte[], long, long) , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

## Examples

```
using System;
using System.Net;
using System.Net.Sockets;

using WebSockets.Core;
using WebSockets.Core.Messages;

namespace ClientExample
{
    class Program
    {
        static void main()
        {
            var endpoint = IPEndPoint.Parse("localhost:8081");
            var tcpClient = new TcpClient();
            tcpClient.Connect(endpoint);

            stream = tcpClient.GetStream();
            handshake = new ClientHandshake("http://client.com", []);
            handshake.WriteRequest("/chat", "www.example.com");

            // Send the request.
```

```
            var buffer = new byte[1024];
            var isDone = false;
            while (!isDone)
            {
                var bytesRead = 0L;
                handshake.ReadData(buffer, ref bytesRead, buffer.LongLength);
                if (bytesRead == 0)
                    isDone = true;
                else
                    stream.Write(buffer, 0, (int)bytesRead);
            }

            // Read the response.
            var offset = 0L;
            isDone = false;
            while (!isDone)
            {
                var bytesRead = stream.Read(buffer);
                handshake.WriteData(buffer, offset, bytesRead);
                if (offset == bytesRead)
                    offset = 0;
                isDone = handshake.ReadResponse() is not null;
            }

            var webResponse = handshake.ReadResponse();
        }
    }
}
```

# Constructors

## ClientHandshake(string, string[])

Construct a client handshake.

```
public ClientHandshake(string origin, string[] subProtocols)
```

### Parameters

`origin` string

The origin is the url of the initiator of the request.

`subProtocols` string[]

A (possibly empty) array of requested sub-protocols.

# Methods

## ReadResponse()

Read a handshake response.

```
public WebResponse? ReadResponse()
```

### Returns

WebResponse

The response from the server, or null if a complete response has yet to be received.

## WriteRequest(string, string)

Write a handshake request.

```
public void WriteRequest(string path, string host)
```

### Parameters

`path` string

The path on the server.

`host` string

The server name.

# Class Handshake

Namespace: WebSockets.Core

Assembly: WebSockets.Core.dll

The base protocol class providing functionality shared by both clients and servers.

```
public abstract class Handshake
```

**Inheritance**

object ← Handshake

**Derived**

ClientHandshake, ServerHandshake

**Inherited Members**

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

## Properties

### SelectedSubProtocol

The sub-protocol negotiated during the handshake.

```
public string? SelectedSubProtocol { get; protected set; }
```

#### Property Value

string

The (possibly null) selected sub-protocol.

### State

The state of the connection.

```
public HandshakeState State { get; protected set; }
```

### Property Value

HandshakeState

The connection state.

## Methods

### ReadData(byte[], ref long, long)

Read handshake data from the provided array into the handshake buffer.

```
public void ReadData(byte[] source, ref long offset, long length)
```

### Parameters

`source` byte[]

The buffer containing the data.

`offset` long

The offset into the buffer.

`length` long

The length of the data.

### WriteData(byte[], long, long)

Write data from the handshake buffer into the provided array.

```
public void WriteData(byte[] destination, long offset, long length)
```

### Parameters

`destination` byte[]

The array to receive the data.

`offset` long

The point in the buffer to start writing the data.

`length` long

The length of the buffer.

# Enum HandshakeState

Namespace: WebSockets.Core

Assembly: WebSockets.Core.dll

The state of the handshake.

```
public enum HandshakeState
```

## Fields

`Failed = 2`

`Pending = 0`

`Succeeded = 1`

# Class MessageProtocol

Namespace: WebSockets.Core

Assembly: WebSockets.Core.dll

The base protocol class providing functionality shared by both clients and servers.

```
public class MessageProtocol
```

**Inheritance**

object ← MessageProtocol

**Inherited Members**

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

## Constructors

### MessageProtocol(bool)

```
public MessageProtocol(bool isClient)
```

#### Parameters

`isClient` bool

## Properties

### State

The state of the protocol.

```
public ProtocolState State { get; protected set; }
```

#### Property Value

ProtocolState

The protocol state.

# Methods

## ReadData(byte[], ref long, long)

Read data from the protocol.

```
public bool ReadData(byte[] destination, ref long offset, long length)
```

### Parameters

`destination` byte[]

The array to which the data should be written.

`offset` long

The offset in the array to start writing the data.

`length` long

The length of the array.

### Returns

bool

## ReadMessage()

Read a message from the protocol.

```
public Message? ReadMessage()
```

### Returns

Message

If there is a complete message the message is returned, otherwise null.

## WriteData(byte[], long, long)

Write data to the protocol.

```
public void WriteData(byte[] source, long offset, long length)
```

### Parameters

`source` byte[]

The data to write.

`offset` long

The offset from which the data should be written.

`length` long

The available length of the data.

## WriteMessage(Message)

Write a message to the protocol.

```
public void WriteMessage(Message message)
```

### Parameters

`message` Message

The message to write.

# Enum ProtocolState

Namespace: WebSockets.Core

Assembly: WebSockets.Core.dll

The state of the protocol.

```
public enum ProtocolState
```

## Fields

`Closed = 2`

`Closing = 1`

`Connected = 0`

`Faulted = 3`

# Class ServerHandshake

Namespace: WebSockets.Core

Assembly: WebSockets.Core.dll

The server side of the WebSocket handshake.

```
public class ServerHandshake : Handshake
```

### Inheritance

object ← Handshake ← ServerHandshake

### Inherited Members

Handshake.State , Handshake.SelectedSubProtocol , Handshake.ReadData(byte[], ref long, long) , Handshake.WriteData(byte[], long, long) , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

## Examples

```
using System;
using System.IO;
using System.Net;
using System.Net.Sockets;

using WebSockets.Core;
using WebSockets.Core.Messages;

namespace ServerExample
{
    class Program
    {
        static void Main()
        {
            var listener = new TcpListener(IPAddress.Any, 8081);
            listener.Start();
            var tcpClient = listener.AcceptTcpClient();

            var stream = client.GetStream();
            var handshake = new ServerHandshake(subProtocols);

            // Read the client request.
```

```
            WebRequest? webRequest = null;
            var buffer = new byte[1024];
            while (webRequest is null)
            {
                var bytesRead = stream.Read(buffer);
                if (bytesRead == 0)
                    throw new EndOfStreamException();
                handshake.WriteData(buffer, 0, bytesRead);

                webRequest = handshake.ReadRequest();
            }

            // Send the response.
            var webResponse = handshake.CreateWebResponse(webRequest);
            handshake.WriteResponse(webResponse);
            bool isDone = false;
            while (!isDone)
            {
                var bytesRead = 0L;
                _handshake.ReadData(buffer, ref bytesRead, buffer.LongLength);
                if (bytesRead == 0)
                    isDone = true;
                else
                {
                    _stream.Write(buffer, 0, (int)bytesRead);
                    Console.WriteLine("Sent client data");
                }
            }
        }
    }
}
```

# Constructors

## ServerHandshake(string[])

Construct a server handshake object.

```
public ServerHandshake(string[] subProtocols)
```

### Parameters

`subProtocols` string[]

The supported sub-protocols.

# Methods

## CreateWebResponse(WebRequest)

Create the WebSocket response using the web request.

If the response is valid an accept/upgrade response is generated. An invalid response will generate a 400 response containing the reason for the rejection.

An application may inspect the request and create it's own bad response, for example if the path is not valid.

```
public WebResponse CreateWebResponse(WebRequest webRequest)
```

### Parameters

`webRequest` WebRequest

The request from the client.

### Returns

WebResponse

The response to be sent to the client.

## ReadRequest()

Read a WebRequest from the protocol.

The request will be available when all of the request bytes have been received.

```
public WebRequest? ReadRequest()
```

### Returns

WebRequest

A WebRequest if the complete message has been received; otherwise null.

## WriteResponse(WebResponse)

Write a web response to the handshake buffer.

```
public void WriteResponse(WebResponse webResponse)
```

### Parameters

`webResponse` WebResponse

The response to send to the client.

# Class WebRequest

Namespace: WebSockets.Core

Assembly: WebSockets.Core.dll

A class modelling the required values of a WebSocket HTTP request.

```
public class WebRequest
```

**Inheritance**

object ← WebRequest

**Inherited Members**

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object)

## Constructors

### WebRequest(string, string, string, IDictionary<string, IList<string>>, byte[]?)

Construct a web request.

```
public WebRequest(string verb, string path, string version, IDictionary<string,
IList<string>> headers, byte[]? body)
```

#### Parameters

`verb` string

The verb (GET, POST, etc).

`path` string

The server path.

`version` string

The HTTP version.

`headers` IDictionary<string, IList<string>>

The headers.

`body` byte[]

An optional body.

# Properties

## Body

```
public byte[]? Body { get; }
```

### Property Value

byte[]

## Headers

The HTTP Headers.

Note: header keys are case insensitive.

```
public IDictionary<string, IList<string>> Headers { get; }
```

### Property Value

IDictionary<string, IList<string>>

The headers.

## Path

The server path.

```
public string Path { get; }
```

### Property Value

string

The path for the server.

## Verb

The HTTP verb.

```
public string Verb { get; }
```

### Property Value

string

The verb.

## Version

The HTTP version.

```
public string Version { get; }
```

### Property Value

string

The version.

# Methods

## CreateUpgradeRequest(string, string, string, string, string[]?)

```
public static WebRequest CreateUpgradeRequest(string path, string host, string
origin, string key, string[]? subProtocols)
```

### Parameters

`path` string

`host` string

`origin` string

`key` string

`subProtocols` string[]

### Returns

WebRequest

## Parse(byte[])

```
public static WebRequest Parse(byte[] data)
```

### Parameters

`data` byte[]

### Returns

WebRequest

## ToString()

Returns a string that represents the current object.

```
public override string ToString()
```

### Returns

string

A string that represents the current object.

# Class WebResponse

Namespace: WebSockets.Core

Assembly: WebSockets.Core.dll

```
public class WebResponse
```

**Inheritance**

object ← WebResponse

**Inherited Members**

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

## Constructors

### WebResponse(string, int, string, IDictionary<string, IList<string>>, byte[]?)

```
public WebResponse(string version, int code, string reason, IDictionary<string,
IList<string>> headers, byte[]? body)
```

#### Parameters

`version` string

`code` int

`reason` string

`headers` IDictionary<string, IList<string>>

`body` byte[]

## Properties

## Body

```
public byte[]? Body { get; }
```

### Property Value

byte[]

## Code

```
public int Code { get; }
```

### Property Value

int

## Headers

```
public IDictionary<string, IList<string>> Headers { get; }
```

### Property Value

IDictionary<string, IList<string>>

## Reason

```
public string Reason { get; }
```

### Property Value

string

## Version

```
public string Version { get; }
```

### Property Value

string

## Methods

### CreateAcceptResponse(string, string?)

```
public static WebResponse CreateAcceptResponse(string responseKey,
string? subProtocol)
```

#### Parameters

`responseKey` string

`subProtocol` string

#### Returns

WebResponse

### CreateErrorResponse(string, DateTime)

```
public static WebResponse CreateErrorResponse(string reason, DateTime date)
```

#### Parameters

`reason` string

`date` DateTime

#### Returns

WebResponse

## Parse(byte[])

```
public static WebResponse Parse(byte[] data)
```

### Parameters

`data` byte[]

### Returns

WebResponse

## ToBytes()

```
public byte[] ToBytes()
```

### Returns

byte[]

# Namespace WebSockets.Core.Messages

## Classes

BinaryMessage
: A WebSocket message with a payload of binary data.

CloseMessage
: A message indicating the connection should be closed.

  If this is an initiating message the other side will respond with a close with the same code and reason.

DataMessage
: The base class for all messages which hold raw data.

Message
: The base class for all WebSocket messages.

PingMessage
: A ping message.

PongMessage
: The message used to respond to a ping message.

TextMessage
: A message with text data.

## Enums

MessageType
: The types of messages.

# Class BinaryMessage

Namespace: WebSockets.Core.Messages

Assembly: WebSockets.Core.dll

A WebSocket message with a payload of binary data.

```
public class BinaryMessage : DataMessage, IEquatable<Message>,
IEquatable<DataMessage>
```

### Inheritance

object ← Message ← DataMessage ← BinaryMessage

### Implements

IEquatable<Message>, IEquatable<DataMessage>

### Inherited Members

DataMessage.Data , DataMessage.Equals(DataMessage) , Message.Type , Message.Equals(Message) , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

## Constructors

### BinaryMessage(byte[])

Construct a binary message.

```
public BinaryMessage(byte[] data)
```

#### Parameters

`data` byte[]

The message data.

# Class CloseMessage

Namespace: WebSockets.Core.Messages

Assembly: WebSockets.Core.dll

A message indicating the connection should be closed.

If this is an initiating message the other side will respond with a close with the same code and reason.

```
public class CloseMessage : Message, IEquatable<Message>, IEquatable<CloseMessage>
```

**Inheritance**

object ← Message ← CloseMessage

**Implements**

IEquatable<Message>, IEquatable<CloseMessage>

**Inherited Members**

Message.Type , Message.Equals(Message) , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

## Constructors

### CloseMessage(ushort?, string?)

Construct a close message.

If a reason is given a code must be specified.

```
public CloseMessage(ushort? code, string? reason)
```

#### Parameters

`code` ushort?

An optional code.

`reason` string

An optional reason.

# Properties

## Code

A code indicating the reason for the close.

```
public ushort? Code { get; }
```

### Property Value

ushort?

The code.

## Reason

A reason for the close.

```
public string? Reason { get; }
```

### Property Value

string

The reason.

# Methods

## Equals(CloseMessage?)

Indicates whether the current object is equal to another object of the same type.

```
public bool Equals(CloseMessage? other)
```

## Parameters

`other` CloseMessage

An object to compare with this object.

## Returns

bool

true if the current object is equal to the `other` parameter; otherwise, false.

# Class DataMessage

Namespace: WebSockets.Core.Messages

Assembly: WebSockets.Core.dll

The base class for all messages which hold raw data.

```
public abstract class DataMessage : Message, IEquatable<Message>,
IEquatable<DataMessage>
```

**Inheritance**

object ← Message ← DataMessage

**Implements**

IEquatable<Message>, IEquatable<DataMessage>

**Derived**

BinaryMessage, PingMessage, PongMessage

**Inherited Members**

Message.Type , Message.Equals(Message) , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

## Constructors

### DataMessage(MessageType, byte[])

Construct a data message.

```
public DataMessage(MessageType type, byte[] data)
```

#### Parameters

`type` MessageType

The type of the message.

`data` byte[]

The message data.

# Properties

## Data

The data associated with the message.

```
public byte[] Data { get; }
```

### Property Value

byte[]

The message data.

# Methods

## Equals(DataMessage?)

Indicates whether the current object is equal to another object of the same type.

```
public bool Equals(DataMessage? other)
```

### Parameters

`other` DataMessage

An object to compare with this object.

### Returns

bool

true if the current object is equal to the `other` parameter; otherwise, false.

# Class Message

Namespace: WebSockets.Core.Messages

Assembly: WebSockets.Core.dll

The base class for all WebSocket messages.

```
public abstract class Message : IEquatable<Message>
```

**Inheritance**

object ← Message

**Implements**

IEquatable<Message>

**Derived**

CloseMessage, DataMessage, TextMessage

**Inherited Members**

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

# Constructors

## Message(MessageType)

Construct a message.

```
protected Message(MessageType type)
```

### Parameters

`type` MessageType

The message type.

# Properties

## Type

The message type.

```
public MessageType Type { get; }
```

### Property Value

MessageType

The type of the message.

# Methods

## Equals(Message?)

Indicates whether the current object is equal to another object of the same type.

```
public bool Equals(Message? other)
```

### Parameters

`other` Message

An object to compare with this object.

### Returns

bool

true if the current object is equal to the `other` parameter; otherwise, false.

# Enum MessageType

Namespace: WebSockets.Core.Messages

Assembly: WebSockets.Core.dll

The types of messages.

```
public enum MessageType
```

## Fields

`Binary = 1`

`Close = 4`

`Ping = 2`

`Pong = 3`

`Text = 0`

# Class PingMessage

Namespace: WebSockets.Core.Messages

Assembly: WebSockets.Core.dll

A ping message.

```
public class PingMessage : DataMessage, IEquatable<Message>, IEquatable<DataMessage>
```

**Inheritance**

object ← Message ← DataMessage ← PingMessage

**Implements**

IEquatable<Message>, IEquatable<DataMessage>

**Inherited Members**

DataMessage.Data , DataMessage.Equals(DataMessage) , Message.Type , Message.Equals(Message) , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

## Constructors

### PingMessage(byte[])

Construct a ping message.

```
public PingMessage(byte[] data)
```

#### Parameters

`data` byte[]

The data that the pong message should return.

# Class PongMessage

Namespace: WebSockets.Core.Messages

Assembly: WebSockets.Core.dll

The message used to respond to a ping message.

```
public class PongMessage : DataMessage, IEquatable<Message>, IEquatable<DataMessage>
```

**Inheritance**

object ← Message ← DataMessage ← PongMessage

**Implements**

IEquatable<Message>, IEquatable<DataMessage>

**Inherited Members**

DataMessage.Data , DataMessage.Equals(DataMessage) , Message.Type , Message.Equals(Message) , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

## Constructors

### PongMessage(byte[])

Construct a pong message.

```
public PongMessage(byte[] data)
```

#### Parameters

`data` byte[]

The data sent by the ping messages.

# Class TextMessage

Namespace: WebSockets.Core.Messages

Assembly: WebSockets.Core.dll

A message with text data.

```
public class TextMessage : Message, IEquatable<Message>, IEquatable<TextMessage>
```

**Inheritance**

object ← Message ← TextMessage

**Implements**

IEquatable<Message>, IEquatable<TextMessage>

**Inherited Members**

Message.Type , Message.Equals(Message) , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

## Constructors

### TextMessage(string)

Construct the text message.

```
public TextMessage(string text)
```

#### Parameters

`text` string

The message text.

## Properties

### Text

The message text.

```
public string Text { get; }
```

### Property Value

string

The text.

## Methods

### Equals(TextMessage?)

Indicates whether the current object is equal to another object of the same type.

```
public bool Equals(TextMessage? other)
```

### Parameters

`other` TextMessage

An object to compare with this object.

### Returns

bool

true if the current object is equal to the `other` parameter; otherwise, false.