Using WebSockets (RFC 6455)

Version 2023.1
2024-07-11

InterSystems Corporation 1 Memorial Drive Cambridge MA 02142 www.intersystems.com

Using WebSockets (RFC 6455)
InterSystems Version 2023.1 2024-07-11
Copyright © 2024 InterSystems Corporation
All rights reserved.

InterSystems®, HealthShare Care Community®, HealthShare Unified Care Record®, IntegratedML®, InterSystems Caché®, InterSystems Ensemble® InterSystems HealthShare®, InterSystems IRIS®, and TrakCare are registered trademarks of InterSystems Corporation. HealthShare® CMS Solution Pack™ HealthShare® Health Connect Cloud™, InterSystems IRIS for Health™, InterSystems Supply Chain Orchestrator™, and InterSystems TotalView™ For Asset Management are trademarks of InterSystems Corporation. TrakCare is a registered trademark in Australia and the European Union.

All other brand or product names used herein are trademarks or registered trademarks of their respective companies or organizations.

This document contains trade secret and confidential information which is the property of InterSystems Corporation, One Memorial Drive, Cambridge, MA 02142, or its affiliates, and is furnished for the sole purpose of the operation and maintenance of the products of InterSystems Corporation. No part of this publication is to be used for any other purpose, and this publication is not to be reproduced, copied, disclosed, transmitted, stored in a retrieval system or translated into any human or computer language, in any form, by any means, in whole or in part, without the express prior written consent of InterSystems Corporation.

The copying, use and disposition of this document and the software programs described herein is prohibited except to the limited extent set forth in the standard software license agreement(s) of InterSystems Corporation covering such programs and related documentation. InterSystems Corporation makes no representations and warranties concerning such software programs other than those set forth in such standard software license agreement(s). In addition, the liability of InterSystems Corporation for any losses or damages relating to or arising out of the use of such software programs is limited in the manner set forth in such standard software license agreement(s).

THE FOREGOING IS A GENERAL SUMMARY OF THE RESTRICTIONS AND LIMITATIONS IMPOSED BY INTERSYSTEMS CORPORATION ON THE USE OF, AND LIABILITY ARISING FROM, ITS COMPUTER SOFTWARE. FOR COMPLETE INFORMATION REFERENCE SHOULD BE MADE TO THE STANDARD SOFTWARE LICENSE AGREEMENT(S) OF INTERSYSTEMS CORPORATION, COPIES OF WHICH WILL BE MADE AVAILABLE UPON REQUEST.

InterSystems Corporation disclaims responsibility for errors which may appear in this document, and it reserves the right, in its sole discretion and without notice, to make substitutions and modifications in the products and practices described in this document.

For Support questions about any InterSystems products, contact:

InterSystems Worldwide Response Center (WRC)
Tel: +1-617-621-0700
Tel: +44 (0) 844 854 2917
Email: support@InterSystems.com

Table of Contents

Using WebSockets (RFC 6455) **1**
1 WebSockets Protocol 1
2 WebSockets Client Code (JavaScript) 2
2.1 Creating a WebSocket 2
2.2 WebSocket Client Events 2
2.3 WebSocket Client Methods 2
3 WebSockets Server Code 3
3.1 WebSocket Server Events 3
3.2 WebSocket Server Methods 3
3.3 WebSocket Server Properties 4
4 WebSockets Server Example 4
5 WebSockets Server Asynchronous Operation 5
6 See Also 6

Using WebSockets (RFC 6455)

The WebSockets protocol (RFC 6455) addresses the fundamental requirement of allowing servers to proactively push messages to clients by providing a full-duplex message-oriented communications channel between a client and its server. The protocol is designed to operate, and hence be secured, over the standard TCP channel already established between the client and server and used to support the HTTP protocol between a web browser and web server.

The WebSockets protocol and its API are standardized by the W3C and the client part is included with HTML 5.

Intermediaries, such as proxies and firewalls, are expected to be aware of, and to support, the WebSockets protocol.

1 WebSockets Protocol

Creating a WebSocket involves an ordered exchange of messages between the client and the server. First, the WebSocket handshake must take place. The handshake is based on, and resembles, an HTTP message exchange so that it can pass without problem through existing HTTP infrastructure.

- Client sends handshake request for a WebSocket connection.
- Server sends handshake response (if it is able to).

The web server recognizes the conventional HTTP header structure in the handshake request message and sends a similarly constructed response message to the client indicating that it supports the WebSocket protocol - assuming it is able to. If both parties agree then the channel is switched from HTTP (http://) to the WebSockets protocol (ws://).

- When the protocol is successfully switched, the channel allows full duplex communication between the client and server.
- The data framing for individual messages is minimal.

Typical WebSocket Handshake Message from Client

```
GET /csp/user/MyApp.MyWebSocketServer.cls HTTP/1.1
Host: localhost
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: x3JJHMbDL1EzLkh9GBhXDw==
Sec-WebSocket-Protocol: chat
Sec-WebSocket-Version: 13
Origin: http://localhost
```

Typical WebSocket Handshake Message from Server

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: HSmrc0sMlYUkAGmm5OPpG2HaGWk=
Sec-WebSocket-Protocol: chat
```

Note how the client handshake message requests that the protocol be upgraded from HTTP to WebSocket. Note also the exchange of unique keys between the client (Sec-WebSocket-Key) and server (Sec-WebSocket-Accept).

2 WebSockets Client Code (JavaScript)

In the browser environment, the client uses JavaScript. Standard text books describe the usage model in detail. This section briefly describes the basics.

2.1 Creating a WebSocket

The first parameter represents the URL identifying the server end of the WebSocket application. The second parameter is optional, and if present, specifies the sub-protocol that the server must support for the WebSocket connection to be successful.

```
var ws = new WebSocket(url, [protocol]);
```

Example:

```
ws = new WebSocket(((window.location.protocol == "https:")
     ? "wss:" : "ws:")
     + "//" + window.location.host
     + /csp/user/MyApp.MyWebSocketServer.cls);
```

Note how the protocol is defined as either `ws` or `wss` depending on whether or not the underlying transport is secured using SSL/TLS.

The read-only attribute `ws.readyState` defines the state of the connection. It can take one of the following values:

- 0 The connection is not yet established.
- 1 The connection is established and communication is possible.
- 2 The connection is subject to the closing handshake.
- 3 The connection is closed or could not be opened.

The read-only attribute `ws.bufferedAmount` defines the number of bytes of UTF-8 text that have been queued using the **send()** method.

2.2 WebSocket Client Events

The following events are available.

- `ws.onopen` Fires when the socket connection is established.
- `ws.onmessage` Fires when the client receives data from the server.

Data received in event.data.

- `ws.onerror` Fires when an error occurs in the communication.
- `ws.onclose` Fires when the connection is closed.

2.3 WebSocket Client Methods

The following methods are available.

- `ws.send(data)` Transmit data to the client.
- `ws.close()` Close the connection.

3 WebSockets Server Code

The base InterSystems IRIS® class for implementing WebSocket Servers is %CSP.WebSocket

When the client requests a WebSocket connection, the initial HTTP request (the initial handshake message) instructs the CSP engine to initialize the application's WebSocket server. The WebSocket server is the class named in the requesting URL. For example, if your WebSocket server is called `MyApp.MyWebSocketServer` and is designed to operate in the USER namespace, then the URL used to request the WebSocket connection is:

```
/csp/user/MyApp.MyWebSocketServer.cls
```

3.1 WebSocket Server Events

To implement a WebSocket server, create a subclass of *%CSP.WebSocket* and define callbacks in that class as needed. Note that the web session is unlocked before calling any of these methods.

OnPreServer()

Implement this method to invoke code that should be executed before the WebSocket server is established. Changes to the SharedConnection property must be made here.

Server() (required)

Implement this method to create the WebSocket server. This is the server-side implementation of the WebSocket application. Messages can be exchanged with the client using the **Read()** and **Write()** methods. Use the **EndServer()** method to gracefully close the WebSocket from the server end.

OnPostServer()

Implement this method to invoke code that should be executed after the WebSocket server has closed.

3.2 WebSocket Server Methods

You can invoke the following methods from within these callbacks:

Read()

```
Method Read(ByRef len As %Integer = 32656,
     ByRef sc As %Status,
     timeout As %Integer = 86400) As %String
```

This method reads up to `len` characters from the client. If the call is successful the status (`sc`) is returned as `$$$OK`; otherwise one of the following error codes is returned:

- `$$$CSPWebSocketTimeout` The Read method has timed-out.
- `$$$CSPWebSocketClosed` The client has terminated the WebSocket.

Write()

```
Method Write(data As %String) As %Status
```

This method writes data to the client.

EndServer()

```
Method EndServer() As %Status
```

This method gracefully ends the WebSocket server by closing the connection with the client.

OpenServer()

```
Method OpenServer(WebSocketID As %String = "") As %Status
```

This method opens an existing WebSocket Server. Only a WebSocket operating asynchronously (`SharedConnection=1`) can be accessed using this method.

3.3 WebSocket Server Properties

You can set or get the following properties from within these callbacks:

SharedConnection (default: 0)

This property determines whether the communication between the client and WebSocket server should be over a dedicated Web Gateway connection or asynchronous over a pool of shared Web Gateway connections. This property must be set in the **OnPreServer()** method and may be set as follows:

- `SharedConnection=0` The WebSocket server communicates synchronously with the client via a dedicated Web Gateway connection. In this mode of operation the hosting connection is effectively 'private' to the application's WebSocket Server.
- `SharedConnection=1` The WebSocket server communicates asynchronously with the client via a pool of shared Web Gateway connections. Also, the socket times out once there is no activity for the CSP session timeout period.

WebSocketID

This property represents the unique identity of the WebSocket.

SessionId

This property represents the hosting CSP Session ID against which the WebSocket was created.

BinaryData

This property instructs the Web Gateway to bypass functionality that would otherwise interpret the transmitted data stream as UTF-8 encoded text and set the appropriate binary data fields in the WebSocket frame header.

This should be set to 1 before writing a stream of binary data to the client. For example:

```
Set ..BinaryData = 1
```

4 WebSockets Server Example

The following simple WebSocket server class accepts an incoming connection from a client and simply echos back data received.

The timeout is set to 10 seconds and each time the **Read()** method times-out a message is written to the client. This illustrates one of the key concepts underpinning WebSockets: initiating a message exchange with the client from the server.

Finally, the WebSocket closes gracefully if the client (i.e. user) sends the string `exit`.

```
Method OnPreServer() As %Status
{
   Quit $$$OK
}

Method Server() As %Status
{
   Set timeout=10
   For  {
      Set len=32656
      Set data=..Read(.len, .status, timeout)
      If $$$ISERR(status) {
                           If $$$GETERRORCODE(status) = $$$CSPWebSocketClosed {
                                                Quit
                           }
         If $$$GETERRORCODE(status) = $$$CSPWebSocketTimeout {
               Set status=..Write("Server timed-out at "_$Horolog)
         }
      }
      else {
         If data="exit" Quit
         Set status=..Write(data)
      }
   }
   Set status=..EndServer()
   Quit $$$OK
}

Method OnPostServer() As %Status
{
   Quit $$$OK
}
```

5 WebSockets Server Asynchronous Operation

The example given in the previous section illustrates a WebSocket server operating synchronously with the client over a dedicated InterSystems IRIS connection. When such a connection is established it is labeled as `WebSocket` in the status column of the Web Gateways Systems Status form. With this mode, the WebSocket is operating within the security context of the hosting web session and all properties associated with that session can be easily accessed.

With the asynchronous mode of operation (`SharedConnection=1`), the hosting connection is released as soon as the WebSocket Object is created and subsequent dialog with the client is over the pool of shared connections: messages from the client arrive via the conventional pool of Web Gateway connections to InterSystems IRIS and messages to the client are dispatched over the pool of Server connections that have been established between the Web Gateway and InterSystems IRIS.

In asynchronous mode, the WebSocket Server becomes detached from the main web session: the SessionId property holds the value of the hosting Session ID but an instance of the session object is not automatically created.

The example given previously can be run asynchronously simply by setting the SharedConnection property in the OnPre-Server() method. However, it is not necessary to have an InterSystems IRIS process permanently associated with the WebSocket. The **Server()** method can exit (and the hosting process halt) without closing the WebSocket. Provided the WebSocketID has been retained, the WebSocket can be subsequently opened in a different InterSystems IRIS process and communication with the client resumed.

In the following example, `MYAPP.SAVE()` and `MYAPP.RETRIEVE()` are placeholders for custom code you create for saving and retrieving a WebSocket ID.

Example:

```
Class MyApp.MyWebSocketServer Extends %CSP.WebSocket
{
    Method OnPreServer() As %Status
    {
        MYAPP.SAVE(..WebSocketID)
        Set ..SharedConnection = 1
        Quit $$$OK
    }

    Method Server() As %Status
    {
        Quit $$$OK
    }

    Method OnPostServer() As %Status
    {
        Quit $$$OK
    }
}
```

Note that the WebSocketID is retained for subsequent use in the **OnPreServer()** method. Note also, the setting of the SharedConnection property in the **OnPreServer()** method and that the **Server()** method simply exits.

Subsequently retrieving the WebSocketID:

```
Set WebSocketID = MYAPP.RETRIEVE()
```

Re-establishing a link with the client:

```
Set ws=##class(%CSP.WebSocket).%New()
Set %status = ws.OpenServer(WebSocketID)
```

Reading from and writing to the client:

```
Set %status=ws.Write(message)
Set data=ws.Read(.len, .%status, timeout)
```

Finally, closing the WebSocket from the server side:

```
Set %status=ws.EndServer()
```

6 See Also

- RFC 6455
- %CSP.WebSocket in the class reference