Introduction to SignalR Protocol Version 1.3

Table of Contents

[Introduction 4](#_Toc387406060)

[Negotiate: Protocol Negotiation 6](#_Toc387406061)

[Client Request 6](#_Toc387406062)

[URI Parameters 6](#_Toc387406063)

[Server Response 7](#_Toc387406064)

[Status Code and Headers 7](#_Toc387406065)

[Response Body 7](#_Toc387406066)

[SignalR Transports 8](#_Toc387406067)

[foreverFrame Transport 9](#_Toc387406068)

[Fiddler Hints 9](#_Toc387406069)

[Connect: Establishing the Connection 9](#_Toc387406070)

[Client Request 9](#_Toc387406071)

[URI Parameters 9](#_Toc387406072)

[Server Response 10](#_Toc387406073)

[Status Code and Headers 10](#_Toc387406074)

[Response Body 10](#_Toc387406075)

[Send: Sending Data to the Server 10](#_Toc387406076)

[Client Request 10](#_Toc387406077)

[URI Parameters 11](#_Toc387406078)

[Request Body 11](#_Toc387406079)

[Server Response 11](#_Toc387406080)

[Status Code and Headers 11](#_Toc387406081)

[Response Body 12](#_Toc387406082)

[Receiving Data from the Server 12](#_Toc387406083)

[Receiving KeepAlive Messages from the Server 13](#_Toc387406084)

[Abort: Stopping the SignalR connection/session 13](#_Toc387406085)

[Client Request 13](#_Toc387406086)

[URI Parameters 14](#_Toc387406087)

[Server Response 14](#_Toc387406088)

[Status Code 14](#_Toc387406089)

[Response body 14](#_Toc387406090)

[longPolling Transport 15](#_Toc387406091)

[Connect: Establishing the Connection 15](#_Toc387406092)

[Client Request 15](#_Toc387406093)

[URI Parameters 15](#_Toc387406094)

[Server Response 15](#_Toc387406095)

[Status Code and Headers 15](#_Toc387406096)

[Response Body 15](#_Toc387406097)

[Poll: Requesting Data from Server 16](#_Toc387406098)

[Client Request 16](#_Toc387406099)

[URI Parameters 16](#_Toc387406100)

[Server Response – When no Data is Available 17](#_Toc387406101)

[Status Code and Headers 17](#_Toc387406102)

[Response Body 17](#_Toc387406103)

[Server Response - When Data is Available 17](#_Toc387406104)

[Status Code and Headers 17](#_Toc387406105)

[Response Body 17](#_Toc387406106)

[Send: Sending Data to Server 18](#_Toc387406107)

[Client Request 18](#_Toc387406108)

[URI Parameters 18](#_Toc387406109)

[Request Body 18](#_Toc387406110)

[Server Response 18](#_Toc387406111)

[Status Code and Headers 18](#_Toc387406112)

[Response Body 18](#_Toc387406113)

[Ping: Ensuring that the server is still responsive 19](#_Toc387406114)

[Client Request 19](#_Toc387406115)

[URI Parameters 19](#_Toc387406116)

[Server Response 19](#_Toc387406117)

[Status Code 19](#_Toc387406118)

[Response Body 19](#_Toc387406119)

[Abort: Stopping the SignalR connection/session 20](#_Toc387406120)

[Client Request 20](#_Toc387406121)

[URI Parameters 20](#_Toc387406122)

[Server Response 20](#_Toc387406123)

[Status Code 20](#_Toc387406124)

[Response Body 20](#_Toc387406125)

[serverSentEvents Transport 21](#_Toc387406126)

[Fiddler Hints 21](#_Toc387406127)

[Connect: Establishing the Connection 21](#_Toc387406128)

[Client Request 21](#_Toc387406129)

[URI Parameters 21](#_Toc387406130)

[Server Response 21](#_Toc387406131)

[Status Code and Headers 21](#_Toc387406132)

[Response Body 22](#_Toc387406133)

[Send: Sending Data to the Server 22](#_Toc387406134)

[Client Request 22](#_Toc387406135)

[URI Parameters 22](#_Toc387406136)

[Request Body 22](#_Toc387406137)

[Server Response 22](#_Toc387406138)

[Status Code and Headers 22](#_Toc387406139)

[Response Body 23](#_Toc387406140)

[Receiving Data from the Server 23](#_Toc387406141)

[Receiving KeepAlive Messages from the Server 23](#_Toc387406142)

[Ping: Ensuring that the server is still responsive 24](#_Toc387406143)

[Client Request 24](#_Toc387406144)

[URI Parameters 24](#_Toc387406145)

[Server Response 24](#_Toc387406146)

[Status Code 24](#_Toc387406147)

[Response Body 24](#_Toc387406148)

[Abort: Stopping the SignalR connection/session 25](#_Toc387406149)

[Client Request 25](#_Toc387406150)

[URI Parameters 25](#_Toc387406151)

[Server Response 25](#_Toc387406152)

[Status Code and Headers 25](#_Toc387406153)

[Response Body 25](#_Toc387406154)

[webSockets Transport 26](#_Toc387406155)

[Fiddler Hints 26](#_Toc387406156)

[Connect: Establishing the Connection 26](#_Toc387406157)

[Client Request 26](#_Toc387406158)

[URI Parameters 26](#_Toc387406159)

[Request Headers 26](#_Toc387406160)

[Server Response 27](#_Toc387406161)

[Status Code and Response Headers 27](#_Toc387406162)

[Response Messages 27](#_Toc387406163)

[Sending and Receiving WebSocket Messages 28](#_Toc387406164)

[Receiving KeepAlive Messages from the Server 29](#_Toc387406165)

[Ping: Ensuring that the server is still responsive 30](#_Toc387406166)

[Client Request 30](#_Toc387406167)

[URI Parameters 30](#_Toc387406168)

[Server Response 30](#_Toc387406169)

[Status Code 30](#_Toc387406170)

[Response Body 30](#_Toc387406171)

[Abort: Stopping the SignalR connection/session 31](#_Toc387406172)

[Client Request 31](#_Toc387406173)

[URI Parameters 31](#_Toc387406174)

[Server Response 31](#_Toc387406175)

[Status Code and Headers 31](#_Toc387406176)

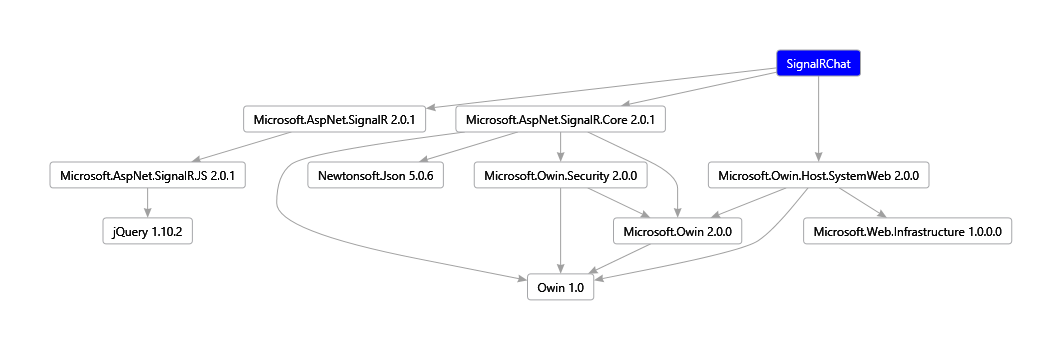
[Response Body 32](#_Toc387406177)

# Introduction

This document is a simple introduction to the latest version (1.3) of the SignalR protocol with emphasis on the client-hub interactions. (The reconnect request is not documented in this simple introduction.) The [auto-generated hub proxy](http://www.asp.net/signalr/overview/signalr-20/hubs-api/hubs-api-guide-javascript-client#genproxy) is used by the client.

Background on Hubs is available in the Connections and Hubs section of [Introduction to SignalR](http://www.asp.net/signalr/overview/signalr-20/getting-started-with-signalr-20/introduction-to-signalr).

Tests were performed with the current stable release:



A minor modification was made to the [Tutorial: Getting Started with SignalR 2.0](http://www.asp.net/signalr/overview/signalr-20/getting-started-with-signalr-20/tutorial-getting-started-with-signalr-20). The original code in index.html that starts the connection:

// Start the connection.

$.connection.hub.start().done(function () {

was updated to select a specific SignalR transport such as *foreverFrame*:

// Start the connection.

$.connection.hub.start(**{transport: 'foreverFrame'}**)**.**done(function () {

Wire traces were captured with [Fiddler](http://www.telerik.com/fiddler). Each trace captures the same steps:

1. The user logs in as “tester” when prompted.
2. The client negotiates with and connects to the ChatHub on the server.
3. The user types and sends the message “hello” to the ChatHub. This invokes the Send method on the ChatHub.

// Get the user name and store it to prepend to messages.

$('#displayname').val(prompt('Enter your name:', ''));

// Set initial focus to message input box.

$('#message').focus();

// Start the connection.

$.connection.hub.start({transport: 'foreverFrame'}).done(function () {

$('#sendmessage').click(function () {

// Call the Send method on the hub.

chat.server.send($('#displayname').val(), $('#message').val());

// Clear text box and reset focus for next comment.

$('#message').val('').focus();

});

});

});

1. When it receives the client message, the ChatHub (ChatHub.cs) invokes the broadcastMessage method on all of its clients:

public void Send(string name, string message)

{

// Call the broadcastMessage method to update clients.

Clients.All.broadcastMessage(name, message);

}

1. When it receives the server message, the client executes broadcastMessage and updates its display:

// Create a function that the hub can call to broadcast messages.

chat.client.broadcastMessage = function (name, message) {

// Html encode display name and message.

var encodedName = $('<div />').text(name).html();

var encodedMsg = $('<div />').text(message).html();

// Add the message to the page.

$('#discussion').append('<li><strong>' + encodedName

+ '</strong>:&nbsp;&nbsp;' + encodedMsg + '</li>');

};

1. The user types and sends the message “hello again” to the ChatHub.
2. Wash-Rinse-Repeat …
3. Finally, the user closes the browser tab which aborts the connection.

# Negotiate: Protocol Negotiation

During negotiation, the client requests the protocol version it prefers for the SignalR session. If the server supports the version, it accepts and responds with the same version. Otherwise, it responds with the version it is willing to use, which may differ from the client preference. Additional information is also returned:

* Hint whether WebSockets is supported by both the client and server
* Timeouts
* Opaque connection token to be returned with future client requests

## Client Request

GET /signalr/negotiate?**connectionData**=%5B%7B%22name%22%3A%22chathub%22%7D%5D&**clientProtocol**=1.3

|  |  |  |
| --- | --- | --- |
| Method | | Request URI |
| GET | http://*authority*/*path*/**negotiate** - as [authority](http://tools.ietf.org/html/rfc3986#section-3.2) and [path](http://tools.ietf.org/html/rfc3986#section-3.3) are defined in RFC3986. | |

### URI Parameters

|  |  |
| --- | --- |
| Parameter | Description |
| clientProtocol | For new client implementations, clientProtocol is set to “*1.3*”. Older clients may request the SignalR server to communicate with legacy versions of the protocol such as “*1.2*”. |
| connectionData | For hub applications, the connectionData is set to the array of hub proxies with active subscriptions and [percent-encoded](http://tools.ietf.org/html/rfc3986#section-2.1):  %5B%7B%22name%22%3A%22chathub%22%7D%5D  which decodes to:  [{“name”:”chathub”}]  in the chat tutorial. |

## Server Response

### Status Code and Headers

A successful operation returns status code 200 (OK) with a content type of JSON:

HTTP/1.1 200 OK

…

Content-Type: application/json; charset=UTF-8

### Response Body

A JSON Object is returned in the HTTP response:

{

"Url":"/signalr",

"ConnectionToken":"BOuXCzJsjW4PlgUQplGszjMVgcJmKa4kkN6BNi7CPzAa3n9nZzVT6lUIAxqV2ZC9xLEOeX/WxsUs+GIKKkDOIQTL0LtkKRIclc5j6VBPbSvOhmSpyDUDybOyQ6qvVnEw",

"ConnectionId":"60cbfef2-8959-4bf8-bd1e-f869211851c6",

"KeepAliveTimeout":20.0,

"DisconnectTimeout":30.0,

"TryWebSockets":true,

"ProtocolVersion":"1.3",

"TransportConnectTimeout":5.0

}

|  |  |
| --- | --- |
| Key | Value |
| Url | The base URI for future SignalR requests. |
| ConnectionToken | The connection token is documented in the [Introduction to SignalR Security](http://www.asp.net/signalr/overview/signalr-20/security/introduction-to-security#connectiontoken). From the client perspective, this is an opaque token which MUST be passed with each request to the server to validate the sender. |
| ConnectionId | The unique ID for the connection. Further details are available in [Mapping SignalR Users to Connections](http://www.asp.net/signalr/overview/signalr-1x/hubs-api/mapping-users-to-connections). |
| KeepAliveTimeout | A keepalive is a ‘heartbeat’ (empty) message sent from the server to the client at regular intervals when the connection is idle to keep the connection alive. If the heartbeat message is not received in *KeepAliveTimeout* seconds on an idle connection, then the client may need to reconnect.  KeepAlive is not implemented or required for the longPolling transport.  Further details on timeouts are available in [Understanding and Handling Connection Lifetime Events in SignalR](http://www.asp.net/signalr/overview/signalr-20/hubs-api/handling-connection-lifetime-events#timeoutkeepalive). |
| DisconnectTimeout | If a client is disconnected, then it SHOULD only attempt to reconnect to the server for *DisconnectTimeout* seconds.  Further details on timeouts are available in [Understanding and Handling Connection Lifetime Events in SignalR](http://www.asp.net/signalr/overview/signalr-20/hubs-api/handling-connection-lifetime-events#timeoutkeepalive). |
| TryWebSockets | Boolean. Hints whether the WebSockets protocol is supported by both the client and the server.  If true, the client MAY select Websockets as the transport in the connect request to the server. If false, the client MUST NOT use this transport. |
| ProtocolVersion | MUST be “1.3”. |
| TransportConnectTimeout | If the client does not receive a connect response within *TransportConnectTimeout* seconds, then it SHOULD either fail the connect request or retry with another transport.  Further details on timeouts are available in [Understanding and Handling Connection Lifetime Events in SignalR](http://www.asp.net/signalr/overview/signalr-20/hubs-api/handling-connection-lifetime-events#timeoutkeepalive). |

# SignalR Transports

A transport is passed in the transport URL parameter by the client in the ***connect*** request to the server. The following transports are supported, but may not be available on all platforms:

* foreverFrame
* longPolling
* serverSentEvents
* webSockets

Further details are available in the *Transports and fallbacks* section of [Introduction to SignalR](http://www.asp.net/signalr/overview/signalr-20/getting-started-with-signalr-20/introduction-to-signalr) and in [Supported Platforms](http://www.asp.net/signalr/overview/signalr-20/getting-started-with-signalr-20/supported-platforms).

# foreverFrame Transport

The [forever-frame technique](http://cometdaily.com/2007/11/05/the-forever-frame-technique/) relies on a [hidden iframe](http://www.w3.org/TR/html5/embedded-content-0.html#the-iframe-element) on the client which receives a *never-ending* response from the server based on the [HTTP/1.1 chunked transfer encoding](http://tools.ietf.org/html/rfc2616#section-3.6.1). The response is an incremental stream of <script> tags which are interpreted and executed by the client as each one is received.

## Fiddler Hints

To examine the response stream:

1. [Enable Streaming Mode](http://fiddlerbook.com/Fiddler/help/streaming.asp)
2. Right-click the connect request session line and select “COMETpeek”.
3. Inspect the Raw Response

## Connect: Establishing the Connection

### Client Request

GET /signalr/connect?**transport**=foreverFrame&**connectionToken**=G1OrxHKwhcPFdVs1Joi78OekdameN7vz%2FZwNSvNGVxspPphgihN0NharYGmEcRAgR5fQkN5smZGIcnB3nfzH%2F217V5q0OUm%2FxgMzWYyEcBSI2XmmUKpoKHCj2vb8ap2s&**connectionData**=%5B%7B%22name%22%3A%22chathub%22%7D%5D&**tid**=10&**frameId**=1

|  |  |  |
| --- | --- | --- |
| Method | | Request URI |
| GET | http://*authority*/*path*/**connect** | |

#### URI Parameters

|  |  |
| --- | --- |
| Parameter | Description |
| connectionData | For hub applications, the connectionData is set to the array of hub proxies with active subscriptions and [percent-encoded](http://tools.ietf.org/html/rfc3986#section-2.1):  %5B%7B%22name%22%3A%22chathub%22%7D%5D  which decodes to:  [{“name”:”chathub”}]  in the chat tutorial. |
| connectionToken | The connectionToken MUST be set to the value of the *ConnectionToken* returned by the server in its response to the ***negotiate*** request. |
| frameId | The frameId is a unique identifier for each hidden iframe. |
| tid | The tid (transport id) is a unique identifier used to differentiate requests (HTTP GET) for internal SignalR logging and diagnostics. |
| transport | The transport MUST be set to “*foreverFrame”*. |

### Server Response

#### Status Code and Headers

A successful operation returns status code 200 (OK) and a chunked transfer encoding:

HTTP/1.1 **200 OK**

…

Transfer-Encoding: **chunked**

#### Response Body

After the successful connect, the server responds with a stream of HTML:

<!DOCTYPE html>

<html>

<head>

<title>SignalR Forever Frame Transport Stream</title>

<script>

var $ = window.parent.jQuery,

ff = $ ? $.signalR.transports.foreverFrame : null,

c = ff ? ff.getConnection('1') : null,

r = ff ? ff.receive : function () {};

ff ? ff.started(c) : '';

</script>

</head>

<body>

<script>

r(c, {

"C": "d-4B7C2E0A-B,0|C,0|D,1|E,0",

"S": 1,

"M": []

});

</script>

…

|  |  |
| --- | --- |
| Response Body | Interpretation |
| <script>  r(c, {  "C": "d-4B7C2E0A-B,0|C,0|D,1|E,0",  "S": 1,  "M": []  });  </script> | A JSON object is passed as the data parameter to the receive (r) script.  State (S) indicates that the connection is in the process of initializing.  Messages (M[]) has no elements which indicates that no actions are required to be processed. |

See *Receiving Data from the Server* for further details.

## Send: Sending Data to the Server

### Client Request

POST /signalr/send?**transport**=foreverFrame&**connectionToken**=G1OrxHKwhcPFdVs1Joi78OekdameN7vz%2FZwNSvNGVxspPphgihN0NharYGmEcRAgR5fQkN5smZGIcnB3nfzH%2F217V5q0OUm%2FxgMzWYyEcBSI2XmmUKpoKHCj2vb8ap2s&**connectionData**=%5B%7B%22name%22%3A%22chathub%22%7D%5D

|  |  |  |
| --- | --- | --- |
| Method | | Request URI |
| POST | http://*authority*/*path*/**send** | |

#### URI Parameters

|  |  |
| --- | --- |
| Parameter | Description |
| connectionData | For hub applications, the connectionData is set to the array of hub proxies with active subscriptions and [percent-encoded](http://tools.ietf.org/html/rfc3986#section-2.1):  %5B%7B%22name%22%3A%22chathub%22%7D%5D  which decodes to:  [{“name”:”chathub”}]  in the chat tutorial. |
| connectionToken | The connectionToken MUST be set to the value of the *ConnectionToken* returned by the server in its response to the ***negotiate*** request. |
| transport | The transport MUST be set to “*foreverFrame”*. |

### Request Body

The user (“tester”) sends two messages to the chat hub: “hello” and “hello again”:

|  |  |
| --- | --- |
| Request Body | Interpretation |
| data=%7B%22H%22%3A%22chathub%22%2C%22M%22%3A%22Send%22%2C%22A%22%3A%5B%22tester%22%2C%22hello%22%5D%2C%22I%22%3A0%7D  which decodes to:  data={"H":"chathub","M":"Send","A":["tester","hello"],"I":0} | The client invokes the *Send* method (M) with the argument list (A) *[“tester, “hello”]* on the hub proxy (H) named “*chathub*”  The message identifier (I) starts at 0 and is incremented by 1 for each subsequent message. |
| data=%7B%22H%22%3A%22chathub%22%2C%22M%22%3A%22Send%22%2C%22A%22%3A%5B%22tester%22%2C%22hello+again%22%5D%2C%22I%22%3A1%7D  which decodes to:  data={"H":"chathub","M":"Send","A":["tester","hello again"],"I":1} | The client invokes the *Send* method (M) with the argument list (A) *[“tester, “hello again”]* on the hub proxy (H) named “*chathub”*  The message identifier (I) is incremented from 0 to 1 for the second message. |

### Server Response

#### Status Code and Headers

A successful operation returns status code 200 (OK) with a content type of JSON:

HTTP/1.1 **200 OK**

…

Content-Type: **application/json**; charset=UTF-8

#### Response Body

The message identifier from the Send (POST) request is returned in the response content. For example, the response to the “hello” message is the JSON object {"I":"0"}.The response to the “hello again” message is {"I":"1"}.

## Receiving Data from the Server

In response to each message sent the client, the hub returns a <script> element:

<!DOCTYPE html>

<html>

<head>

<title>SignalR Forever Frame Transport Stream</title>

<script>

var $ = window.parent.jQuery,

ff = $ ? $.signalR.transports.foreverFrame : null,

c = ff ? ff.getConnection('1') : null,

r = ff ? ff.receive : function () {};

ff ? ff.started(c) : '';

</script>

</head>

<body>

<script>

r(c, {

"C": "d-4B7C2E0A-B,0|C,0|D,1|E,0",

"S": 1,

"M": []

});

</script>

<script>

r(c, {

"C": "d-4B7C2E0A-B,1|C,0|D,1|E,0",

"M": [{

"H": "ChatHub",

"M": "broadcastMessage",

"A": ["tester", "hello"]

}]

});

</script>

<script>

r(c, {});

</script>

…

<script>

r(c, {

"C": "d-4B7C2E0A-B,2|C,0|D,1|E,0",

"M": [{

"H": "ChatHub",

"M": "broadcastMessage",

"A": ["tester", "hello again"]

}]

});

</script>

<script>

r(c, {});

</script>

…

|  |  |
| --- | --- |
| Response Body | Interpretation |
| <script>  r(c, {  "C": "d-4B7C2E0A-B,1|C,0|D,1|E,0",  "M": [{  "H": "ChatHub",  "M": "broadcastMessage",  "A": ["tester", "hello"]  }]  });  </script> | In response to the first message sent by the client, a JSON object is passed as the data parameter to the receive (r) script.  The Message (M[]) indicates that:  The hub (H) named “ChatHub” invokes the *broadcastMessage* method (M) with argument list (A) ["tester", "hello"] on the client. |
| <script>  r(c, {  "C": "d-4B7C2E0A-B,2|C,0|D,1|E,0",  "M": [{  "H": "ChatHub",  "M": "broadcastMessage",  "A": ["tester", "hello again"]  }]  });  </script> | In response to the second message sent by the client, a JSON object is passed as the data parameter to the receive (r) script.  The Message (M[]) indicates that:  The hub (H) named “ChatHub” invokes the *broadcastMessage* method (M) with argument list (A) ["tester", "hello again"] on the client (C). |

### Receiving KeepAlive Messages from the Server

The server sends <script> elements with no data parameter ({}) to the client to keep the connection alive:

<script>

r(c, {});

</script>

<script>

r(c, {});

</script>

## Abort: Stopping the SignalR connection/session

### Client Request

POST /signalr/abort?**transport**=foreverFrame&**connectionToken**=G1OrxHKwhcPFdVs1Joi78OekdameN7vz%2FZwNSvNGVxspPphgihN0NharYGmEcRAgR5fQkN5smZGIcnB3nfzH%2F217V5q0OUm%2FxgMzWYyEcBSI2XmmUKpoKHCj2vb8ap2s&**connectionData**=%5B%7B%22name%22%3A%22chathub%22%7D%5D

|  |  |  |
| --- | --- | --- |
| Method | | Request URI |
| POST | http://*authority*/*path*/**abort** | |

#### URI Parameters

|  |  |
| --- | --- |
| Parameter | Description |
| connectionData | For hub applications, the connectionData is set to the array of hub proxies with active subscriptions and [percent-encoded](http://tools.ietf.org/html/rfc3986#section-2.1):  %5B%7B%22name%22%3A%22chathub%22%7D%5D  which decodes to:  [{“name”:”chathub”}]  in the chat tutorial. |
| connectionToken | The connectionToken MUST be set to the value of the *ConnectionToken* returned by the server in its response to the ***negotiate*** request. |
| transport | The transport MUST be set to “*foreverFrame*”. |

## Server Response

### Status Code

A successful operation returns status code 200 (OK).

### Response body

The response body is empty.

# longPolling Transport

Long Polling is a variation of polling based on a hanging-GET operation. The client makes a HTTP GET request to the server. If the server does not have available data, it delays returning a response to the HTTP GET request until either new data arrives or a timeout occurs.

## Connect: Establishing the Connection

### Client Request

GET /signalr/connect?**transport**=longPolling&**connectionToken**=9zyu8mdmEqHcOW4H4PmiOfRsYqzS8L5CkfsuwxXNanlt0IPELZRoFjZpMeTWKVovxcGMBrbQwBe0mJ2wU3oNtie8h6SG20y5iTl0Qgx4WeiP1XHFNwne1ifuX5Y92p%2B9&**connectionData**=%5B%7B%22name%22%3A%22chathub%22%7D%5D&**tid**=6

|  |  |  |
| --- | --- | --- |
| Method | | Request URI |
| GET | http://*authority*/*path*/**connect** | |

#### URI Parameters

|  |  |
| --- | --- |
| Parameter | Description |
| connectionData | For hub applications, the connectionData is set to the array of hub proxies with active subscriptions and [percent-encoded](http://tools.ietf.org/html/rfc3986#section-2.1):  %5B%7B%22name%22%3A%22chathub%22%7D%5D  which decodes to:  [{“name”:”chathub”}]  in the chat tutorial. |
| connectionToken | The connectionToken MUST be set to the value of the *ConnectionToken* returned by the server in its response to the ***negotiate*** request. |
| tid | The tid (transport id) is a unique identifier used to differentiate requests (HTTP GET) for internal SignalR logging and diagnostics. |
| transport | The transport MUST be set to “*longPolling”*. |

### Server Response

#### Status Code and Headers

A successful operation returns status code 200 (OK) with a content type of JSON:

HTTP/1.1 **200 OK**

…

Transfer-Encoding: chunked

Content-Type: application/**json**; charset=UTF-8

#### Response Body

A JSON Object is returned in the HTTP response:

{

"C":" d-E36DDACC-B,0|C,0|D,1|E,0",

"S":1,

"M":[]

}

## Poll: Requesting Data from Server

Poll is based on the *hanging-GET* or long poll design pattern.

It’s straightforward to demonstrate that the server delays its response to the Poll (HTTP GET) request until either data is available or a timeout occurs.

Fiddler properties include time information for the traces of the interleaved *Poll* (GET) and *Send* (POST) client operations. The client *Polls* for new data and the response hangs from 14:54:54.772 until 14:55:18.511 when the interleaved *Send* request arrives at 14:55:17.958 and delivers new data to the server which is then be returned in the pending *Poll* response:

|  |  |
| --- | --- |
| Timestamp | Fiddler Action |
| 14:54:54.772 | Poll Request - ServerGotRequest |
| - | Delay to demonstrate that GET is hanging |
| 14:55:17.955 | Send Request – ClientDoneRequest |
| 14:55:17.958 | Send Request - ServerGotRequest |
| 14:55:18.511 | Poll Response – ServerBeginResponse |
| 14:55:18.515 | Poll Response – ClientDoneResponse |
| 14:55:18.582 | Send Response - ServerDoneResponse |

### Client Request

GET /signalr/poll?**transport**=longPolling&**connectionToken**=9zyu8mdmEqHcOW4H4PmiOfRsYqzS8L5CkfsuwxXNanlt0IPELZRoFjZpMeTWKVovxcGMBrbQwBe0mJ2wU3oNtie8h6SG20y5iTl0Qgx4WeiP1XHFNwne1ifuX5Y92p%2B9&**messageId**=d-E36DDACC-B%2C0%7CC%2C0%7CD%2C1%7CE%2C0&**connectionData**=%5B%7B%22name%22%3A%22chathub%22%7D%5D&**tid**=2

|  |  |  |
| --- | --- | --- |
| Method | | Request URI |
| GET | http://*authority*/*path*/**poll** | |

#### URI Parameters

|  |  |
| --- | --- |
| Parameter | Description |
| connectionData | For hub applications, the connectionData is set to the array of hub proxies with active subscriptions and [percent-encoded](http://tools.ietf.org/html/rfc3986#section-2.1):  %5B%7B%22name%22%3A%22chathub%22%7D%5D  which decodes to:  [{“name”:”chathub”}]  in the chat tutorial. |
| connectionToken | The connectionToken MUST be set to the value of the *ConnectionToken* returned by the server in its response to the ***negotiate*** request. |
| messageId | The messageId MUST be set to the [percent-encoded](http://tools.ietf.org/html/rfc3986#section-2.1) value of the last opaque C: key received in a HTTP response. For example:  d-E36DDACC-B,0|C,0|D,1|E,0  which encodes to:  d-E36DDACC-B%2C0%7CC%2C0%7CD%2C1%7CE%2C0 |
| tid | The tid (transport id) is a unique identifier used to differentiate requests (HTTP GET) for internal SignalR logging and diagnostics. |
| transport | The transport MUST be set to “longPolling”. |

### Server Response – When no Data is Available

#### Status Code and Headers

A successful operation returns status code 200 (OK) with a content type of JSON:

HTTP/1.1 **200 OK**

…

Transfer-Encoding: chunked

Content-Type: application/**json**; charset=UTF-8

#### Response Body

A JSON Object is returned in the HTTP response:

{

"C":" d-E36DDACC-B,0|C,0|D,1|E,0",

"M":[]

}

There are no Message (M[]) elements, so no new data is available to process.

### Server Response - When Data is Available

#### Status Code and Headers

A successful operation returns status code 200 (OK) with a content type of JSON:

HTTP/1.1 **200 OK**

…

Transfer-Encoding: chunked

Content-Type: application/**json**; charset=UTF-8

#### Response Body

A JSON Object is returned in the HTTP response:

{

"C":" d-E36DDACC-B,1|C,0|D,1|E,0",

"M":[{"H":"ChatHub","M":"broadcastMessage","A":["tester","hello"]

}

## Send: Sending Data to Server

### Client Request

POST /signalr/send?**transport**=longPolling&**connectionToken**=9zyu8mdmEqHcOW4H4PmiOfRsYqzS8L5CkfsuwxXNanlt0IPELZRoFjZpMeTWKVovxcGMBrbQwBe0mJ2wU3oNtie8h6SG20y5iTl0Qgx4WeiP1XHFNwne1ifuX5Y92p%2B9&**connectionData**=%5B%7B%22name%22%3A%22chathub%22%7D%5D

|  |  |  |
| --- | --- | --- |
| Method | | Request URI |
| POST | http://*authority*/*path*/**send** | |

#### URI Parameters

|  |  |
| --- | --- |
| Parameter | Description |
| connectionData | For hub applications, the connectionData is set to the array of hub proxies with active subscriptions and [percent-encoded](http://tools.ietf.org/html/rfc3986#section-2.1):  %5B%7B%22name%22%3A%22chathub%22%7D%5D  which decodes to:  [{“name”:”chathub”}]  in the chat tutorial. |
| connectionToken | The connectionToken MUST be set to the value of the *ConnectionToken* returned by the server in its response to the ***negotiate*** request. |
| transport | The transport must be set to “*longPolling*”. |

#### Request Body

The request body is the same format documented for the foreverFrame *Send* Request Body.

### Server Response

#### Status Code and Headers

A successful operation returns status code 200 (OK) with a content type of JSON:

HTTP/1.1 **200 OK**

…

Content-Type: **application/json**; charset=UTF-8

#### Response Body

The message identifier from the Send (POST) request is returned in the response content. For example, the response to the “hello” message is the JSON object {"I":"0"}.The response to the “hello again” message is {"I":"1"}.

# Ping: Ensuring that the server is still responsive

The client MAY send a SignalR ***ping*** request to the server to determine if it’s still responsive to requests. This is only supported by the following transports:

* longPolling
* serverSentEvents
* webSockets

## Client Request

GET /signalr/ping?**connectionData**=%5B%7B%22name%22%3A%22chathub%22%7D%5D

|  |  |  |
| --- | --- | --- |
| Method | | Request URI |
| GET | http://*authority*/*path*/**ping** | |

### URI Parameters

|  |  |
| --- | --- |
| Parameter | Description |
| connectionData | For hub applications, the connectionData is set to the array of hub proxies with active subscriptions and [percent-encoded](http://tools.ietf.org/html/rfc3986#section-2.1):  %5B%7B%22name%22%3A%22chathub%22%7D%5D  which decodes to:  [{“name”:”chathub”}]  in the chat tutorial. |

## Server Response

### Status Code

A successful operation returns status code 200 (OK) with a content type of JSON:

HTTP/1.1 200 OK

…

Content-Type: application/json; charset=UTF-8

### Response Body

A JSON Object is returned in the HTTP response:

{"Response":"pong"}

## Abort: Stopping the SignalR connection/session

### Client Request

POST /signalr/abort?**transport**=longPolling&**connectionToken**=9zyu8mdmEqHcOW4H4PmiOfRsYqzS8L5CkfsuwxXNanlt0IPELZRoFjZpMeTWKVovxcGMBrbQwBe0mJ2wU3oNtie8h6SG20y5iTl0Qgx4WeiP1XHFNwne1ifuX5Y92p%2B9&**connectionData**=%5B%7B%22name%22%3A%22chathub%22%7D%5D

|  |  |  |
| --- | --- | --- |
| Method | | Request URI |
| POST | http://*authority*/*path*/**abort** | |

#### URI Parameters

|  |  |
| --- | --- |
| Parameter | Description |
| connectionData | For hub applications, the connectionData is set to the array of hub proxies with active subscriptions and [percent-encoded](http://tools.ietf.org/html/rfc3986#section-2.1):  %5B%7B%22name%22%3A%22chathub%22%7D%5D  which decodes to:  [{“name”:”chathub”}]  in the chat tutorial. |
| connectionToken | The connectionToken MUST be set to the value of the *ConnectionToken* returned by the server in its response to the ***negotiate*** request. |
| transport | The transport must be set to “*longPolling*”. |

### Server Response

#### Status Code

A successful operation returns status code 200 (OK).

#### Response Body

The response body is empty.

# serverSentEvents Transport

The serverSentEvents transport implements the W3C [Server-Sent Events](http://www.w3.org/TR/eventsource/) specification. After the connection is established, the server responds with a stream of events.

## Fiddler Hints

To examine the response stream:

1. [Enable Streaming Mode](http://fiddlerbook.com/Fiddler/help/streaming.asp)
2. Right-click the connect request session line and select “COMETpeek”.
3. Inspect the Raw Response

## Connect: Establishing the Connection

GET /signalr/connect?**transport**=serverSentEvents&**connectionToken**=UGeKKpr%2BObYDHSomtXXPOX%2Bu5Ai5%2FdsHZ25xdwBiKB%2By6RLAFQTFuN829CcHrBtvRuxLDeWRSlz5qQUVSZBpM1wvzsmH3Z2AiJVxZ%2BBEsJ7zsCC9lSr282BgI40z4DeA&**connectionData**=%5B%7B%22name%22%3A%22chathub%22%7D%5D&**tid**=5

### Client Request

|  |  |  |
| --- | --- | --- |
| Method | | Request URI |
| GET | http://*authority*/*path*/**connect** | |

#### URI Parameters

|  |  |
| --- | --- |
| Parameter | Description |
| connectionData | For hub applications, the connectionData is set to the array of hub proxies with active subscriptions and [percent-encoded](http://tools.ietf.org/html/rfc3986#section-2.1):  %5B%7B%22name%22%3A%22chathub%22%7D%5D  which decodes to:  [{“name”:”chathub”}]  in the chat tutorial. |
| connectionToken | The connectionToken MUST be set to the value of the *ConnectionToken* returned by the server in its response to the ***negotiate*** request. |
| tid | The tid (transport id) is a unique identifier used to differentiate requests (HTTP GET) for internal SignalR logging and diagnostics. |
| transport | The transport MUST be set to “*serverSentEvents”*. |

### Server Response

#### Status Code and Headers

A successful operation returns status code 200 (OK) with a content type of event-stream:

HTTP/1.1 **200 OK**

…

Content-Type: text/**event-stream**

#### Response Body

After the successful connect, the server responds with the following events:

data: initialized

data: {"C":"d-A2CFDE4C-B,0|C,0|D,1|E,0","S":1,"M":[]}

## Send: Sending Data to the Server

### Client Request

POST /signalr/send?**transport**=serverSentEvents&**connectionToken**=UGeKKpr%2BObYDHSomtXXPOX%2Bu5Ai5%2FdsHZ25xdwBiKB%2By6RLAFQTFuN829CcHrBtvRuxLDeWRSlz5qQUVSZBpM1wvzsmH3Z2AiJVxZ%2BBEsJ7zsCC9lSr282BgI40z4DeA&**connectionData**=%5B%7B%22name%22%3A%22chathub%22%7D%5D

|  |  |  |
| --- | --- | --- |
| Method | | Request URI |
| POST | http://*authority*/*path*/**send** | |

#### URI Parameters

|  |  |
| --- | --- |
| Parameter | Description |
| connectionData | For hub applications, the connectionData is set to the array of hub proxies with active subscriptions and [percent-encoded](http://tools.ietf.org/html/rfc3986#section-2.1):  %5B%7B%22name%22%3A%22chathub%22%7D%5D  which decodes to:  [{“name”:”chathub”}]  in the chat tutorial. |
| connectionToken | The connectionToken MUST be set to the value of the *ConnectionToken* returned by the server in its response to the ***negotiate*** request. |
| transport | The transport MUST be set to “*serverSentEvents”*. |

#### Request Body

The request body is the same format documented for the foreverFrame *Send* Request Body.

### Server Response

#### Status Code and Headers

A successful operation returns status code 200 (OK) with a content type of JSON:

HTTP/1.1 **200 OK**

…

Content-Type: **application/json**; charset=UTF-8

#### Response Body

The message identifier from the Send (POST) request is returned in the response content. For example, the response to the “hello” message is the JSON object {"I":"0"}.The response to the “hello again” message is {"I":"1"}.

## Receiving Data from the Server

A *never-ending* stream of events are received on the client. In response to each message sent by the client, the hub on the server pushes an event:

data: {"C":"d-A2CFDE4C-B,1|C,0|D,1|E,0","M":[{"H":"ChatHub","M":"broadcastMessage","A":["tester","hello"]}]}

data: {"C":"d-A2CFDE4C-B,2|C,0|D,1|E,0","M":[{"H":"ChatHub","M":"broadcastMessage","A":["tester","hello again"]}]}

### Receiving KeepAlive Messages from the Server

The server sends events with no data ({}) to the client to keep the connection alive:

data: {}

…

data: {}

data: {}

# Ping: Ensuring that the server is still responsive

The client MAY send a SignalR ***ping*** request to the server to determine if it’s still responsive to requests. This is only supported by the following transports:

* longPolling
* serverSentEvents
* webSockets

## Client Request

GET /signalr/ping?**connectionData**=%5B%7B%22name%22%3A%22chathub%22%7D%5D

|  |  |  |
| --- | --- | --- |
| Method | | Request URI |
| GET | http://*authority*/*path*/**ping** | |

### URI Parameters

|  |  |
| --- | --- |
| Parameter | Description |
| connectionData | For hub applications, the connectionData is set to the array of hub proxies with active subscriptions and [percent-encoded](http://tools.ietf.org/html/rfc3986#section-2.1):  %5B%7B%22name%22%3A%22chathub%22%7D%5D  which decodes to:  [{“name”:”chathub”}]  in the chat tutorial. |

## Server Response

### Status Code

A successful operation returns status code 200 (OK) with a content type of JSON:

HTTP/1.1 200 OK

…

Content-Type: application/json; charset=UTF-8

### Response Body

A JSON Object is returned in the HTTP response:

{"Response":"pong"}

## Abort: Stopping the SignalR connection/session

### Client Request

POST /signalr/abort?**transport**=serverSentEvents&**connectionToken**=UGeKKpr%2BObYDHSomtXXPOX%2Bu5Ai5%2FdsHZ25xdwBiKB%2By6RLAFQTFuN829CcHrBtvRuxLDeWRSlz5qQUVSZBpM1wvzsmH3Z2AiJVxZ%2BBEsJ7zsCC9lSr282BgI40z4DeA&**connectionData**=%5B%7B%22name%22%3A%22chathub%22%7D%5D

|  |  |  |
| --- | --- | --- |
| Method | | Request URI |
| POST | http://*authority*/*path*/**abort** | |

#### URI Parameters

|  |  |
| --- | --- |
| Parameter | Description |
| connectionData | For hub applications, the connectionData is set to the array of hub proxies with active subscriptions and [percent-encoded](http://tools.ietf.org/html/rfc3986#section-2.1):  %5B%7B%22name%22%3A%22chathub%22%7D%5D  which decodes to:  [{“name”:”chathub”}]  in the chat tutorial. |
| connectionToken | The connectionToken MUST be set to the value of the *ConnectionToken* returned by the server in its response to the ***negotiate*** request. |
| transport | The transport MUST be set to “*serverSentEvents*”. |

### Server Response

#### Status Code and Headers

A successful operation returns status code 200 (OK).

#### Response Body

The response body is empty.

# webSockets Transport

The webSockets transport implements [RFC6455 – The WebSocket protocol](http://datatracker.ietf.org/doc/rfc6455/?include_text=1). An introduction to the WebSockets Protocol is available on the [IE Blog](http://blogs.msdn.com/b/ie/archive/2012/03/19/websockets-in-windows-consumer-preview.aspx).

## Fiddler Hints

Detailed information and Fiddler scripts are available in [Debug / Inspect WebSocket traffic with Fiddler](http://www.codeproject.com/Articles/718660/Debug-Inspect-WebSocket-traffic-with-Fiddler).

## Connect: Establishing the Connection

### Client Request

GET

/signalr/connect?**transport**=webSockets&**connectionToken**=e8NhgP5Sb3QhDoDu38Mk%2BhbyKM0wtIJv4PfEDtWHKddL83Zjrsu9PLK5QYSmyAMnWxMMpvGG3h8IlRsncc2soMY1mrQ7%2FyumKH3v0yEyKkcKkOOA7jF4EpLjR4I8vZr9&**connectionData**=%5B%7B%22name%22%3A%22chathub%22%7D%5D&**tid**=1

|  |  |  |
| --- | --- | --- |
| Method | | Request URI |
| GET | http://*authority*/*path*/**connect** | |

#### URI Parameters

|  |  |
| --- | --- |
| Parameter | Description |
| connectionData | For hub applications, the connectionData is set to the array of hub proxies with active subscriptions and [percent-encoded](http://tools.ietf.org/html/rfc3986#section-2.1):  %5B%7B%22name%22%3A%22chathub%22%7D%5D  which decodes to:  [{“name”:”chathub”}]  in the chat tutorial. |
| connectionToken | The connectionToken MUST be set to the value of the *ConnectionToken* returned by the server in its response to the ***negotiate*** request. |
| tid | The tid (transport id) is a unique identifier used to differentiate requests (HTTP GET) for internal SignalR logging and diagnostics. |
| transport | The transport MUST be set to “*webSockets”*. |

#### Request Headers

The GET request MUST include the required HTTP request headers to negotiate an upgrade to the WebSockets protocol as specified in [RFC6455: Reading the Client's Opening Handshake](http://tools.ietf.org/html/rfc6455#section-4.2.1):

Sec-WebSocket-Key: ccfkGVDlqO7lkuLs5y8h8g==

Connection: Upgrade

Upgrade: Websocket

Sec-WebSocket-Version: 13

### Server Response

#### Status Code and Response Headers

A successful operation returns status code 101 (Switching Protocols).

HTTP/1.1 101 Switching Protocols

The HTTP response MUST include the required HTTP response headers as specified in [RFC6455: Sending the Server's Opening Handshake](http://tools.ietf.org/html/rfc6455#section-4.2.2):

Upgrade: Websocket

…

Sec-WebSocket-Accept: kDVE0tQEYaw+8rU88SfBcFFGKLg=

Connection: Upgrade

The application-layer protocol switches from HTTP to WebSockets which uses the previously established TCP connection. **HTTP is completely out of the picture at this point.** Using the WebSocket wire protocol, messages can now be sent or received by either endpoint at any time.

#### Response Messages

Upon connection, the SignalR server immediately sends a WebSocket [text data frame](http://tools.ietf.org/html/rfc6455#section-5.6) with a UTF-8 encoded payload:

16:21:32:5193 Upgrading Session #14 to websocket

16:21:32:7849 WSSession14.WebSocket'WebSocket #14'

MessageID: Server.1

MessageType: Text

PayloadString: {"C":"d-94DAFC36-B,0|C,0|D,1|E,0","S":1,"M":[]}

16:21:32:7849 WSSession14.WebSocket'WebSocket #14'

MessageID: Server.2

MessageType: Continuation

PayloadString:

## Sending and Receiving WebSocket Messages

|  |  |
| --- | --- |
| Client Sends | Server Responds |
| 16:22:37:0560 WSSession14.WebSocket'WebSocket #14'  MessageID: Client.9  MessageType: Text  PayloadString: {"H":"chathub","M":"Send","A":["tester","hello"],"I":0} | 16:22:37:5092 WSSession14.WebSocket'WebSocket #14'  MessageID: Server.10  MessageType: Text  PayloadString: {"C":"d-94DAFC36-B,1|C,0|D,1|E,0","M":[  16:22:37:5248 WSSession14.WebSocket'WebSocket #14'  MessageID: Server.11  MessageType: Continuation  PayloadString: 7B-22-48-22-3A-22-43-68-61-74-48-75-62-22-2C-22-4D-22-3A-22-62-72-6F-61-64-63-61-73-74-4D-65-73-73-61-67-65-  22-2C-22-41-22-3A-5B-22-74-65-73-74-65-72-22-2C-22-68-65-6C-6C-6F-22-5D-7D  16:22:37:5717 WSSession14.WebSocket'WebSocket #14'  MessageID: Server.12  MessageType: Continuation  PayloadString: 5D-7D  …  Which decodes and reassembles into:  {"C":"d-94DAFC36-B,1|C,0|D,1|E,0","M":[{”broadcastMessage”,”A”:[“tester”,”hello”]}]}  Followed by the message identifier (I) from the client message:  16:22:37:6029 WSSession14.WebSocket'WebSocket #14'  MessageID: Server.14  MessageType: Text  PayloadString: {"I":"0"} |
| 16:23:09:4337 WSSession14.WebSocket'WebSocket #14'  MessageID: Client.19  MessageType: Text  PayloadString: {"H":"chathub","M":"Send","A":["tester","hello again"],"I":1} | 16:23:09:4493 WSSession14.WebSocket'WebSocket #14'  MessageID: Server.20  MessageType: Text  PayloadString: {"C":"d-94DAFC36-B,2|C,0|D,1|E,0","M":[  16:23:09:4650 WSSession14.WebSocket'WebSocket #14'  MessageID: Server.21  MessageType: Continuation  PayloadString: 7B-22-48-22-3A-22-43-68-61-74-48-75-62-22-2C-22-4D-22-3A-22-62-72-6F-61-64-63-61-73-74-4D-65-73-73-61-67-65-22-2C-22-41-22-3A-5B-22-74-65-73-74-65-72-22-2C-22-68-65-6C-6C-6F-20-61-67-61-69-6E-22-5D-7D  16:23:09:4806 WSSession14.WebSocket'WebSocket #14'  MessageID: Server.22  MessageType: Continuation  PayloadString: 5D-7D  …  Which decodes and reassembles into:  {"C":"d-94DAFC36-B,2|C,0|D,1|E,0","M":[{”broadcastMessage”,”A”:[“tester”,”hello again”]}]}  Followed by the message identifier (I) from the client message:  16:23:09:5119 WSSession14.WebSocket'WebSocket #14'  MessageID: Server.24  MessageType: Text  PayloadString: {"I":"1"} |

### Receiving KeepAlive Messages from the Server

The server sends empty text messages to the client to keep the connection alive. No response is required.

16:22:42:6190 WSSession14.WebSocket'WebSocket #14'

MessageID: Server.16

MessageType: Text

PayloadString: {}

16:22:52:6355 WSSession14.WebSocket'WebSocket #14'

MessageID: Server.17

MessageType: Text

PayloadString: {}

# Ping: Ensuring that the server is still responsive

The client MAY send a SignalR ***ping*** request to the server to determine if it’s still responsive to requests. This is only supported by the following transports:

* longPolling
* serverSentEvents
* webSockets

## Client Request

GET /signalr/ping?**connectionData**=%5B%7B%22name%22%3A%22chathub%22%7D%5D

|  |  |  |
| --- | --- | --- |
| Method | | Request URI |
| GET | http://*authority*/*path*/**ping** | |

### URI Parameters

|  |  |
| --- | --- |
| Parameter | Description |
| connectionData | For hub applications, the connectionData is set to the array of hub proxies with active subscriptions and [percent-encoded](http://tools.ietf.org/html/rfc3986#section-2.1):  %5B%7B%22name%22%3A%22chathub%22%7D%5D  which decodes to:  [{“name”:”chathub”}]  in the chat tutorial. |

## Server Response

### Status Code

A successful operation returns status code 200 (OK) with a content type of JSON:

HTTP/1.1 200 OK

…

Content-Type: application/json; charset=UTF-8

### Response Body

A JSON Object is returned in the HTTP response:

{"Response":"pong"}

## Abort: Stopping the SignalR connection/session

After the WebSocket connection is closed as specified in [RFC6455: Close the Websocket Connection](http://tools.ietf.org/html/rfc6455#section-7.1.1):

16:27:55:6348 WSSession14.WebSocket'WebSocket #14'

MessageID: Client.55

MessageType: Close

PayloadString:

16:27:55:6660 WSSession14.WebSocket'WebSocket #14'

MessageID: Server.56

MessageType: Close

PayloadString: 03-E8

a SignalR ***abort*** request is sent to the server.

### Client Request

POST /signalr/abort?**transport**=webSockets&**connectionToken**=e8NhgP5Sb3QhDoDu38Mk%2BhbyKM0wtIJv4PfEDtWHKddL83Zjrsu9PLK5QYSmyAMnWxMMpvGG3h8IlRsncc2soMY1mrQ7%2FyumKH3v0yEyKkcKkOOA7jF4EpLjR4I8vZr9&**connectionData**=%5B%7B%22name%22%3A%22chathub%22%7D%5D

|  |  |  |
| --- | --- | --- |
| Method | | Request URI |
| POST | http://*authority*/*path*/**abort** | |

#### URI Parameters

|  |  |
| --- | --- |
| Parameter | Description |
| connectionData | For hub applications, the connectionData is set to the array of hub proxies with active subscriptions and [percent-encoded](http://tools.ietf.org/html/rfc3986#section-2.1):  %5B%7B%22name%22%3A%22chathub%22%7D%5D  which decodes to:  [{“name”:”chathub”}]  in the chat tutorial. |
| connectionToken | The connectionToken MUST be set to the value of the *ConnectionToken* returned by the server in its response to the ***negotiate*** request. |
| transport | The transport MUST be set to “*webSockets*”. |

## Server Response

### Status Code and Headers

A successful operation returns status code 200 (OK).

### Response Body

The response body is empty.