SignalRServer C++



**Developed by:**

BeKa Software GmbH

Gewerbepark-Wagram 7, 4061 Pasching

Tel: +43 720 / 901 348   
Fax: +43 720 / 901 348-99  
E-Mail: info (at) beka-software.at  
Web: [www.beka-software.at](http://www.beka-software.at/)

**Author:**

Ing.Norbert Kleininger

Inhalt

[Foreword 4](#_Toc402448284)

[Webservers and SignalRServer 4](#_Toc402448285)

[Starting the SignalRServer 4](#_Toc402448286)

[Stopping the SignalRServer 5](#_Toc402448287)

[The Hub-Factory 5](#_Toc402448288)

[The Hub implementation 5](#_Toc402448289)

[The SignalR commands 6](#_Toc402448290)

[The “ping” command 6](#_Toc402448291)

[The “negotiate” command 6](#_Toc402448292)

[The “connect” command 7](#_Toc402448293)

[The “start” command 7](#_Toc402448294)

[The “poll” command (Long Poll only) 7](#_Toc402448295)

[The “send” command 7](#_Toc402448296)

[The “abort” command 8](#_Toc402448297)

[The “reconnect” command 8](#_Toc402448298)

[Query String Components 8](#_Toc402448299)

[JSON Response String 9](#_Toc402448300)

[Authentication 10](#_Toc402448301)

[Server side errors 10](#_Toc402448302)

[PersistentConnection Defaults 10](#_Toc402448303)

[Logging 11](#_Toc402448304)

[Sample “Chat Hub” 11](#_Toc402448305)

[Classes Diagram 12](#_Toc402448306)

[Class members 13](#_Toc402448307)

[SignalRServer 13](#_Toc402448308)

[PersistentConnection 13](#_Toc402448309)

[Subscriber 13](#_Toc402448310)

[PersistentConnectionFactory 13](#_Toc402448311)

[Group 13](#_Toc402448312)

[ClientMessage 14](#_Toc402448313)

[Transport 14](#_Toc402448314)

[Hub 14](#_Toc402448315)

[HubFactory 14](#_Toc402448316)

[HubManager 14](#_Toc402448317)

[UserCredential 14](#_Toc402448318)

[PersistentConnectionInfo 14](#_Toc402448319)

[Request 14](#_Toc402448320)

[Log 14](#_Toc402448321)

[Helper 14](#_Toc402448322)

[SubscriberGarbage 15](#_Toc402448323)

[SignalRHubServer 15](#_Toc402448324)

[HubDispatcher 15](#_Toc402448325)

[HubSubscriber 15](#_Toc402448326)

[SubscriberList 15](#_Toc402448327)

[HubSubscriberList 15](#_Toc402448328)

[HubDispatcherFactory 16](#_Toc402448329)

[HubGroupList 16](#_Toc402448330)

[HubClientMessage 16](#_Toc402448331)

[LongPollingTransport 16](#_Toc402448332)

[SignalR Workflow 16](#_Toc402448333)

# Foreword

SignalRServer is a server developed by Beka-Software in C++ implementing the SignalR protocol.

It was developed in Debian from scratch without using any 3rd party components.

The following document describes the whole functionality of SignalRServer.

Note: Currently only following transports are available in the current version:

|  |  |
| --- | --- |
| Transport | Available? |
| foreverFrame | **no** |
| serverSentEvents | **no** |
| longPolling | yes |
| webSockets | **no** |

# Webservers and SignalRServer

SignalRServer can either be used as a standalone server or it can be included into Apache2 or nginx server for proxy passes.

e.g.

in Apache2 use can simply use the command „ProxyPass“ to route all HTTP requests to SignalRServer.

/etc/apache2/apache2.conf

# Proxy pass for signalR  
<Location /signalr/>  
 ProxyPass <http://127.0.0.1:7788/>  
</Location>

Of course SignalRServer must listen on port 7788 here to accept all HTTP requests of Apache.

# Starting the SignalRServer

Starting SignalRServer can be done in 2 different ways:

1. over UNIX sockets and path
2. over TCP sockets and defining port number

e.g. Starting over UNIX sockets

SignalRHubServer server(new MyHubFactory())  
 hubs.credentials().push\_back(new UserCredential("wiki","pedia"));  
 server.\_options.\_longPollDelay = 20;  
 server.\_options.\_disconnectTimeout = 40;  
 server.\_options.\_keepAliveTimeout = 40;  
 server.\_options.\_connectionIdleTimeout = 300;  
 server.startUnix("/tmp/signalr.socket");

e.g. Starting over TCP sockets

SignalRHubServer server(new MyHubFactory())  
 server.startTcp(7788);

# Stopping the SignalRServer

The server can be stopped easily using the command below:

server.stop();

# The Hub-Factory

In the constructor of SignalRServer you have to mandatorily define an instance of type „HubFactory“.

e.g.

class MyHubFactory : public HubFactory  
 {  
 public:  
 Hub \*createInstance(const char\* hubName) override;  
 };

Hub \*MyHubFactory::createInstance(const char \*hubName)  
 {  
 if (strcmp(hubName,"Chat")==0)  
 return new ChatHub();

return NULL;  
 }

# The Hub implementation

The Hub Factory described above creates a hub that is used to process incoming SignalR-requests and must be implemented the following way:

class ChatHub : public Hub  
 {  
 public:  
 ChatHub();

protected:  
 void onConnected() override;  
 void onReconnected() override;  
 void onDisconnected() override;  
 Variant onMessage(const char\* functionName, vector<Variant>& params) override;  
 };

ChatHub::ChatHub()  
 : Hub(P3\_MACROSTR(ChatHub))  
 {  
 }

void ChatHub::onConnected()  
 {   
 getGroups().add(this,connectionId().c\_str(),"microsoft");  
 getGroups().add(this,connectionId().c\_str(),"beka");  
 }

void ChatHub::onReconnected()  
 {  
 getGroups().add(this,connectionId().c\_str(),"microsoft");  
 getGroups().add(this,connectionId().c\_str(),"beka");  
 }

void ChatHub::onDisconnected()  
 {  
 getGroups().kill(this,connectionId().c\_str(),"microsoft");  
 getGroups().kill(this,connectionId().c\_str(),"beka");  
 }

Variant ChatHub::onMessage(const char\* functionName, vector<Variant>& params)  
 {  
 Variant ret;

if (string(functionName)=="Send")  
 {  
 Log::GetInstance()->Write("Send called.", LOGLEVEL\_DEBUG);  
 }

return ret;  
 }

# The SignalR commands

In this chapter I am listing all commands, that are understood by SignalRServer.

The following commands are listed here: ping, negotiate, connect, start, poll, send, abort, reconnect.

## The “ping” command

|  |  |
| --- | --- |
| Method | GET |
| URI | /signalr/ping |
| Response | "HTTP/1.0 200 OK\r\nContent-Length: 19\r\n\r\n{\"Response\":\"pong\"}" |
| Usage | Checks, if SignalRServer is available and ready to take commands |

## The “negotiate” command

|  |  |
| --- | --- |
| Method | GET |
| URI | /signalr/negotiate?clientProtocol=1.4&connectionData=[%7B%22Name%22:%22Chat%22%7D] |
| Response | "HTTP/1.0 200 OK\r\nContent-Length: 273\r\n\r\n{"ConnectionId":"63bc3f53-06e3-4c01-8ffc-1772829428b4","ConnectionToken":"63bc3f53-06e3-4c01-8ffc-1772829428b4:wiki","DisconnectTimeout":40,"KeepAliveTimeout":40,"LongPollDelay":20,"ProtocolVersion":"1.4","TransportConnectTimeout":5,"TryWebSockets":false,"Url":"\/signalr"}" |
| Usage | Negotiate the server´s capabilities with the connecting client |

## The “connect” command

|  |  |
| --- | --- |
| Method | POST |
| URI | /signalr/connect?transport=longPolling&clientProtocol=1.4&connectionToken=b1d75f3c-3290-410c-9e89-58761a737edb0X0.000000000000CP-1022wiki&connectionData=[%7B%22Name%22:%22Chat%22%7D] |
| Content | - |
| Response | "HTTP/1.0 200 OK\r\nContent-Length: 21\r\n\r\n{"C":"","M":[],"S":1}" |
| Usage | After negotiation, a connection from client to server is initiated. |

## The “start” command

|  |  |
| --- | --- |
| Method | POST |
| URI | /signalr/start?transport=longPolling&clientProtocol=1.4&connectionToken=b1d75f3c-3290-410c-9e89-58761a737edb0X0.000000000000CP-1022wiki&connectionData=[%7B%22Name%22:%22Chat%22%7D] |
| Content | - |
| Response | "HTTP/1.0 200 OK\r\nContent-Length: 22\r\n\r\n{"Response":"started"}" |
| Usage | Signal to start communication |

## The “poll” command (Long Poll only)

|  |  |
| --- | --- |
| Method | POST |
| URI | /signalr/poll?transport=longPolling&clientProtocol=1.4&connectionToken=b1d75f3c-3290-410c-9e89-58761a737edb0X0.000000000001EP-1022wiki&messageId=C,0&connectionData=[%7B%22Name%22:%22Chat%22%7D] |
| Content | - |
| Response | "HTTP/1.0 200 OK\r\nContent-Length: 98\r\n\r\n{"C":"","G":"YjFkNzVmM2MtMzI5MC00MTBjLTllODktNTg3NjFhNzM3ZWRiOlsibWljcm9zb2Z0IiwiYmVrYSJd","M":[{"H":"demo", "M":"myfunc", "A":["88"]}]}" |
| Usage | If long poll transport is desired, initiate a long poll. Connection is kept open for a long time and connection will be closed only on timeout or if messages are available on the server for the subscriber. |

## The “send” command

|  |  |
| --- | --- |
| Method | POST |
| URI | /signalr/send?transport=longPolling&clientProtocol=1.4&connectionData=[%7B%22Name%22:%22Chat%22%7D]&connectionToken=b1d75f3c-3290-410c-9e89-58761a737edb0X0.000000000001EP-1022wiki |
| Content | "data={\"I\":\"0\",\"H\":\"Chat\",\"M\":\"Send\",\"A\":[\"asdfasdf\"]}" |
| Response | "HTTP/1.0 200 OK\r\nContent-Length: 16\r\n\r\n{"I":"0","R":77}" |
| Usage | Call a remote function on the server |

## The “abort” command

|  |  |
| --- | --- |
| Method | POST |
| URI | /signalr/abort?transport=longPolling&clientProtocol=1.4&connectionData=[%7B%22Name%22:%22Chat%22%7D]&connectionToken=2fba1b45-cbc5-42dc-b786-380bc752614b%3Awiki |
| Content | - |
| Response | "HTTP/1.0 200 OK\r\nContent-Length: 15\r\n\r\n{"C":"","M":[]}" |
| Usage | Close an connection that was opened with “connect” |

## The “reconnect” command

|  |  |
| --- | --- |
| Method | POST |
| URI | /signalr/reconnect? transport=longPolling&clientProtocol=1.4&connectionToken=e2026072-224c-4d62-b668-4f16e2c4da53%3Awiki&messageId=S74%2C4&groupsToken=ZTIwMjYwNzItMjI0Yy00ZDYyLWI2NjgtNGYxNmUyYzRkYTUzOlsibWljcm9zb2Z0IiwiYmVrYSJd&connectionData=[%7B%22Name%22:%22Chat%22%7D] |
| Content | - |
| Response | "HTTP/1.0 200 OK\r\nContent-Length: 15\r\n\r\n{"C":"","M":[]}" |
| Usage | Perform a reconnect after lost connection / slow network speed |

# Query String Components

|  |  |
| --- | --- |
| **Query String Attribute** | **Description** |
| messageId | For each client connection, the client’s progress in reading the message stream is tracked using a cursor.(A cursor represents a position in the message stream.) If a client disconnects and then reconnects, it asks the bus for any messages that arrived after the client’s cursor value. The same thing happens when a connection uses long polling.  After a long poll request completes, the client opens a new connection and asks for messages that arrived after the cursor.  The cursor mechanism works even if a client is routed to a different server on reconnect.  The backplane is aware of all the servers, and it doesn’t matter which server a client connects to. |
| connectionToken | A string in the form “GUID”+”:”+”username”  e.g.  e2026072-224c-4d62-b668-4f16e2c4da53:wiki |
| groupsToken | Groups token string sent between client and server. The groups token is a protected and base64 encoded string (e.g. groupsToken=ZTIwMjYwNzItMjI0Yy00ZDYyLWI2NjgtNGYxNmUyYzRkYTUzOlsibWljcm9zb2Z0IiwiYmVrYSJd). In plain and unprotected text it looks like this:  “connectionID” + “:” + JSON-Array  JSON-Array = [“group1”,”group2”,”group3”] |
| connectionData | Connection data is in the following form:  [{“Name”:”Chat”}]  It is an array of hub names requesting.  e.g. if you call “connect” or “abort” or “reconnect” it will always be called on all hubs in the given array. |
| clientProtocol | The client protocol nr (e.g. 1.4) that was resolved during handshake (negotiation) |
| transport | Transport is the name of the protocol used to communicate. One of foreverFrame, serverSentEvents, longPolling, webSockets |

# JSON Response String

The HTTP Response is always a string containing some JSON code.

It may contain the following parts:

|  |  |  |
| --- | --- | --- |
| **JSON key** | **Name** | **What it contains** |
|  |  |  |
| **Persistent Response** |  |  |
| C | Cursor | The MessageID of the message that should be fetched on next poll. |
| M | Messages | An array of messages called on client. For each subscription exactly 1 message will be returned. A subscription is meant as the combination of connectionID and hubName. |
| T | Timeout | Indicates that client must reconnect |
| D | Disconnect | Set when the host is shutting down |
| R | All groups | Contains a list of all groups |
| G | Groups added | Groups that were added on the server. Signed token representing the list of groups. Updates on change |
| g | Groups removed | Groups that were removed on the server |
| S | Init phase | True if the connection is in process of initializing |
| L | Long poll delay | The time the long polling client should wait before reestablishing a connection if no data is received. |
|  |  |  |
| **Hub Message** |  |  |
| H | Hub name | Name of the hub |
| M | Method name | Method that should be called on the hub on the server |
| A | Arguments | Arguments passed to method |
| S | State | JSON containing a session state sent by client to server (see state on hub method return) |
| I | Index | Index for asynchronous calls (see hub method return) |
|  |  |  |
| **Hub Method Return** |  |  |
| I | Index | Message index for asynchronous calls. It must be the same received in the hub message. |
| R | Result | Outcome of function call |
| S | State | JSON containing a session state sent from server to client. It will be re-sent from client to server on next hub message. |
| E | Error | String of error message |
| T | Stack trace | String of stack trace |

# Authentication

SignalRServer C++ currently only supports BASIC authentication.  
Basic authentication requires a special parameter in the HTTP Header.

Authorization: Basic d2lraTpwZWRpYQ==

e.g. Basic Authentication in C#.NET

var hubConnection = new HubConnection(url);

hubConnection.Headers.Add("Authorization", "Basic " + Convert.ToBase64String(System.Text.Encoding.UTF8.GetBytes("wiki:pedia")));

In SignalRServer C++ you can add user credentials that are allowed to connect to server.

SignalRHubServer server(new MyHubFactory());  
 hubs.credentials().push\_back(new UserCredential("wiki","pedia"));

Now only the user “wiki” with password “pedia” is allowed to connection.  
If you do not add any credentials to the server, everybody is permitted.

# Server side errors

The server may send following responses to the client (depending on the called command):

|  |  |
| --- | --- |
| **Response Code** | **Hint** |
| 200 | OK |
| 401 | Unauthorized |
| 408 | Request Timeout |
| 429 | Could not create threads |
| 429 | Too many threads |
| 500 | Internal Server Error |

# PersistentConnection Defaults

On server startup these default values will be used:

|  |  |
| --- | --- |
| DEFAULT\_TRANSPORT\_CONNECTIONTIMEOUT | 5 (sec) |
| DEFAULT\_KEEPALIVE\_TIMEOUT | 30 (sec) |
| DEFAULT\_DISCONNECT\_TIMEOUT | 30 (sec) |
| DEFAULT\_LONGPOLLDELAY | 0 (sec) |
| DEFAULT\_TRYWEBSOCKETS | false |

You can change them, if you set the server options before launching the server.

SignalRHubServer server(new MyHubFactory());

server.\_options.\_longPollDelay = 20;  
 server.\_options.\_disconnectTimeout = 40;  
 server.\_options.\_keepAliveTimeout = 40;  
 server.\_options.\_connectionIdleTimeout = 300;

# Logging

SignalRServer C++ uses an internal logger that is implemented as singleton. All messages (errors, warnings, infos, debug outputs) are logged using the “Log”-class.

Before starting the signalR server you can change the logger options.

The next sample shows how to turn off logging into a file and writing output to screen using an own callback function:

void cbLgCallback(const char\* msg, int , void\* )  
 {  
 printf("%s",msg);  
 printf("\n");  
 }

Log::GetInstance()->SetLogFile("/home/dev/prj/SystemTera/70\_SignalR/signalr.log");  
 Log::GetInstance()->SetEnabled(true); // Turn on logging  
 Log::GetInstance()->SetLogLevel(LOGLEVEL\_INFO); // Log level switched to info  
 Log::GetInstance()->SetCallback(cbLgCallback,NULL); // call a user function for logging  
 Log::GetInstance()->SetUseFileLog(false); // turn off file logging

# Sample “Chat Hub”

The Following sample shows you how to use SignalRServer C++.

#include "ChatHub.h"  
#include <Log.h>  
#include <Hubs/HubSubscriberList.h>  
#include <Helper.h>

ChatHub::ChatHub()  
 : Hub(P3\_MACROSTR(ChatHub))  
{

}

void ChatHub::onConnected()  
{   
 getGroups().add(this,connectionId().c\_str(),"microsoft");  
 getGroups().add(this,connectionId().c\_str(),"beka");  
}

void ChatHub::onReconnected()  
{  
 getGroups().add(this,connectionId().c\_str(),"microsoft");  
 getGroups().add(this,connectionId().c\_str(),"beka");  
}

void ChatHub::onDisconnected()  
{  
 getGroups().kill(this,connectionId().c\_str(),"microsoft");  
 getGroups().kill(this,connectionId().c\_str(),"beka");  
}

Variant ChatHub::onMessage(const char\* functionName, vector<Variant>& params)  
{  
 Variant ret;

if (string(functionName)=="Send")  
 {  
 Log::GetInstance()->Write("Send called.", LOGLEVEL\_DEBUG);  
 ret = Variant::fromValue<int>(send(params[0].toString()));  
 }   
 return ret;  
}

int ChatHub::send(string message)  
{  
 P3\_UNUSED(message);  
 VariantList args;  
 string re = "Hello World";  
 args.push\_back(Variant::fromValue(re));

vector<std::string> groups = { "beka","microsoft" };

getClients().send(this, "Receive", args);  
 getClients().allExcept(connectionId().c\_str()).send(this,"Receive", args);  
 getClients().client(connectionId().c\_str()).send(this,"Receive", args);  
 getClients().groups(groups).send(this,"Receive", args);  
 getClients().othersInGroup(this,"beka").send(this,"Receive", args);

return 77;  
}

# Classes Diagram

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Base class | Inherited classes |  |  | Function |
| SignalRServer |  |  |  | The starting point of all. Our server |
| **|------------------------** | SignalRHubServer |  |  | SignalR Hub server |
|  |  |  |  |  |
| PersistentConnection |  |  |  | Base class for persistent connections |
| **|------------------------** | HubDispatcher |  |  | Receiver and dispatcher of hub messages |
|  |  |  |  |  |
| Subscriber |  |  |  | Base subscriber class holding the connectionid and all messages |
| **|------------------------** | HubSubscriber |  |  | Inherited class containing info about the hub |
|  | SubscriberList |  |  | List of multiple subscribers. Functions to broadcast messages |
|  | **|------------------------** | HubSubscriberList | | Functions like “allExcept”,”group”,”groups”,”othersInGroup”, “clients”,”client” to broadcast messages |
|  |  |  |  |  |
| PersistentConnectionFactory | |  |  | Factory for persistent-connections. Can be passed to ctor of SignalRServer. |
| **|------------------------** | HubDispatcherFactory |  |  | SignalRHubServer uses HubDispatcherFactory to create instances. |
|  |  |  |  |  |
| Group |  |  |  | Group object with connectionId and groupName |
| **|------------------------** | HubGroupList |  |  | List of groups |
|  |  |  |  |  |
| ClientMessage |  |  |  | Message object with method-name and arguments and also the messageId |
| **|------------------------** | HubClientMessage |  |  | Client message with hubName inside |
|  |  |  |  |  |
| Transport |  |  |  | Base transport layer. Holding virtual func. For connect, abort, reconnect. |
| **|------------------------** | LongPollingTransport |  |  | Long polling specific transport layer holding implementations especially for long polling. |
|  |  |  |  |  |
| Hub |  |  |  | The hub class. Contains hubName, a persistentconnection and the http-request. |
| HubFactory |  |  |  | HubFactory instance is passed on server ctor. |
| HubManager |  |  |  | HubManager singleton with global “subscribers” and “groups” list. |
| UserCredential |  |  |  | UserCredential with username and password for BASIC auth. |
| PersistentConnectionInfo |  |  |  | For each real connection with a unique ID an info object will be created. While a full communication runs through multiple persistent connection instances, there will only be 1 info object. It is created at “connect” and will be destroyed on “abort”. An info instance can even point to 2 persistent connections at one time: 1 long polling connection and 1 hub message connection. |
| Request |  |  |  | The HTTP request object holding querystring, body, http-version, method, uri, user, pwd |
| Log |  |  |  | Logger singleton for logging |
| Helper |  |  |  | Helper function for string operations.. |
| SubscriberGarbage |  |  |  | Garbage collector for old subscriptions |

# Class members

## SignalRServer

**SignalRServer**()  
**SignalRServer**(PersistentConnectionFactory\* factory, int maxThreads=10)  
void **startTcp**(int port)  
void **startUnix**(const char \*sock)  
bool **stop**(int timeout\_ms=1000)  
bool **isRunning**()

## PersistentConnection

**PersistentConnection**()  
virtual bool **authorizeRequest**(Request\* requ)  
virtual void **onConnected**(Request \*request, const char\* connectionId)  
virtual void **onReconnected**(Request \*request, const char\* connectionId)  
virtual string **onReceived**(Request \*request, const char\* connectionId, const char\* data)  
virtual void **onDisconnected**(Request \*request, const char\* connectionId)

## Subscriber

const string &**connectionId**() const  
list<ClientMessage \*> &**clientMessages**()

## PersistentConnectionFactory

virtual PersistentConnection\* **createInstance**()

## Group

string **connectionId**()  
void **setConnectionId**(const char\* connectionId)   
string **groupName**()  
void **setGroupName**(const char\* groupName)  
string **removePrefix**()

## ClientMessage

const string &**clientMethod**() const  
const VariantList &**arguments**() const  
int **messageId**()  
void **setMessageId**(int id)

## Transport

virtual void **processAbortRequest**(PersistentConnection\* conn, Request\* request)  
virtual void **processConnectRequest**(PersistentConnection\* conn, Request\* request)  
virtual void **processReconnectRequest**(PersistentConnection\* conn, Request\* request)

## Hub

HubSubscriberList **getClients**()  
HubGroupList& **getGroups**()  
const string &**hubName**() const  
string **connectionId**()

## HubFactory

virtual Hub\* **createInstance**(const char\* hubName)

## HubManager

HubSubscriberList& **getSubscribers**()  
HubGroupList& **getGroups**()

## UserCredential

void **setUsername**(const char \*username)   
void **setPassword**(const char \*password)   
string **username**()  
string **password**()

## PersistentConnectionInfo

string& **connectionId**()  
time\_t &**start**()  
time\_t **timeout**()  
bool **exceeded**()  
list<PersistentConnection\*>& **getConnections**()

## Request

string **getParameter**(const char\* name)  
string **queryString**() const  
string **body**() const  
string **version**() const  
string **method**() const  
string **uri**() const  
string **user**() const  
string **password**() const

## Log

void **Write**(const char\* str,int level=LOGLEVEL\_INFO)  
void **SetLogFile**(const char\* path)  
void **SetEnabled**(bool enabled=true)  
void **SetLogLevel**(int level=LOGLEVEL\_INFO)  
void **SetCallback**(LogCallback cb, void\* data=NULL)  
void **SetUseFileLog**(bool fl=true)

## Helper

static string **tail**(string const& source, size\_t const length)  
static bool **endWith**(string const& source, string const& checkval)  
static bool **replace**(string& str, const string& from, const string& to)  
static string **createGUID**()  
static string **extractConnectionIdFromToken**(const char\* connectionToken)  
static string **getQueryStringParam**(const char\* param, const char\* query)  
static string **getTimeStr**()  
static string **decode**(const char\* str)  
static string **getHttpParam**(const char\* param, const char\* req)  
static string **getStrByIndex**(int i,const char\* req)  
static string **getLine**(const char\* req)  
static string **getLeftOfSeparator**(const char\* str, const char\* sep)  
static string **getRightOfSeparator**(const char\* str, const char\* sep)  
static string **base64\_encode**(unsigned char const\* , unsigned int len)  
static string **base64\_decode**(string const& s)  
static string **getBasicUser**(const char\* auth)  
static string **getBasicPassword**(const char\* auth)  
static int **generateMessageId**()  
static string **NullToEmpty**(const char\* str)  
static string **GetNextMessageId**(const char\* messageId)  
static string **IntToStr**(int a)  
static list<string> **split**(const char\* str, const char\* sep)

## SubscriberGarbage

static SubscriberGarbage& **getInstance**()  
void **add**(Subscriber\* ptr)  
void **collect**()  
list<sSubscriberGarbage>& **garbage**()

## SignalRHubServer

Hub\* **createHub**(const char\* hubName, PersistentConnection\* conn, Request\* r)

## HubDispatcher

virtual void **onConnected**(Request \*request, const char\* connectionId) override  
virtual void **onReconnected**(Request \*request, const char\* connectionId) override  
virtual string **onReceived**(Request \*request, const char\* connectionId, const char\* data) override  
virtual void **onDisconnected**(Request \*request, const char\* connectionId) override

## HubSubscriber

const string &**hubName**() const  
const list<HubClientMessage \*> &**clientMessages**() const

## SubscriberList

virtual void **send**(const char\* func, VariantList& args)  
bool **hasMessages**(const char\* connectionId)  
list<ClientMessage \*> **getMessages**(const char\* connectionId)  
void **removeAllMessages**(const char\* connectionId)  
list<Subscriber\*> **getSubscriptions**(const char\* connectionId)

## HubSubscriberList

HubSubscriberList **allExcept**(const char\* connectionId)  
HubSubscriberList **group**(const char\* group)  
HubSubscriberList **groups**(std::vector<std::string>& groups)  
HubSubscriberList **othersInGroup**(Hub\* hub,const char\* g)  
HubSubscriberList **othersInGroups**(Hub\* hub,std::vector<std::string>& groups)  
HubSubscriberList **clients**(std::vector<std::string>& connectionIds)  
HubSubscriberList **client**(const char\* connectionId)  
HubSubscriberList **byHub**(const char\* hubName)  
bool **contains**(Subscriber \*s)  
void **send**(const char \*hub, const char \*func, VariantList &args)  
void **send**(Hub \*h, const char\* func, VariantList& args)  
void **subscribe**(const char\* hubName, const char\* connectionId)  
void **unsubscribe**(const char\* connectionId)  
bool **exists**(const char\* hubName, const char\* connectionId)

## HubDispatcherFactory

virtual PersistentConnection\* **createInstance**()

## HubGroupList

bool **exists**(Hub\* hub,const char\* connectionId, const char\* groupName)  
void **add**(Hub\* hub,const char\* connectionId, const char\* groupName)  
void **kill**(Hub\* hub,const char\* connectionId, const char\* groupName)  
list<std::string> **getForClient**(const char\* connectionId)  
Group\* **getAnyGroup**(const char\* connectionId)  
void **killAll**(const char\* connectionId)

## HubClientMessage

string **hubName**() const

## LongPollingTransport

void **processAbortRequest**(PersistentConnection\* conn, Request\* request) override  
void **processConnectRequest**(PersistentConnection\* conn, Request\* request) override

# 

# SignalR Workflow



















