Simple SignalR Server and Client Applications Demonstrating Common Usage Scenarios

Mustafa Kok

16 Jul 2019 CPOL

SignalR server and client applications for demonstrating common scenarios like connecting, disconnecting, joining and leaving groups, sending messages to all clients, to specific or group of clients

Source code for the sample applications (GitHub)

Introduction

SignalR is a library that simplifies the real-time communication between the web page and the server. It uses WebSockets or long-polling method depending on the web browser used.

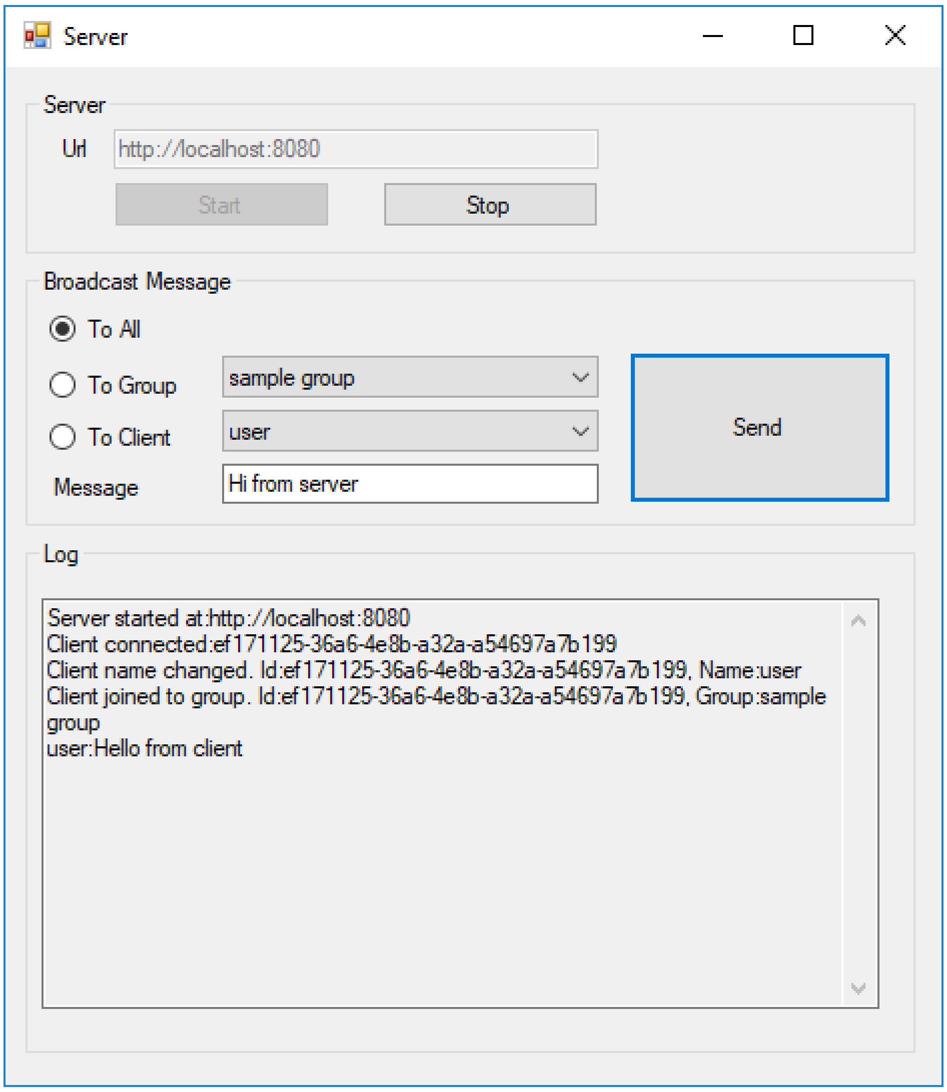
There are too many articles about hosting a SignalR hub and communicating with it from a client applications. But some of them are too simple (just show how to send a message), some of them are too complex that are hard to understand and some of them have badly organized code. I wanted to demonstrate simple scenarios with simple code samples with almost no extra code.

Demonstrated scenarios are:

- Self hosting a SignalR server with a sample hub on a WindowsForms application
- Connecting to SignalR hub from WindowsForms and JavaScript clients
- Subscribing to hub events (callback methods) from client applications
- Calling methods defined on a hub, from client applications
- Joining and leaving groups from client applications
- Broadcasting messages to all clients, to group of clients or to a single client from a SignalR hub

WinForms Server Application

WindowsForms self hosted SignalR application looks like this.



```
Server started at:http://localhost:8080
Client connected:ef171125-36a6-4e8b-a32a-a54697a7b199
Client name changed. Id:ef171125-36a6-4e8b-a32a-a54697a7b199, Name:user
Client joined to group. Id:ef171125-36a6-4e8b-a32a-a54697a7b199, Group:sample
group
user:Hello from client
```

To host a SignalR server on a WindowsForms application, the following nuget packages must be installed either from Package Manager Console or from Manage Nuget Packages window.

- `Microsoft.Owin.SelfHost`
- `Microsoft.Owin.Cors`
- `Microsoft.AspNet.SignalR.Core`
- `Microsoft.AspNet.SignalR.SelfHost`

The next step is preparing the self-hosting environment. To do this, create a class named `Startup` and add the following code into it. The `Configuration` method will be called automatically by the self hosting environment.

```
class Startup
{
     public void Configuration(IAppBuilder app)
     {
          //CORS need to be enabled for calling SignalR service
          //from external web applications/pages
          app.UseCors(CorsOptions.AllowAll);
          //Find and register SignalR hubs
          app.MapSignalR();
     }
}
```

To start Web Application which hosts the SignalR service, call `WebApp.Start` method with the desired SignalR service address.

```
_signalR = WebApp.Start<Startup>(txtUrl.Text);
```

The next step is preparing the SignalR hub class which will handle the communication with the clients. Every SignalR hub class derives from `Microsoft.AspNet.SignalR.Hub` class. It includes methods that can be called by the clients. It can also raise events (call callback methods on clients) that clients can listen to.

```
public class SimpleHub : Hub
{
//Called when a client is connected
override OnConnected()

//Called when a client is disconnected
override OnDisconnected(bool stopCalled)

//Public methods inside this region are callable from any SignalR client
#region Client Methods
//A client provides a user name for himself
void SetUserName(string userName)

//Client calls this to join a group
Task JoinGroup(string groupName)

//Client calls this to leave a group
Task LeaveGroup(string groupName)

//Clients call this to send a message to the hub
//Hub then broadcasts the message to all the connected clients
void Send(string msg)

#endregion
}
```

`Context` property of a hub whose type is `HubCallerContext` has several properties that provide information about current request like `ConnectionId`, `Headers` and `RequestCookies`.

`Clients` property can be used to perform operations on all the connected clients or on specific clients. Every client has a `ConnectionId` which can be used for performing operations on a client. The following calls return dynamic objects which can be used to invoke operations on the client side.

```
//For all the clients
Clients.All

//For a specific client
Clients.Client(connectionId)

//For clients that are members of a group
Clients.Group(groupName)
```

The following code block invokes `addMessage` operation on all the clients. If no such operation is defined on the client side, nothing happens.

```
Clients.All.addMessage(senderUserName, message);
```

`Groups property` can be used to add or remove clients to groups.

```
//To add a user to a group
Groups.Add(userConectionId, groupName);

//To remove a user from a group
Groups.Remove(userConectionId, groupName);
```

A new instance of the hub class is created for each request, so hub classes cannot contain any state. Also, it is not possible to add instance events to a hub class and track the operations by subscribing to these events.

To overcome this problem, two methods can be used:

1. Adding static state variables and events
2. Injecting a state object to each created hub instance, using dependency injection

In this sample project, I have used static variables and events approach for simplicity. So the `SimpleHub` class contains a static dictionary for holding state data for connected clients and several static events to inform its subscribers.

```
static ConcurrentDictionary<string, string> _users = new ConcurrentDictionary<string, string>
();

public static event ClientConnectionEventHandler ClientConnected;

public static event ClientConnectionEventHandler ClientDisconnected;

//...
```

In order to send a message from server to clients (or invoke an operation on a client), we need to access the `Clients` property of the hub, but it is normally only available when processing a request in a `hub` method. There is, of course, another way to access hub related context outside a hub class. `Microsoft.AspNet.SignalR.GlobalHost` has several static properties that can be used to access and manipulate SignalR host. One of them is `ConnectionManager` property.

The following code gets the hub context for `SimpleHub` type and invokes a method on all the connected clients.

```
var hubContext = GlobalHost.ConnectionManager.GetHubContext<SimpleHub>();

hubContext.Clients.All.addMessage("SERVER", txtMessage.Text);
```

WinForms Client Application

WindowsForms client application looks like this:

For SignalR client components, add `Microsoft.AspNet.SignalR.Client` nuget package to the project.

The first step of connecting to a SignalR hub is creating a `HubConnection` to the SignalR server. Then, getting the proxy object that will be used to communicate with the desired hub.

```
//Create a connection for the SignalR server
_signalRConnection = new HubConnection(txtUrl.Text);
```

```
//Get a proxy object that will be used to interact with the specific hub on the server
//There may be many hubs hosted on the server, so provide the type name for the hub
_hubProxy = _signalRConnection.CreateHubProxy("SimpleHub");
```

The next step is to register hub events (methods invoked by the hub). The following code registers a handler method for `AddMessage` event. Registered lambda expression takes two parameters and writes them to `log` area.

```
_hubProxy.On<string, string>("AddMessage",
                    (name, message) => writeToLog($"{name}:{message}"));
```

After configuring the hub events, it is time to connect to SignalR hub by calling the `Start` method of the `HubConnection` object.

```
await _signalRConnection.Start();
```

Public operations defined on the connected hub can be invoked using the `IHubProxy` objects `Invoke` method with the remote operation's name and required parameters. Sample code invokes the `Send` method of `SimpleHub` class.

```
_hubProxy.Invoke("Send", txtMessage.Text);
```

JavaScript Client Application

JavaScript client application looks like the WinForms client application. I tried to construct a similar interface and implement exactly the same operations. The only functional difference is, user interface elements are enabled and disabled based on the current state.

JavaScript client uses `jQuery` and `jQuery.signalR` script libraries. You can add them to your project by installing the following nuget packages:

- jQuery
- Microsoft.AspNet.SignalR.JS

In order to connect and use remote SignalR hub, another script library must be referenced. But it is not a file that you can locate and add as a script reference in HTML file. This script is automatically generated by the SignalR server. Script address is **"signalR address" + "/hubs"**

```
<script src="http://localhost:8080/signalr/hubs"></script>
<!-- or relative path -->
<script src="~/signalr/hubs"></script>
```

The test client application can connect any signalR service by entering the URL address of the service into the Url inputbox. So the script location is not to be fixed for this sample application. In order to solve dynamic script location problem, I have loaded this automatically created hubs script inside the `connect` function using the `jQuery.getScript` function. `getScript` function takes two parameters; address of the JavaScript to be loaded and a function that will be called when downloading the script has finished. As you can see in `connect` JavaScript function in the sample *Index.html* page, actual connection operations are performed after this auto generated has downloaded successfully.

```
//Connect to the SignalR server and get the proxy for simpleHub
function connect() {

     //Load auto generated hub script dynamically and perform connection
     //operation when loading completed
     //SignalR server location is specified by 'Url' input element,
     //hub script must be loaded from the same location
     //For production, remove this call and uncomment the script block in the header part
      $.getScript( $("#txtUrl").val() + "/hubs", function() {

                $.connection.hub.url = $("#txtUrl").val();

                // Declare a proxy to reference the hub.
                simpleHubProxy = $.connection.simpleHub;

                //Register to the "AddMessage" callback method of the hub
                //This method is invoked by the hub
                simpleHubProxy.client.addMessage = function (name, message) {
                    writeToLog(name + ":" + message);
                };

                //Connect to hub
                $.connection.hub.start().done( function () {
                    writeToLog("Connected.");
                    simpleHubProxy.server.setUserName($('#txtUserName').val());
                });
        });
}
```

Steps for connecting and interacting with a SignalR hub are the same as WindowsForms client. Sample code demonstrates how to prepare a connection to a SignalR hub, create a proxy for interacting with it, registering a server event (or callback) and calling (invoking) a method (`setUserName` in this case) on the server side.

There is a simple JavaScript function for each operation. For example, following is the `sendMessage` function that send the provided message to the server by invoking the "`send`" method with the text entered into `txtMessage` input box.

```
//Send a message to server
function sendMessage() {
    if(simpleHubProxy != null) {
        simpleHubProxy.server.send($('#txtMessage').val());
    }
}
```

History

- 16th July, 2019: Initial version

License

About the Author

Mustafa Kok

Software Developer (Senior) Freelance
Sweden

Experienced senior C# developer, sometimes codes in Java and C++ also. He designs and codes desktop applications (WinForms, WPF), windows services and Web APIs (WCF, ASP.Net MVC). He is currently improving his ASP.Net MVC and JavaScript capabilities. He acts as an architect, coder and trainer.
He is mostly experienced in Media Asset Management (MAM) applications, broadcasting sector and medical applications.

LinkedIn: www.linkedin.com/in/mustafa-kok-75352944/
GitHub: github.com/nthdeveloper
StackOverflow: stackoverflow.com/users/1844220/nthdeveloper

Comments and Discussions

6 messages have been posted for this article Visit **https://www.codeproject.com/Articles/5162436/Simple-SignalR-Server-and-Client-Applications-Demo** to post and view comments on this article, or click **here** to get a print view with messages.