SignalR on .NET 6 The Complete Guide

The easiest way to enable real time two-way HTTP communication on .NET 6

Fiodar Sazanavets

SignalR on .NET 6 - the Complete Guide

The easiest way to enable real-time two-way HTTP communication on .NET 6

Fiodar Sazanavets

This book is for sale at http://leanpub.com/signalronnet6-thecompleteguide

This version was published on 2022-10-24

Tweet This Book!

Please help Fiodar Sazanavets by spreading the word about this book on Twitter!

The suggested tweet for this book is:

Here is the complete guide on using SignalR on .NET 6!

The suggested hashtag for this book is #signalr.

Find out what other people are saying about the book by clicking on this link to search for this hashtag on Twitter:

#signalr

Contents

1 - Introduction to SignalR

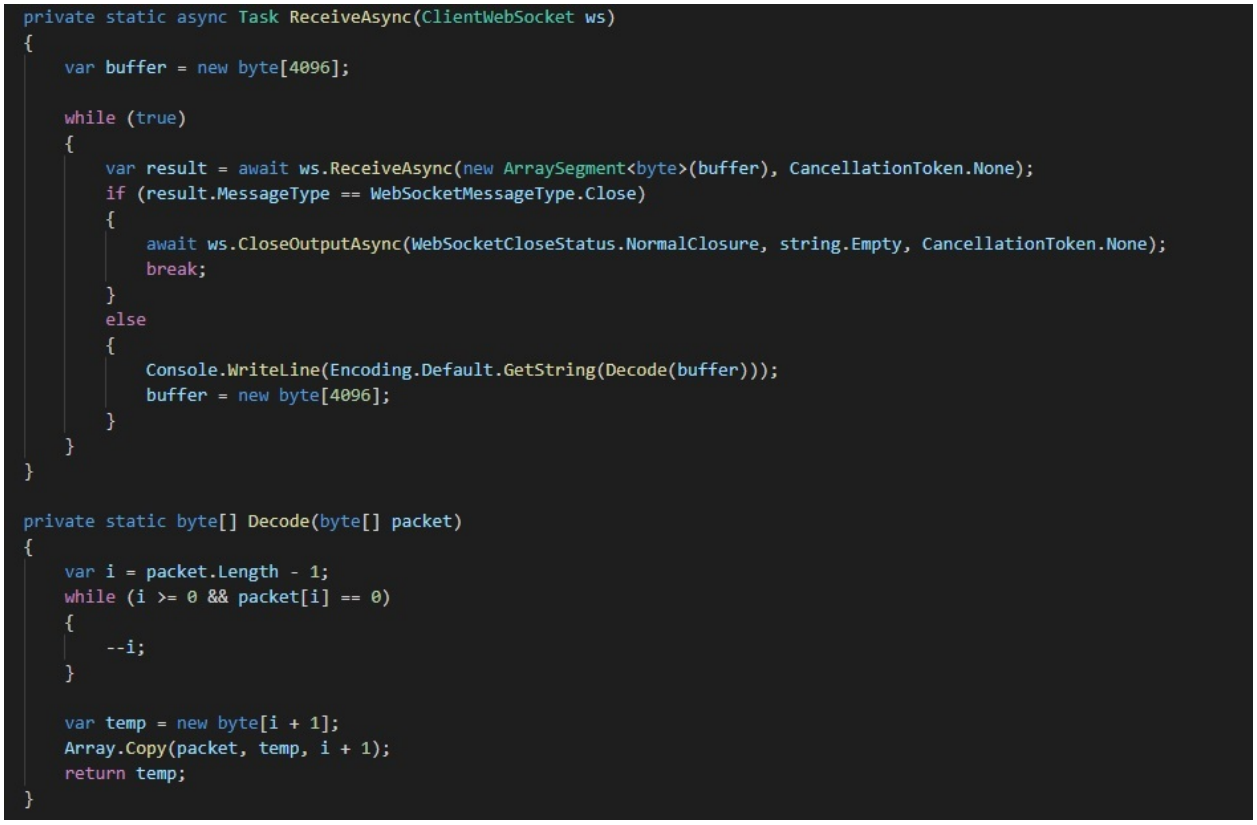
If you are a software developer who builds web, mobile or internet of things (IoT) applications, you will appreciate how important it is for your applications to have the ability to communicate with the server asynchronously and in real time. Also, you probably realize that the standard request-response model of communication isn't fully suitable for modern apps. Sometimes, your client application (whether it's a web page, a mobile app or a service on an IoT device) needs to be able to receive a real-time update from the server that was actually triggered by an event on the server and not by a client request.

We see this functionality everywhere. When you use a messenger app, you would expect to receive a message as soon as someone has sent you one. When you control your IoT devices, you expect them to respond to your commands as soon as you trigger them.

But the problem with such functionality is that the said functionality is not always easy to implement. You could still use the classic request-response model and just carry on sending requests to the server until you receive a right type of response. But this would be wasteful. And, if your client application has limited bandwidth to work with, you may not even be able to do it at all, as continuous requests to the server may end up using up the entire bandwidth really quickly. As well as all of this, your code to implement such a behavior would probably be way more complicated than it should be.

There is also an alternative - `WebSocket` protocol. This is a much better solution. The protocol was specifically designed to enable two-way communication between the client and the server. All you have to do is establish a connection between the two endpoints. And, as long as this connection is live, the messages can flow both ways. As well as sending any arbitrary message from the client to the server, you can also send messages from the server to the client. And, on top of this, WebSocket protocol is very efficient. Maintaining the connection doesn't use much bandwidth at all.

However, WebSocket doesn't come without its own problems. It's far from being the easiest thing to work with. Out of the box, WebSocket protocol uses raw data instead of human-readable abstractions. It transmits messages by using raw bytes. So, it's up to you to do all the assembling and disassembling of messages. It's also up to you to monitor the state of the connection. The WebSocket code you'll have to write will probably look similar to this, which is neither very intuitive nor easily readable:



```
private static async Task ReceiveAsync(ClientWebSocket ws)
{
    var buffer = new byte[4096];

    while (true)
    {
        var result = await ws.ReceiveAsync(new ArraySegment<byte>(buffer), CancellationToken.None);
        if (result.MessageType == WebSocketMessageType.Close)
        {
            await ws.CloseOutputAsync(WebSocketCloseStatus.NormalClosure, string.Empty, CancellationToken.None);
            break;
        }
        else
        {
            Console.WriteLine(Encoding.Default.GetString(Decode(buffer)));
            buffer = new byte[4096];
        }
    }
}

private static byte[] Decode(byte[] packet)
{
    var i = packet.Length - 1;
    while (i >= 0 && packet[i] == 0)
    {
        --i;
    }

    var temp = new byte[i + 1];
    Array.Copy(packet, temp, i + 1);
    return temp;
}
```

Figure 1.1 - WebSocket implementation example

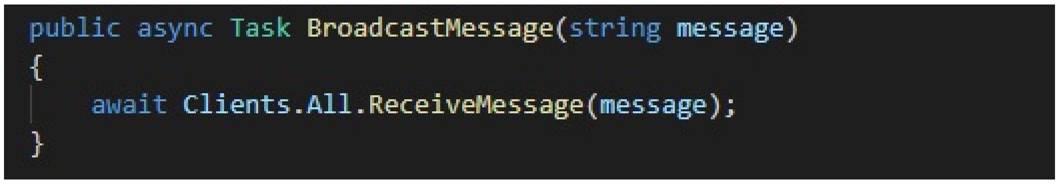
But if you are .NET developer, you won't have to deal with any of this. Enabling efficient real-time two-way communication will almost be as easy as making classes inside a single application call each other's methods. And all of this was made possible with a help of a library called `SignalR`, which is one of the in-built libraries of AS.NET Core.

What makes SignalR so great

SignalR is a library that is incredibly easy to implement compared to the alternatives. Using this library is just as easy as writing remote procedure calls. You will have some endpoint methods on your server that are just bog-standard C# methods. And, as long as you specify the methods of the same names on the client and put the expected parameters into them, the connection will be made and the method will be triggered.

Same applies the other way round. Your client will have event listeners with arbitrary names. And, as long as you spell the name of the listener correctly in your server-side code and apply expected data types as parameters, the listener on the client will be triggered.

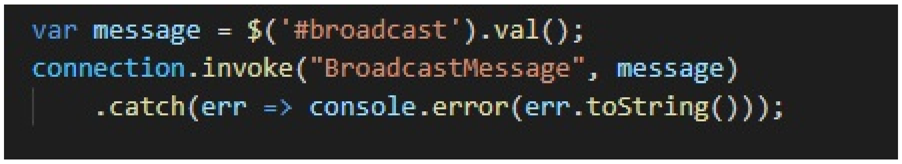
For example, your server-side method may look like this:



```
public async Task BroadcastMessage(string message)
{
    await Clients.All.ReceiveMessage(message);
}
```

Figure 1.2 - Server-side SignalR method

And your client code that will trigger this method may look like this:



```
var message = $('#broadcast').val();
connection.invoke("BroadcastMessage", message)
    .catch(err => console.error(err.toString()));
```

Figure 1.3 - Client-side SignalR method invocation

Under the hood, SignalR library uses WebSocket protocol. But you, as a developer, don't have to worry about implementation details of it. Those are abstracted away to make things as convenient for you as possible.

But WebSocket is not the only protocol that SignalR uses, even though it is the default protocol it will try to use. It just happens that, although there aren't many use cases where you won't be able to use WebSocket, occasionally you may encounter such a situation. And this is why SignalR comes with two fallback protocols, which are server-sent events and long polling. The fallback order is as follows:

1. If possible, use WebSocket
2. If WebSocket can't be used, use server-sent events
3. If server-sent events can't be used, use long polling

Long polling is the least efficient protocol of them all. This is where the client sends a standard HTTP request to the server and just waits until the response is sent back. This is why you won't be using this protocol until absolutely necessary. But the main benefit of it is that absolutely any system that supports HTTP supports long polling.

If you need to, you can even explicitly specify the protocol in the client configuration. But in most cases, you won't have to. But even if you do, the choice of the protocol will have zero impact on the way you write your code. All your code will be identical regardless of the protocol being used.

Example use cases for SignalR

SignalR is a perfect library to be used in any scenarios where clients need to receive real-time updates from the server or there is high a high frequency of data exchange between the client and the server. As per the official documentation[1], the following are some of the examples where SignalR is an ideal library to use:

[1]https://docs.microsoft.com/en-us/azure/azure-signalr/signalr-overview

- **High frequency data updates**: gaming, voting, polling, auction.
- **Dashboards and monitoring**: company dashboard, financial market data, instant sales update, multi-player game leader board, and IoT monitoring.
- **Chat**: live chat room, chat bot, on-line customer support, real-time shopping assistant, messenger, in-game chat, and so on.
- **Real-time location on map**: logistic tracking, delivery status tracking, transportation status updates, GPS apps.
- **Real time targeted ads**: personalized real time push ads and offers, interactive ads.
- **Collaborative apps**: coauthoring, whiteboard apps and team meeting software.
- **Push notifications**: social network, email, game, travel alert.
- **Real-time broadcasting**: live audio/video broadcasting, live captioning, translating, events/news broadcasting.
- **IoT and connected devices**: real-time IoT metrics, remote control, real-time status, and location tracking.
- **Automation**: real-time trigger from upstream events.

Who is this book for

This book will be useful to any ASP.NET Core developer who is interested in enabling real-time two-way communication between the clients and the server. Whether you are building a real-time chat application, a messenger app, or an IoT control hub, you will find the information in this book useful.

The scope of this book

This book will teach you everything you will need to know about SignalR, so you will be able to use it in any type of project where it can provide benefits. We won't go too deep into the inner workings of the library, but we will cover enough so you know how to use it in the most optimal way. We will even cover some use cases that aren't officially documented, but that may actually happen. For example, you will learn how to connect a raw WebSocket client to the SignalR server hub. This is a use case I have personally dealt with in one of my projects.

Another topic that we will address is how to integrate the latest .NET 6 and C# 10 features with SignalR. There is a wealth of information online on how to use SignalR, but since .NET 6 is relatively recent, there isn't much information on how to take advantage of the latest .NET 6 features while using it. And this book is aiming to address this gap.

In this book, you will be working on the same .NET solution throughout the chapters, adding new features to it as you go along. All the code samples used in the book are available in this GitHub repo:

https://github.com/fiodarsazanavets/SignalR-on-.NET-6---the-complete-guide

Prerequisites

This book assumes that you are already somewhat familiar with .NET in general and ASP.NET Core in particular. Although anyone will be able to follow the examples provided in the chapters, to fully understand them, you need to understand what ASP.NET Core applications are and how they are structured.

One of the best places to learn ASP.NET Core is its official documentation website[2]. In our examples, we will mostly be using MVC (model-view-controller) template. So, to take the maximum benefit from this book, you will need to know what it is and how it works.

How to use this book

If you are completely new to SignalR, then my recommendation would be to follow this book from the beginning. However, you don’t necessary have to read every chapter.

This book is intended to be both a complete tutorial and a reference book. So you can just read any specific chapter related to a specific SignalR feature that you are interested in.

Even though the book is structured in such a way that we add features to the same application throughout the chapters, you don’t have to go through all previous chapters if you are interested only in a specific chapter. You can download the complete GitHub repository and just open the code sample folder that was relevant to the final part of the previous chapter.

Book structure

The book will consist of the following chapters

1 - Introduction to SignalR

This chapter provides a high-level overview of SignalR and outlines the structure for the remainder of the book.

2 - Setting up your project

This chapter will show you how to set up your environment, create an ASP.NET Core application and add a SignalR hub to it. We will cover how server-side components of SignalR work.

The chapter consists of the following topics:

- Setting up your environment
- Setting up SignalR hub
- Making SignalR hub strongly-typed

[2]https://docs.microsoft.com/en-us/aspnet/core/

3 - In-browser SignalR clients

In this chapter, you will learn how to set up in-browser SignalR clients. This is perhaps the most commonly used type of SignalR clients. For example, this is how you can build a real-time chat application.

The chapter consists of the following topics:

- Setting up JavaScript client
- Setting up Blazor WebAssembly client

4 - External SignalR clients

In this chapter, you will learn how to set up external self-contained applications as SignalR clients. For example, such a client may be an IoT application. We will cover all remaining types of clients that are officially supported and documented. But, on top of this, you will learn how to connect a raw WebSocket to the SignalR server, so you can write a client for it in absolutely any language of your choice.

The chapter consists of the following topics:

- Setting up .NET client
- Setting up Java client
- Setting up a raw WebSocket client

5 - Sending messages to individual clients or groups of clients

This chapter will walk you through the process of grouping SignalR clients. We will also have a look at how clients can be called selectively by the server. This is useful, because you won't always have to notify all connected clients when some event occurs.

The chapter consists of the following topics:

- Broadcasting messages to all clients
- Sending messages to specific clients
- Working with client groups

6 - Streaming in SignalR

As well as having an ability to make single calls, SignalR has an ability to stream data from the server to the client and vice versa. In this chapter, you will learn how to enable it.

The chapter consists of the following topics:

- What is streaming used for
- Client streaming in SignalR
- Server streaming in SignalR

7 - Advanced SignalR configuration

In this chapter, you will learn how to apply advanced configuration to SignalR, both client- and server-side. We will also cover an additional messaging protocol that can be used with SignalR - MessagePack. This protocol will allow you to substantially increase the performance of your communication.

The chapter consists of the following topics:

- Configuring SignalR server
- Configuring SignalR client
- Pros and cons of MessagePack protocol

8 - Securing your SignalR applications

Security is an important part of any software applications. You don't want unauthorized users to gain access to sensitive information or functionality. And SignalR is not exception. This chapter will teach you how to secure your SignalR application.

The chapter consists of the following topics:

- What is CORS and why it's important
- Setting up single sign-on provider
- Applying authentication in SignalR
- Applying authorization in SignalR

9 - Scaling out SignalR application

If your application is intended to work with a large number of clients, you will eventually need to scale it out. And this chapter will teach you how to scale out your SignalR hub.

The chapter consists of the following topics:

- Setting up Redis cache
- Running multiple hub instances via Redis backplane
- Using HubContext to send messages from outside SignalR hub

10 - Introducing Azure SignalR Service

You can scale out a SignalR server on premises, but you can also outsource the task to Azure cloud. This chapter will show you how to scale out your SignalR server by using Azure SignalR Service

The chapter consists of the following topics:

- Setting up Azure SignalR Service
- Adding Azure SignalR Service dependencies to your application
- Overview of Azure SignalR Service REST API

About the author

Fiodar Sazanavets is an experienced lead software engineer whose main area of expertise is Microsoft stack, which includes ASP.NET (Framework and Core), SQL Server, Azure, and various front-end technologies. Fiodar is familiar with industry-wide best practices, such as SOLID principles, software design patterns, automation testing principles (BDD and TDD) and microservices architecture.

Fiodar has built his software engineering experience while working in a variety of industries, including water engineering, financial, retail, railway and defence. He has played a leading role in various projects and, as well as writing software, he gained substantial experience in architecture and design.

Fiodar is an author of a number of technical books and online courses. He regularly writes about software development on his personal website, https://scientificprogrammer.net. He is also available to book for personal mentoring sessions via https://mentorcruise.com/mentor/fiodarsazanavets.

Getting in touch with the author

If you want to get in touch with me regarding the content of the book, you can contact me via either Twitter or LinkedIn. Perhaps, there is additional content that you want to have included in the next edition of the book. Or maybe you found some errors in the current edition. Any feedback is welcome. And this is how you can get in touch:

Twitter: https://twitter.com/FSazanavets

LinkedIn: https://www.linkedin.com/in/fiodar-sazanavets/

2 - Setting up your project

This chapter will show you how to set up web application project and add server-side SignalR dependencies to it. We will cover the most fundamental components of SignalR and explain how the library works.

The chapter covers the following topics:

- Setting up your environment
- Setting up SignalR hub
- Making SignalR hub strongly-typed

By the end of the chapter, you will have learned how to enable SignalR in an ASP.NET Core application. We will focus on an MVC application. However, the principles taught in this chapter will be universally applicable to other types of ASP.NET Core applications, including Razor Pages and Web API.

Prerequisites

To take the most out of this chapter, you will need to already be somewhat familiar with the structure of ASP.NET Core applications. There are no other prerequisites, as full instruction on how to set up your environment will be provided.

Another prerequisite is that you need a computer with either Windows, Mac OS, or Linux operating system. You will be working with .NET 6, which was designed to be compatible with any of these platforms.

The complete code samples from this chapter are available from the following location in the GitHub, which has separate folders corresponding to individual parts of the chapter:

https://github.com/fiodarsazanavets/SignalR-on-.NET-6---the-complete-guide/tree/main/Chapter-02

Setting up your environment

The two main things that you will need in order to be able to follow the examples from this book are software development kit (SDK) for .NET 6 and a code editor that is compatible with C#.

If you already have these, you can skip this section. Otherwise, this is how you set everything up.

Setting up .NET 6 SDK

.NET 6 SDK contains both the platform that you will run your applications on and the collection of tools that will allow you to develop the applications. The best place to download the SDK from is its official website[3].

.NET 6 SDK will give you access to all the tools that you will need to instantiate your projects and build your applications. One of the powerful tools it comes with is `dotnet` command line interface (CLI) tool. It allows you to instantiate, build and run .NET applications via any command line terminal on any supported operating system. And all of the commands will be the same regardless of which operating system are you using.

Of course, you don't necessarily have to use CLI to build and run your application. You can do all of this via graphical user interface (GUI) of the integrated development environment (IDE) of your choice. However, because different operating systems have different IDEs available and they look completely different, all instructions provided in the book will be based on CLI commands.

Setting up a code editor

There is a significant difference between a code editor and an IDE. Code editor merely allows you to write the code. It provides all the necessary syntax highlighting and auto-formatting, but this is about it. You will need to either install special plug-ins or use external tools to be able to instantiate your projects or build your applications. But the biggest advantage that code editors have over IDEs is that code editors are much more light-weight and are much quicker to load.

IDE, on the other hand, is one-stop-shop for software development. All steps of developing your software can be done inside the IDE without having to rely on any external tools at all. Each of the actions, such as instantiating your project, running unit tests, building and running your application, would have a corresponding menu item in the GUI. But all these features come at a price. IDEs would occupy much more disk space than code editors and they are much more resource-hungry.

Whether you will decide to use a code editor or an IDE, it doesn't matter. All examples provided in this book would work with either.

Now, let's set up the tools specific to the OS you intend to use. You don't have to read each of this sections. Just pick the one that is relevant to you.

Setting up on Windows environment

If you are using Windows, here are the options that are available to you.

Visual Studio 2022

Perhaps the most popular .NET IDE for Windows is Visual Studio. In order to be able to work with .NET 6, you will need Visual Studio 2022. You can download it from its official page[4].

[3]https://dotnet.microsoft.com/en-us/download/dotnet/6.0
[4]https://visualstudio.microsoft.com/downloads/

There are three editions of Visual Studio 2022: Community, Professional and Enterprise. The community edition is free to download, but you will need to register.

JetBrains Rider

Rider IDE by JetBrains doesn't come for free. But there is a reason for it. It contains many useful tools, such as ReSharper, that allows you to decompile .NET assemblies back into human-readable code.

But even though the IDE itself isn't free, there is a free trial available. It can be downloaded from its official page[5].

Visual Studio Code

Even though this tool sounds like Visual Studio, it's a completely different tool. Visual Studio Code (also known as VS Code) is a lightweight, but powerful, code editor. It comes completely free of charge and you won't need to register to use it. Also, it's open-source and ever-green, which means that it will continuously receive updates, so you won't have to download a new version every time a new SDK becomes available. On top of all of that, it has a very powerful plugin system, so you can almost turn it into a fully-fledged IDE.

VS Code is available on any platform. And you can download it from its official website[6].

Setting up on Mac OS environment

The following options are recommended if you are a Mac user.

Visual Studio for Mac

Visual Studio for Mac is a Mac version of the classic Windows-based Visual Studio IDE. You can download it from its official page[7]. It is available to download free of charge, but you will have to register.

JetBrains Rider

Rider IDE by JetBrains doesn't come for free. But there is a reason for it. It contains many useful tools, such as ReSharper, that allows you to decompile .NET assemblies back into human-readable code.

But even though the IDE itself isn't free, there is a free trial available. It can be downloaded from its official page[8].

[5] https://www.jetbrains.com/rider/
[6] https://code.visualstudio.com/
[7] https://visualstudio.microsoft.com/downloads/
[8] https://www.jetbrains.com/rider/

Visual Studio Code

Even though this tool sounds like Visual Studio, it's a completely different tool. Visual Studio Code (also known as VS Code) is a lightweight, but powerful, code editor. It comes completely free of charge and you won't need to register to use it. Also, it's open-source and ever-green, which means that it will continuously receive updates, so you won't have to download a new version every time a new SDK becomes available. On top of all of that, it has a very powerful plugin system, so you can almost turn it into a fully-fledged IDE.

VS Code is available on any platform. And you can download it from its official website[9].

Setting up on Linux environment

Linux has fewer options available than either Windows or Mac OS. However, you will still be able to set up your development environment with the full set of capabilities.

JetBrains Rider

Rider IDE by JetBrains doesn't come for free. But there is a reason for it. It contains many useful tools, such as ReSharper, that allows you to decompile .NET assemblies back into human-readable code.

But even though the IDE itself isn't free, there is a free trial available. It can be downloaded from its official page[10].

Visual Studio Code

Even though this tool sounds like Visual Studio, it's a completely different tool. Visual Studio Code (also known as VS Code) is a lightweight, but powerful, code editor. It comes completely free of charge and you won't need to register to use it. Also, it's open-source and ever-green, which means that it will continuously receive updates, so you won't have to download a new version every time a new SDK becomes available. On top of all of that, it has a very powerful plugin system, so you can almost turn it into a fully-fledged IDE.

VS Code is available on any platform. And you can download it from its official website[11].

Enabling development HTTPS certificate

This step is not strictly necessary, but it will allowed you to add HTTPS protocol to your applications via a self-signed development certificate. .NET SDK comes with its own self-signed certificate and to ensure that it works, you need to trust it. To do so on either Windows or Mac OS, all you have to do is run the following command:

[9] https://code.visualstudio.com/
[10] https://www.jetbrains.com/rider/
[11] https://code.visualstudio.com/

```
dotnet dev-certs https --trust
```

On Linux, the process of trusting the certificate would be different, so you would need to read the manual specific to the distro you are using. However, some guidance will be provided in **further reading** section.

Or you may choose not to use HTTPS at all. It's up to you, as it's not strictly needed to work with the code samples from the book. But please use it if you can, as it will make your application slightly more similar to what you would build in a real-life project.

Setting up the solution

In your file system, create a folder and call it `LearningSignalR`. Now, open any command line terminal of your choice. It doesn't matter whether it's PowerShell, cmd, bash or anything else that your OS has. The commands we will execute will work on any of them.

In the terminal, make sure you navigate to the newly created `LearningSignalR` folder and execute the following command inside of it:

```
dotnet new sln
```

This command is expected to generate a solution file with the same name as the folder. So, in our case, to ensure that the command has worked, we need to check whether our folder contains `LearningSignalR.sln` file.

Next, we will instantiate a project based on ASP.NET Core MVC template. To do so, while having the terminal open in the solution folder, we will excute the following command:

```
dotnet new mvc -o SignalRServer
```

This command will generate `SignalRServer` folder inside the solution folder with all the default code setup for ASP.NET Core MVC project. We will now need to add the project to the solution. And to do so, this is the command we will need to execute:

```
dotnet sln add SignalRServer/SignalRServer.csproj
```

The right-most argument of this command is the relative path to the C# project file, which has `csproj` extension.

That's it. We now have a solution with an MVC project in it. If you are using an IDE, you can open the solution by double-clicking on the `sln` file. Otherwise, you can use your code editor to open the solution folder.

Now, we are ready to add SignalR components to our project.

Setting up SignalR hub

The server-side SignalR components center around the so-called **hub**. SignalR hub is an equivalent to MVC or Web API controller. Or, if you are familiar with gRPC, it is an equivalent of gRPC service implementation. It is a class with a collection of methods that SignalR clients will be able to remotely trigger.

We will add a basic example of a hub to our project and register all necessary dependencies. Then, we will go through its structure in a little bit more detail.

Adding SignalR hub to the project

We will follow similar conventions to MVC. As we have **Models**, **Views** and **Controllers** folders inside `SignalRServer` project, we will add another folder and call it **Hubs**. We will then create `LearningHub.cs` file inside this folder and populate it with the following content:

```
using Microsoft.AspNetCore.SignalR;

namespace SignalRServer.Hubs
{
    public class LearningHub : Hub
    {
        public async Task BroadcastMessage(string message)
        {
            await Clients.All.SendAsync("ReceiveMessage", message);
        }

        public override async Task OnConnectedAsync()
        {
            await base.OnConnectedAsync();
        }

        public override async Task OnDisconnectedAsync(Exception? exception)
        {
            await base.OnDisconnectedAsync(exception);
        }
    }
}
```

So, let's go through the structure of the class. This is a basic example, so don't worry. You won't be overwhelmed with the information.

SignalR hub overview

Firstly, a class that we want to use as a SignalR hub needs to inherit from `Hub` class, which is a part of `Microsoft.AspNetCore.SignalR` namespace. This namespace is included in the standard ASP.NET Core libraries, so we won't have to add any external references. But we still need to reference this namespace somewhere. We can either do it in the file containing the class, like we have done in the above example, or you can just add the `using` statement anywhere in your application and prepend `global` keyword to it to make it available everywhere within the project. There is no difference between these two ways of referencing namespaces. You can choose either, depending on your personal preferences or any specific guidelines that you follow.

Inside the hub, we have `BroadcastMessage` method that accepts a `string` parameter. This is an example of a method that clients will be able to call once they are connected to the hub. There is nothing special about this method, other than it needs to return a `Task`. It can have any parameters of any data types. The messages are sent and received in the form of JSON, so any data that gets transferred can be easily deserialized either into basic data types, like `string` or `int` or more complex types, like `class` or `struct`.

There is a limit to how many parameters you can have. But the number is reasonably large, so, as long as you are writing clean code and following best practices, you shouldn't worry about running out of available parameters.

So, `BroadcastMessage` method receive a `string` message from a client. Then, inside the method, this message gets re-sent to all clients, including the one that has sent it. Sending message to any entity that has tuned in is known as broadcasting. And this is precisely why the method is called `BroadcastMessage`.

But how does the hub know what clients are connected to it? Well, that's easy. The base `Hub` class has a property called `Clients`. This property has the details of all connected clients. And you can use this property to choose which clients to send the message to.

In our case, to make things as simple as possible, we are choosing to send the message to all clients. We do so by accessing `All` property. But there are multiple different ways of selecting specific clients to send the message to. We will cover this in **chapter 5**.

To actually send the message and trigger a specific event in the client code, we call `SendAsync` method. The first parameter of this method is the even name in the client code. The spelling has to be exactly the same as it's spelled on the client. Otherwise the event won't be triggered. The other parameters are the input parameters for the event.

Next, we have overrides of `OnConnectedAsync` and `OnDisconnectedAsync` methods from the original `Hub` class. In our example, we aren't doing anything with them. But these are the methods that get triggered when a client establishes connection or disconnects. Because disconnection can happen due to an error, `OnDisconnectedAsync` methods also has nullable `Exception` parameter.

There are also some other public members of `Hub` class that we haven't covered. There is `Groups` property, which allows you to assign individual clients to groups and then broadcast messages to the members of a particular group. There is also `Context` property, which contains information of the current session. For example, you can extract a unique client identifier from this property, which gets auto-generated when the client connects and remains constant until the client disconnects. `Context` property is conceptually similar to `HttpContext` from Web API controllers. We will cover both `Groups` and `Context` in **chapter 5**.

One important thing to remember about a SignalR hub is that its lifetime is restricted to a single call. So don't store any durable data in it. It will all disappear once the next call is executed.

And this completes a basic overview of SignalR hub. Now, we need to enable it, so our clients can actually access it.

Enabling SignalR hub endpoint

Enabling SignalR hub is simple. All we have to do is add a couple of lines to our `Program.cs` file. First, we will need to reference the hub namespace by adding this `using` statement to the file:

```
using SignalRServer.Hubs;
```

You can prepend it with `global` keyword to make it universally accessible to all files, so you won't have to insert the `using` statement again.

Next, we will need to add the following line just before `app` variable gets instantiated:

```
builder.Services.AddSignalR();
```

This statement enables us to use SignalR middleware in the application. Finally, we need to add the following line before `Run` method is called on the `app` variable:

```
app.MapHub<LearningHub>("/learningHub");
```

In this line, we have mapped our SignalR hub to a specific path in the URL. So, a client is now able to register with the hub by submitting a HTTP request to `{Base URL}/learningHub` address. The initial request is done via the standard HTTP protocol. But then, if the protocol needs to change (for example, to WS, which represents WebSocket), this will happen under the hood. You, as a developer, won't have to worry about it.

That's it. Our hub is now registered and our SignalR server is ready to accept client connections. But there is one additional modification that we can apply to the hub to minimize the risk of misspelling the client event names. And this is what we will do next.

Making SignalR hub strongly-typed

So far, when we've been calling clients from the hub, we have used just a normal `string` to specify the client-side event name that needs to be triggered. But the problem with using a `string` is that it can contain any arbitrary text. If we misspell the name, the code will not warn us. Perhaps, it's not a big problem in our case, because we only have one such a call, so it's easy enough to manage. But what if we had a complex hub (or multiple hubs) with many of such calls?

Fortunately, there is a solution. You can make your hub strongly-typed, so any method you have on the client can be transformed into a C# method representation. And you can't misspell a name of a C# method. Your IDE will show an error and your code won't compile.

To do so, we will need to add an interface that represent client events. And then we will enforce this interface on the hub. We will start by adding `ILearningHubClient.cs` file to the root folder of our project. This file will contain the following content:

```
namespace SignalRServer
{
    public interface ILearningHubClient
    {
        Task ReceiveMessage(string message);
    }
}
```

So, as you can see, we have a `Task` method with the name of `ReceiveMessage` - the same name that we previously specified as the client-side event name in the call to `SendAsync` method. Now, we will modify our hub. To make it simple, we will replace the content of `LearningHub.cs` file with the following:

```
using Microsoft.AspNetCore.SignalR;

namespace SignalRServer.Hubs
{
    public class LearningHub : Hub<ILearningHubClient>
    {
        public async Task BroadcastMessage(string message)
        {
            await Clients.All.ReceiveMessage(message);
        }

        public override async Task OnConnectedAsync()
        {
            await base.OnConnectedAsync();
        }

        public override async Task OnDisconnectedAsync(Exception? exception)
        {
            await base.OnDisconnectedAsync(exception);
        }
    }
}
```

Let's break it down. Our class no longer just inherits from `Hub` base class. We also have our interface name in the angle brackets next to the base class name. And that forces us to not be able to use `SendAsync` method on the objects that represent clients. Instead, we have to use any of the methods defined in the interface. So now, we can no longer accidentally misspell the name.

And that concludes the overview of a SignalR hub. We are now ready to start connecting clients to it. But first, let's summarize what we have covered.

Summary

SignalR hub is a server-side class that inherits from `Hub` class of `Microsoft.AspNetCore.SignalR` namespace. This is the class that would have any arbitrary methods that the clients will be able to trigger remotely.

As well as having arbitrary methods, SignalR hub can override the default connection and disconnection events. The disconnection even accepts a nullable `Exception` parameter, because a disconnection may happen due to a failure. Other than that, a hub class would have access to `Clients`, `Groups` and `Context` properties.

SignalR hub is not durable, so no persistent data can be stored in its variables. The data will be deleted before the next call. To enable SignalR hub, you need to enable SignalR middleware in `Program.cs` file and then map a specific hub to a specific URL path.

SignalR hub can be strongly typed. This is achieved by creating an interface containing allowed client methods and then forcing the client-related properties of the hub to implement it.

In the the next chapter, we will start connecting clients to our hub. We will focus on in-browser clients first.

Test yourself

1. What is a SignalR hub?
   A. A cloud provider hosting SignalR service
   B. The application that hosts SignalR components

   C. A class that contains the methods that clients can trigger
   D. The machine that hosts SignalR application
2. Which of the following properties exists in the `Hub` base class?
   A. Context
   B. Clients
   C. Groups
   D. All of the above
3. What is the best way to minimize the risk of misspelling client-side event name on SignalR server?
   A. Save it in a `const` variable
   B. Save it in a `readonly` variable
   C. Enforce client interface on the hub
   D. Have a unit test to check the spelling

Further reading

Official SignalR Hub documentation: https://docs.microsoft.com/en-us/aspnet/core/signalr/hubs

3 - In-browser SignalR clients

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Prerequisites

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Setting up JavaScript client

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Adding SignalR client dependencies

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Adding SignalR library via CDN

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Adding SignalR library via NPM

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Adding SignalR client logic

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Overview of JavaScript SignalR client implementation

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Launching JavaScript client

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Setting up Blazor WebAssembly client

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Setting up SignalR client components

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Hosting Blazor application inside an existing ASP.NET Core application

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Launching Blazor client

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Test yourself

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Further reading

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

4 - External SignalR clients

In the previous chapter, we have covered the basic of using in-browser clients to connect to a SignalR hub. Even though these clients are suitable to cover a wide range of SignalR use cases, they aren't sufficient to cover them all. Not all of your clients will be running in the browser. And not all your clients will be a part of the same web application.

In this chapter, we will cover external SignalR clients. All of these can be set up from a stand-alone application of any type. The SignalR client can be set up in some background service running on an IoT device. Or it can be in the back-end of a mobile app.

The chapter covers the following topics:

- Setting up .NET client
- Setting up Java client
- Setting up a raw WebSocket client

By the end of this chapter, you will have learned how to use officially supported SignalR clients in stand-alone application. But, on top of this, you will have learned how to get a bare WebSocket to communicate with a SignalR server. This knowledge will give you the ability to write your own SignalR client in any language that isn't officially supported.

Prerequisites

This chapter assumes that you already have set up your development environment, as described in **chapter 1**. You will need the following:

- A machine with either Windows, Mac OS or Linux operating system
- A suitable IDE or code editor (Visual Studio, JetBrains Rider or VS Code)
- .NET 6 SDK (or newer)

Also, since we are continuing to build on top of the application that we have developed in the previous chapter, we need the code that we have written previously. If you have skipped the previous chapter, you can access the complete code from the following location in the GitHub repository:

https://github.com/fiodarsazanavets/SignalR-on-.NET-6---the-complete-guide/tree/main/Chapter-03/Part-02/LearningSignalR

The complete code samples from this chapter are available from the following location in the GitHub repo, which has separate folders corresponding to individual parts of the chapter:

https://github.com/fiodarsazanavets/SignalR-on-.NET-6---the-complete-guide/tree/main/Chapter-04

Setting up .NET client

In **chapter 3**, we have already set up .NET SignalR client inside of a Blazor WebAssembly application. The process that we will go through now will be similar. For example, we will rely on the same NuGet package. But this time, we will set the client up inside a stand-alone console application.

Setting up .NET console app as a SignalR client

The first thing that we will need to do is to create our console application. Because this would be a stand-alone application without any dependencies shared with the other projects in your solution, it's up to you whether you create it inside the solution folder and add it to the solution. Keeping the new project inside the solution may make it easier to manage, but it isn't strictly necessary.

Once you have selected the folder that you will create the console application in, execute the following command inside that folder to instantiate the project:

```
dotnet new console -o DotnetClient
```

Now, we will need to add a SignalR NuGet package to it. To do it, open your command line terminal inside the project folder and execute the following command:

```
dotnet add package Microsoft.AspNetCore.SignalR.Client
```

Next, we will apply some SignalR client logic to our `Program.cs` file inside of the project. To do so, we will delete all existing content from this file. Then, we will add the following statement to reference the namespace of SignalR client library:

```
using Microsoft.AspNetCore.SignalR.Client;
```

Next, we will add a prompt for the full SignalR hub URL and will create a `hubConnection` object based on it:

```
Console.WriteLine("Please specify the URL of SignalR Hub");

var url = Console.ReadLine();

var hubConnection = new HubConnectionBuilder()
                         .WithUrl(url)
                         .Build();
```

Next, we will map `ReceiveMessage` event to the `hubConnection` object. Every time the server-side hub would trigger this event, the message sent from the server would be written in the console:

```
hubConnection.On<string>("ReceiveMessage",
    message => Console.WriteLine($"SignalR Hub Message: {message}"));
```

After this, we will add a loop that will allow us to carry on using the client until we explicitly type `exit`. Every message that we type will be sent as a parameter to `BroadcastMessage` method on the SignalR hub:

```
try
{
    await hubConnection.StartAsync();

    while (true)
    {
        var message = string.Empty;

        Console.WriteLine("Please specify the action:");
        Console.WriteLine("0 - broadcast to all");
        Console.WriteLine("exit - Exit the program");

        var action = Console.ReadLine();

        Console.WriteLine("Please specify the message:");
        message = Console.ReadLine();

        if (action == "exit")
            break;

        await hubConnection.SendAsync("BroadcastMessage", message);
    }
}
catch (Exception ex)
{
    Console.WriteLine(ex.Message);
    Console.WriteLine("Press any key to exit...");
    Console.ReadKey();
    return;
}
```

The above code consist of our SignalR logic and basic error handling, which is always a good idea to do to prevent our application from crashing unexpectedly.

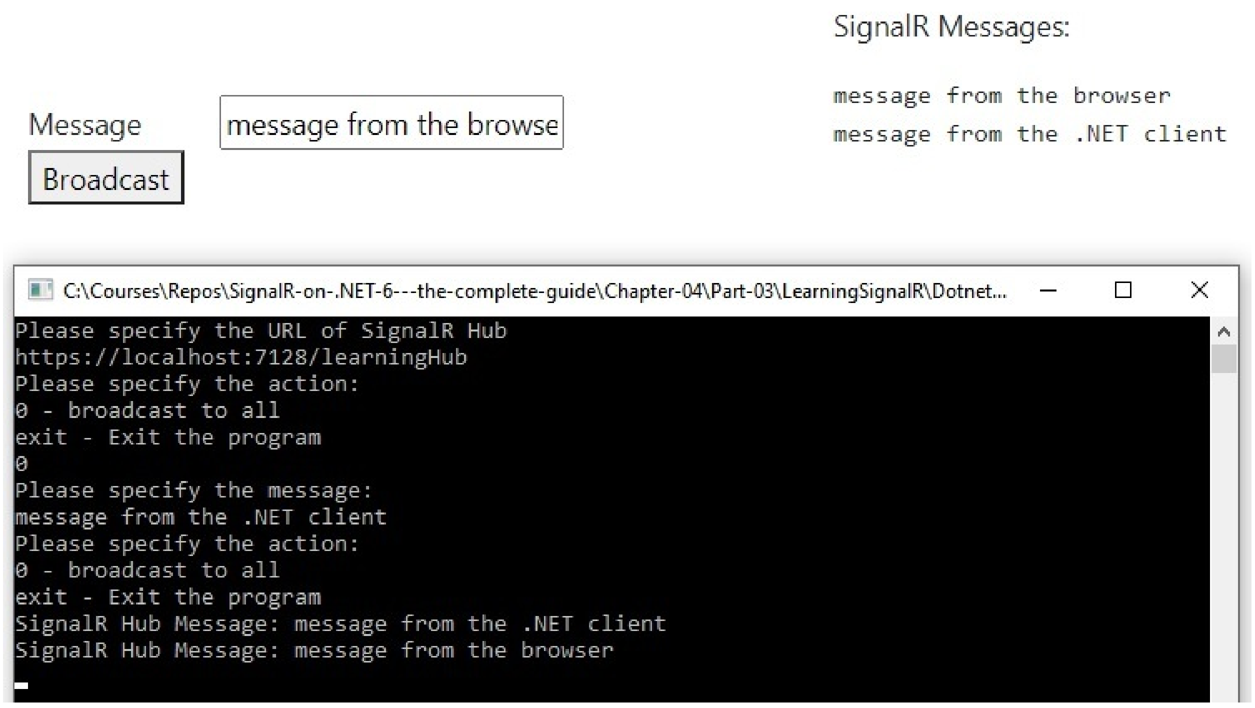
Now, we can launch our application and see it in action.

Testing our .NET client

First, we will need to launch our SignalR server application. We can do so by executing `dotnet run` command from `SignalRServer` project folder. Then, once the application is up and running, we can launch the SignalR client console application by executing `dotnet run` command from `DotnetClient` project folder.

When we are prompted to enter the SignalR hub URL, we can enter the URL listed in `applicationUrl` section of `launchSettings.josn` file of `SignalRServer` project followed by `/learningHub`. So, for example, if your base application URL is `https://localhost:7128`, then the URL you need to enter is `https://localhost:7128/learningHub`.

For the testing purposes, we may want to open the home page of our web application to get JavaScript client running in the browser. Because both applications have been set up to broadcast message to all connected SignalR clients, what you will see is that whenever you enter a message in your console application, in-browser application will receive it. Likewise, if you send a message from your in-browser application, your console application will receive it. This can be seen on the following screenshot:


Message
message from the browse
Broadcast
SignalR Messages:
C:\Courses\Repos\SignalR-on-.NET-6---the-complete-guide\Chapter-04\Part-03\LearningSignalR\Dotnet...

```
message from the browser
message from the .NET client
```

```
Please specify the URL of SignalR Hub
https://localhost:7128/learningHub
Please specify the action:
0 - broadcast to all
exit - Exit the program
0
Please specify the message:
message from the .NET client
Please specify the action:
0 - broadcast to all
exit - Exit the program
SignalR Hub Message: message from the .NET client
SignalR Hub Message: message from the browser
```

Figure 4.1 - .NET console application can communicate with JavaScript client

And this demonstrates the basics of running SignalR client inside a stand-alone .NET application. Next, we will cover another supported client type - Java client. If you are not a Java developer and you never intend to build a Java client, you can skip this section. But because Java is a widely used and universal language, this information will still be useful.

Setting up Java client

Just like .NET 6, Java is a universal language that can be used on any of the popular operating systems. To use it, you will first need to install Java SDK. There are multiple ways you can do it. But if you already have .NET environment set up, perhaps the easiest way to do it is to download it via VS Code extension pack[12]. Although there are several Java IDEs, VS Code is more than adequate as a code editor.

Once you have installed Java SDK, you will need to install one of its build tools. The most popular of these are Gradle and Maven. They work differently, but the outcome will be roughly the same. You will end up with the same Java code. And you will be able to compile and run your application either way.

We will be using Maven package repository to download our SignalR client library from. But Gradle can access it too. So it doesn't matter which build tool you use. It's outside the scope of this book to provide a detailed instruction on how to install and use either Gradle or Maven. But referenced to detailed user manuals are available in the **further reading** section of this chapter.

Setting up Java project

Once you have your build tool set up, we need to instantiate a Java project. This process will be different depending on whether you are using Gradle or Maven.

Generating a project template with Gradle

If you are using Gradle, you need to run the following command inside any folder of your choice:

```
gradle init
```

Then, you will be asked to provide various parameters to your project. You can select anything for most of the options (or just use defaults), but do make sure that the language that you have selected is Java and the project type is `application`. With these options selected, it will instantiate your project with some code already being inside.

The SignalR package will work with other languages, such as Kotlin. After all, it's a JVM language, just like Java. But since Java has been out there for much longer, this is the language that you will see the samples in.

Generating a project template with Maven

With Maven, you need to generate a project from a so-called archetype. It's equivalent to .NET project template. And perhaps one of the simplest archetypes is `maven-archetype-quickstart`. To generate a project from it, you can use the following command:

[12]https://code.visualstudio.com/docs/languages/java

```
mvn archetype:generate -DarchetypeGroupId=org.apache.maven.archetypes -DarchetypeArt\
ifactId=maven-archetype-quickstart -DarchetypeVersion=1.4
```

Whether you have used Gradle or Maven, you should now have a folder structure that contains `App.java` file in it with `main` method. This is equivalent to `Program.cs` in C#. And this is the file we will be adding SignalR client to. But first we will need to add a reference to a relevant dependency.

Adding SignalR client Java dependency

You will first need to visit the following page and verify what is the latest version of SignalR package is:

https://search.maven.org/artifact/com.microsoft.signalr/signalr

Then you will need to click on that version number to open the page, which will provide you dependency insertion syntax for various build manager types. For example, assuming that the latest version of the version package is 6.0.0, this is the markup you will need to insert into the `dependencies` section of the `pom.xml` file if you are using Maven:

```
<dependency>
  <groupId>com.microsoft.signalr</groupId>
  <artifactId>signalr</artifactId>
  <version>6.0.0</version>
</dependency>
```

This would be the entry you would need to insert into `dependencies` section of `gradle.build` file if you have used Groovy DSL during project setup:

```
implementation 'com.microsoft.signalr:signalr:6.0.0'
```

This would be the entry you would need to insert into `dependencies` section of `gradle.build.kts` file if you have used Kotlin DSL during project setup:

```
implementation("com.microsoft.signalr:signalr:6.0.0")
```

And there are quite a few other options, as Maven and Gradle are far from being the only build tools for Java.

Once you have added the dependencies, you may want to remove the default tests (if you have any), as those would cause the build to break after you've changed the content of `App.java` file. And now we are ready to start modifying the code.

Adding SignalR client code to Java application

We will open our `App.java` file and ensure that we have all of the following `import` statements in it:

```java
import com.microsoft.signalr.HubConnection;
import com.microsoft.signalr.HubConnectionBuilder;
import java.util.Scanner;
```

We will then need to ensure that the signature of the `main` method looks like follows:

```java
public static void main(String[] args) throws Exception {
```

Inside this method, we prompt the user to enter the URL of the SignalR hub:

```java
System.out.println("Please specify the URL of SignalR Hub");
Scanner reader = new Scanner(System.in);
String input = reader.nextLine();
```

We can’t read textual input from the console directly, like we can in C#. Therefore we are using `Scanner` class for it. Then we build our `hubConection` object and map a`ReceiveMessage` event to it:

```java
HubConnection hubConnection = HubConnectionBuilder.create(input)
        .build();

hubConnection.on("ReceiveMessage", (message) -> {
    System.out.println(message);
}, String.class);
```

Then we start the connection and getting the application to send any messages that we type to `BroadcastMessage` method on the SignalR hub. At any point, we can type `exit` and this will stop the connection:

```java
hubConnection.start().blockingAwait();

while (!input.equals("exit")){
    input = reader.nextLine();
    hubConnection.send("BroadcastMessage", input);
    }

hubConnection.stop();
```

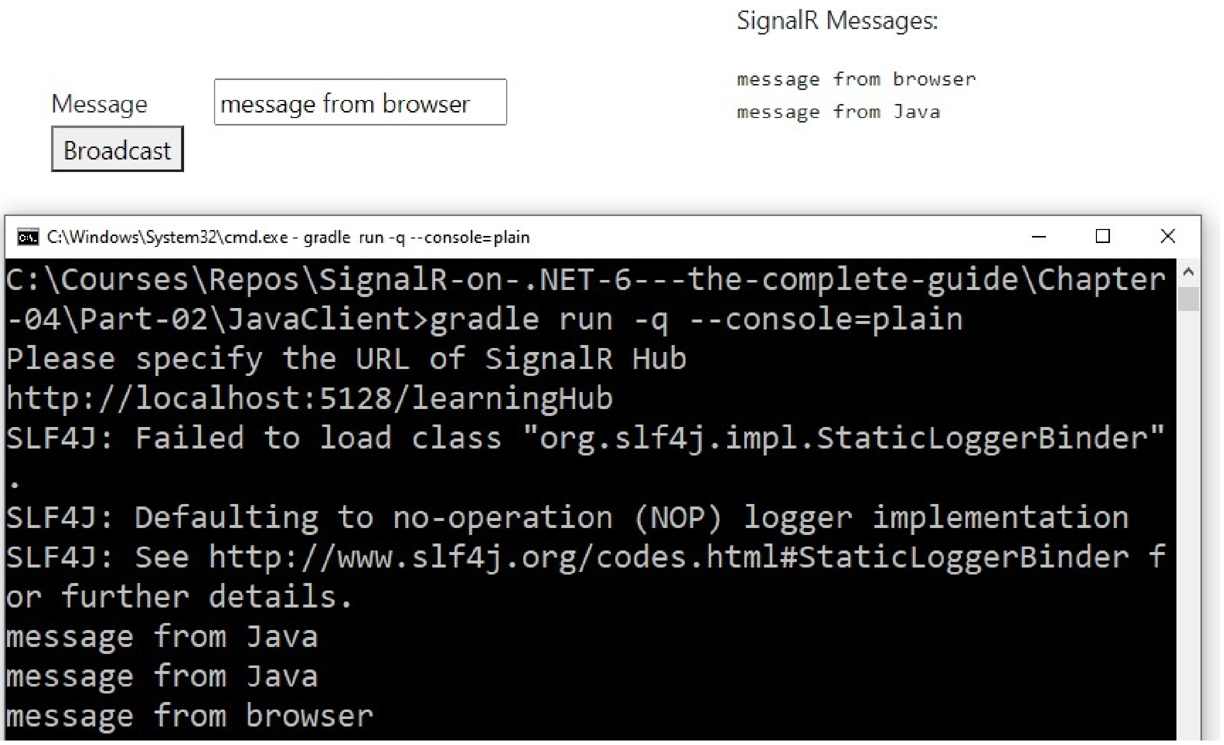
`reader` object (which is an instance of `Scanner`) that we have created earlier just keeps reading the messages from the console. And this completes the setup of our Java application. We can now launch it to see it in action.

Launching Java SignalR client

To launch our `SignalRServer` application, all we have to do is execute `dotnet run` command from its project folder. But with a Java application, building and launching it will depend on the build manager that you use. It could be as simple as executing `gradle run` command. But it could be more complicated if you use any other build tools. So please check the documentation of your build tool for the exact command that you need to execute.

Another caveat is that, depending on configuration, Java HTTP client may not work with development HTTPS certificate. If this is the case, then the easiest way to resolve it is to remove the HTTPS URL from `applicationUrl` entry of `launchSetting.json` file of your `SignalRServer` application.

Once you launch the application, you can verify that it can send and receive messages. The following screenshot shows an example of the application that was set up with Kotlin DSL on Gradle. The launch command was `gradle run -q --console=plain`, which makes it run as a plain console.



```
message from browser
message from Java
```

```
C:\Courses\Repos\SignalR-on-.NET-6---the-complete-guide\Chapter
-04\Part-02\JavaClient>gradle run -q --console=plain
Please specify the URL of SignalR Hub
http://localhost:5128/learningHub
SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder"
.
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder f
or further details.
message from Java
message from Java
message from browser
```

Figure 4.2 - Java client in action

You can safely ignore any errors related to `slf4j`. These errors simply mean that no default logging provider was found. You will need to enable an additional third-party package to make these errors disappear.

And this concludes our overview of Java client, which, at the time of writing, is the only client that is supported outside of .NET. But the good news is that SignalR supports raw WebSockets too, which can be written in any language. And this is what we will have a look at next.

Setting up a raw WebSocket client

We have now covered all SignalR client types that Microsoft officially supports. But what if the technology you want to use is not on the list? Also, what if there is no way to actually write a SignalR client and you need to connect an existing WebSocket client to it?

Well, the good news is that you can connect a raw WebSocket to SignalR hub. And once it is connected, you can easily see the structure of the messages that are being exchanged. This will allow you to write your own SignalR client implementation in any language of your choice, as WebSocket is a standard protocol which you can write code for in any language. But today we will focus on .NET implementation of it.

Setting up WebSocket client

We will create another project and call it `WebSocketClient`. You may choose to include this project in an existing solution. But you don't have to, as it will not share any direct dependencies with any of the projects inside your solution.

The project will be based on .NET **Console Application** template. To initiate the project, execute the following command in a folder of your choice:

```
dotnet new console -o WebSocketClient
```

You won't need to add any external dependencies to the project at all. WebSocket library is already included in .NET `System` library.

The first thing we will need to do in the new project is delete all existing content from the `Program.cs` file. Then, we will add references to WebSocket and text processing namespaces. To do so, add the following `using` statements:

```
using System.Net.WebSockets;
using System.Text;
```

Next, we will add some code to prompt the user to enter WS protocol URL pointing at the SignalR hub endpoint:

```
Console.WriteLine("Please specify the URL of SignalR Hub with WS/WSS protocol");
var url = Console.ReadLine();
```

WebSocket uses WS instead of HTTP in the URLs (and WSS instead of HTTPS). But otherwise, it uses the same TCP/IP protocol as HTTP. Therefore you will be able to point it at the same addresses and ports. So, your SignalR hub endpoint URL will hardly change. Instead of being `https://{base URL}/learningHub`, it will become `wss://{base URL}/learningHub`.

Next, we will connect our WebSocket and send the initial request to SignalR hub. Our request is nothing more than a handshake in JSON format that SignalR hub expects. It defines the protocol and the version.

```
try
{
    var ws = new ClientWebSocket();

    await ws.ConnectAsync(new Uri(url), CancellationToken.None);

    var handshake = new List<byte>(Encoding.UTF8.GetBytes(@"{""protocol"":""json"", \
""version"":1}"))
                {
                    0x1e
                };

    await ws.SendAsync(new ArraySegment<byte>(handshake.ToArray()), WebSocketMessage\
Type.Text, true, CancellationToken.None);

    Console.WriteLine("WebSockets connection established");
    await ReceiveAsync(ws);
}
catch (Exception ex)
{
    Console.WriteLine(ex.Message);
    Console.WriteLine("Press any key to exit...");
    Console.ReadKey();
    return;
}
```

As you can see, we cannot just send human-readable text over WebSocket. We need to convert it to bytes. So, as this example clearly shows, SignalR makes your job much easier compared to raw WebSocket programming.

The last statement that we have entered inside the `try` block is a call to `ReceiveAsync` method. We will add this method now:

```
static async Task ReceiveAsync(ClientWebSocket ws)
{
    var buffer = new byte[4096];

    try
    {
        while (true)
        {
            var result = await ws.ReceiveAsync(new ArraySegment<byte>(buffer), Cance\
llationToken.None);
```

```
            if (result.MessageType == WebSocketMessageType.Close)
            {
                await ws.CloseOutputAsync(WebSocketCloseStatus.NormalClosure, string\
.Empty, CancellationToken.None);
                break;
            }
            else
            {
                Console.WriteLine(Encoding.Default.GetString(Decode(buffer)));
                buffer = new byte[4096];
            }
        }
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
        Console.WriteLine("Press any key to exit...");
        Console.ReadKey();
        return;
    }
}
```

In this method, we are using a byte buffer of a fixed size. We then just carry on listening on WebSocket connection, while populating this buffer with bytes that we receive from it. Every time we receive some data, we reset the buffer and start again. We do this until WebSocket gets closed by the server.

Before we can actually convert bytes to human-readable message, we use `Decode` method. And this is what this method consists of:

```
static byte[] Decode(byte[] packet)
{
    var i = packet.Length - 1;
    while (i >= 0 && packet[i] == 0)
    {
        --i;
    }

    var temp = new byte[i + 1];
    Array.Copy(packet, temp, i + 1);
    return temp;
}
```

This method is necessary, because we use buffer of a fixed size, which will inevitably contain empty bytes. If we just attempt to convert it into text as is, the empty bytes will be converted too. There is actually a textual symbol that represents them. And this would make our message less readable. To prevent this from happening, we are removing all empty bytes from our message before we process it.

And this concludes the basic setup of WebSocket client. Even though we have only added the most basic WebSocket functionality, the code that we have written is already fairly complicated. This is why you shouldn't use raw WebSocket client unless you absolutely have to. Because SignalR exists, writing raw WebAssembly is almost like writing a web application in Assembler.

Now, we can launch our WebSocket application and see it in action.

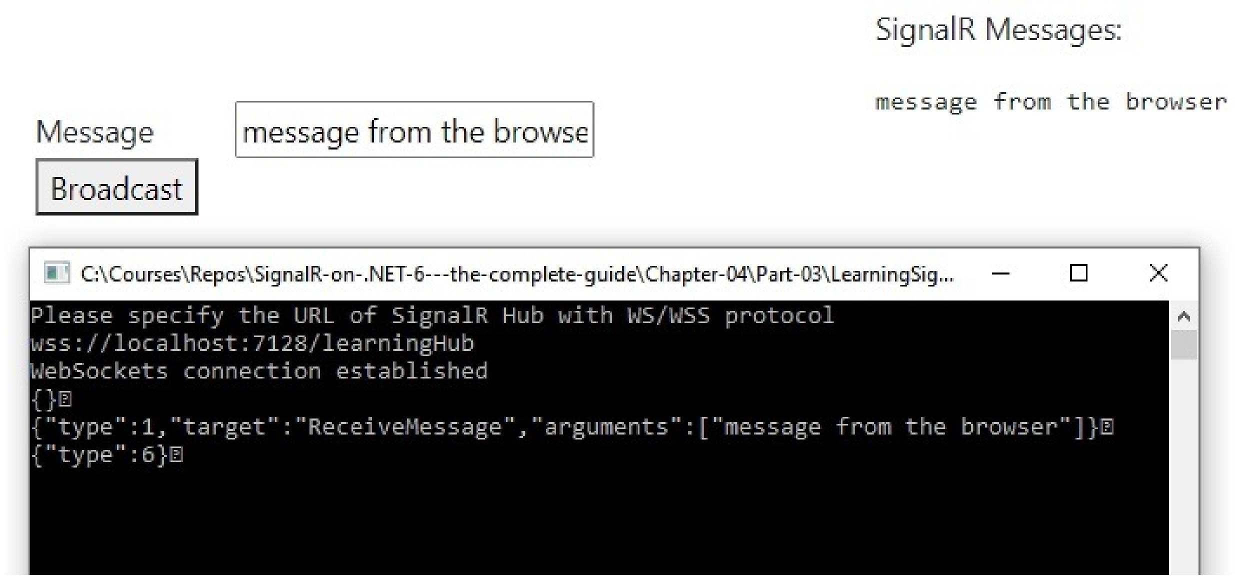
Launching WebSocket client

We will now launch our `SignalRServer` project by executing `dotnet run` command in the project directory. Then we will execute the same command inside `WebSocketClient` project directory.

When our WebSocket client application launches, it will ask us to provide the URL to the SignalR endpoint. But remember that we are dealing with WebSocket protocol here and not with HTTP. Therefore, although we can use our original Hub URL, we will need to replace `https` prefix with `wss` (or, if you are using `http` URL, replace it with `ws`).

We don't have any listeners in the WeBSocket client. But we still receive the messages. And, if the connection was successful, what you will notice right away is that we have received an empty object from the server. This was a normal part of the handshake.

But interesting things will start to happen when we start broadcasting messages from the home page of our SignalR Server application. Our WebSocket client will pick them up, but it won't be just the content of the message itself. It will come as a JSON with some other fields, like it can be seen from the following screenshot:


SignalR Messages:
message from the browser
Message
message from the browse
Broadcast
C:\Courses\Repos\SignalR-on-.NET-6---the-complete-guide\Chapter-04\Part-03\LearningSig...

```
Please specify the URL of SignalR Hub with WS/WSS protocol
wss://localhost:7128/learningHub
WebSockets connection established
{}
{"type":1,"target":"ReceiveMessage","arguments":["message from the browser"]}
{"type":6}
```

Figure 4.3 - WebSocket client picks up detailed messages from the server

In this JSON object, we have `type` field, which is set to 1. Then we have `target` field, which is set to the name of the event handler. And then we have `arguments` field, containing a collection of parameters. And this is how SignalR middleware knows what events to trigger and what parameters to apply. It's a simple format. And if you get familiar with it, you will be able to write your own SignalR client implementation in any language of your choice.

But another interesting thing happens when you leave the application running for a while. You will start occasionally receiving a JSON message that has nothing in it except `type` field with the value of 6. Well, this is a heartbeat message. And its purpose is to ensure that the connection is still operational. `type` field tells the system what kind of message it is. It's 1 for a standard message and 6 for heartbeat. There are some more values, but you will need to study the code[13] of the library to learn them. After all, it's open source and publicly available.

And this completes our overview of SignalR clients. Let's summarize what we have learned.

Summary

In this chapter, you have learned how to set up a SignalR client inside a stand-alone .NET application. To do so, you will need to install a NuGet package with the SignalR client library. It's the same process for all .NET client types, including Blazor WebAssembly or a mobile app.

We have covered how to set up a Java client. To do so, you will need to install SignalR client package from Maven central package repository. Once done, you will be able to set up SignalR hub connection in any type of Java application back-end.

Finally, we have covered the use of raw WebSocket as a SignalR client. Even though we only had a look at .NET example, the overall principles of WebSocket programming will be the same in any

[13]https://github.com/SignalR/SignalR

language. And, as we so, SignalR messages are nothing more than nicely formatted JSON, we can use WebSocket programming to write a client library for it in any language that isn't officially supported.

In the next chapter, we will start adding some complexity to our SignalR clients. You will learn how to group clients together and how to send messages from the server-side hub to specific clients and not just broadcast them to all.

Test yourself

1. Which SignalR client types are officially supported?
   A. JavaScript
   B. .NET
   C. Java
   D. All of the above
2. What data can you read from SignalR messages if you connect a raw WebSocket to it?
   A. You cannot, as all data in encrypted
   B. Only the name of event handlers
   C. Message type, the name of the event handlers and full message payload
   D. Detailed message metadata, including the headers
3. If you write a SignalR client in Java, which type of build tool can you use?
   A. Gradle
   B. Maven
   C. Neither of the above
   D. Both of the above

Further reading

Official documentation of SignalR Java client: https://docs.microsoft.com/en-us/aspnet/core/signalr/java-client

Java client API references: https://docs.microsoft.com/en-us/java/api/com.microsoft.signalr

Gradle user manual: https://docs.gradle.org/current/userguide/userguide.html

Maven documentation: https://maven.apache.org/guides/

.NET WebSocket programming: https://docs.microsoft.com/en-us/aspnet/core/fundamentals/websockets

5 - Sending messages to individual clients or groups of clients

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Prerequisites

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Broadcasting messages to all clients

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Applying changes to JavaScript client

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Updating .NET Client

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Testing exclusive broadcasting functionality

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Sending messages to specific clients

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Enabling self-messages

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Sending messages to other clients

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Modifying SignalR hub

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Modifying the clients

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Seeing individual client messages in action

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Working with client groups

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Using SignalR groups inside the hub

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Enabling SignalR clients to use groups

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Test yourself

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Further reading

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

6 - Streaming in SignalR

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Prerequisites

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

What is streaming used for

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Why raw data cannot be sent as individual messages

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Why streaming collections also makes sense in SignalR

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Client streaming in SignalR

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Applying client streaming functionality to JavaScript client

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Applying client streaming functionality to .NET client

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Adding client streaming listener to SignalR hub

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Testing client streaming functionality

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Server streaming in SignalR

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Adding server streaming capabilities to SignalR hub

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Adding server streaming listener to JavaScript client

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Adding server streaming listener to .NET client

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Testing server streaming functionality

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Test yourself

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Further reading

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

7 - Advanced SignalR configuration

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Prerequisites

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Configuring SignalR server

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Top-level SignalR configuration

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

KeepAliveInterval

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

MaximumReceiveMessageSize

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

HandshakeTimeout

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

MaximumParallelInvocationsPerClient

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

EnableDetailedErrors

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

StreamBufferCapacity

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

SupportedProtocols

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

JSON message protocol settings

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

PropertyNamingPolicy

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Encoder

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

IncludeFields

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

IgnoreReadOnlyFields

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

IgnoreReadOnlyProperties

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

MaxDepth

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

NumberHandling

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

DictionaryKeyPolicy

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

DefaultIgnoreCondition

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

PropertyNameCaseInsensitive

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

DefaultBufferSize

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

ReadCommentHandling

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

ReferenceHandler

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

UnknownTypeHandling

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

WriteIndented

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Converters

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Applying advanced transport configuration

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Transports

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

CloseOnAuthenticationExpiration

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

ApplicationMaxBufferSize

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

TransportMaxBufferSize

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

MinimumProtocolVersion

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

TransportSendTimeout

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

WebSockets

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

LongPolling

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

AuthorizationData

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Configuring SignalR client

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Configuring JavaScript client

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

transport

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

headers

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

accessTokenFactory

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

logMessageContent

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

skipNegotiation

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

withCredentials

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

timeout

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Setting logging level

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Changeable options

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

serverTimeoutInMilliseconds

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

keepAliveIntervalInMilliseconds

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Configuring .NET client

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

AccessTokenProvider

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

HttpMessageHandlerFactory

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Headers

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

SkipNegotiation

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

ApplicationMaxBufferSize

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

ClientCertificates

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

CloseTimeout

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Cookies

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

DefaultTransferFormat

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Credentials

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Proxy

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

UseDefaultCredentials

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

WebSocketConfiguration

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

WebSocketFactory

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Setting logging level

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Applying dynamic configuration options

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

HandshakeTimeout

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

ServerTimeout

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

KeepAliveInterval

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Configuring Java client

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

withHeader

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

shouldSkipNegotiate

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

withHandshakeResponseTimeout

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

withTransport

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Dynamic configuration options

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

getServerTimeout / setServerTimeout

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

withHandshakeResponseTimeout

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

getKeepAliveInterval / setKeepAliveInterval

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Pros and cons of MessagePack protocol

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Enabling MessagePack on the server

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

SerializerOptions

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

WithSecurity

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

WithCompression

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

WithAllowAssemblyVersionMismatch

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

WithOldSpec

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

WithOmitAssemblyVersion

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Applying MessagePack on JavaScript client

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Applying MessagePack on .NET client

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Applying MessagePack on Java client

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Disadvantages of MessagePack protocol

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Test yourself

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Further reading

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

8 - Securing your SignalR applications

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Prerequisites

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

What is CORS and why it's important

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Setting up single sign-on provider

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Overview of OpenID Connect and OAuth

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Setting up IdentityServer 4

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Configuring SSO application

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Applying authentication in SignalR

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Setting up authentication on SignalR server

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Authority

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

ClientId

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

ClientSecret

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

ResponseType

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

CallbackPath

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

SaveTokens

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

RequireHttpsMetadata

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Authenticating JavaScript client

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Authenticating .NET client

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Authenticating Java client

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Retrieving access token from the SSO provider

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

JWT format structure

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Applying authorization in SignalR

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Creating a custom requirement

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Configuring authorization middleware in ASP.NET Core application

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Applying authorization rules to individual endpoints

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Test yourself

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Further reading

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

9 - Scaling out SignalR application

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Prerequisites

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Setting up Redis backplane

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Running Redis on Linux

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Running Redis on Mac

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Running Redis on Windows

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Running Redis on Docker (all operating systems)

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Running multiple hub instances via Redis backplane

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Setting up multiple SignalR hubs

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Hosting the same SignalR hub in different applications

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Moving SignalR hub to a class library

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Adding another web application to host the hub

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Launching distributed SignalR hub

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Using HubContext to send messages from outside SignalR hub

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Implementing HubContext in our application

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Testing HubContext on a distributed SignalR hub

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Test yourself

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Further reading

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

10 - Introducing Azure SignalR Service

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Prerequisites

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Setting up Azure SignalR Service

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Adding Azure SignalR Service dependencies to your application

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Making HubContext work with Azure SignalR Server

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Overview of Azure SignalR Service REST API

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Full list of Azure SignalR Service REST API endpoints

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Broadcast a message to all clients connected to target hub

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Broadcast a message to all clients belong to the target user

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Send message to the specific connection

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Check if the connection with the given connectionId exists

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Close the client connection

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Broadcast a message to all clients within the target group

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Check if there are any client connections inside the given group

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Check if there are any client connections connected for the given user

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Add a connection to the target group

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Remove a connection from the target group

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Check whether a user exists in the target group

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Add a user to the target group

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Remove a user from the target group

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Remove a user from all groups

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Authenticating into Azure SirnalR Service REST API

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Test yourself

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Further reading

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Wrapping up

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Answers to self-assessment questions

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Chapter 2

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Chapter 3

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Chapter 4

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Chapter 5

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Chapter 6

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Chapter 7

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Chapter 8

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Chapter 9

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.

Chapter 10

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/signalronnet6-thecompleteguide.