Quick answers to common problems

SignalR Real-time Application Cookbook

Use SignalR to create real-time, bidirectional, and asynchronous applications based on standard web technologies

Roberto Vespa

[PACKT] PUBLISHING open source* community experience distilled

SignalR Real-time Application Cookbook

Use SignalR to create real-time, bidirectional, and asynchronous applications based on standard web technologies

Roberto Vespa

[PACKT] open source
community experience distilled
PUBLISHING

BIRMINGHAM - MUMBAI

SignalR Real-time Application Cookbook

First published: April 2014

Production Reference: 1160414

Published by Packt Publishing Ltd.
Livery Place
35 Livery Street
Birmingham B3 2PB, UK.

ISBN 978-1-78328-595-2

www.packtpub.com

Cover Image by Aniket Sawant (aniket_sawant_photography@hotmail.com)

Credits

Author

Roberto Vespa

Reviewers

Sriram Cherukumilli

Robin Karlsson

Duncan Mole

Emanuele Rabino

Richard Seroter

Commisioning Editor

Saleem Ahmed

Acquisition Editor

Rebecca Youe

Content Development Editor

Dayan Hyames

Technical Editors

Dennis John

Sebastian Rodrigues

Copy Editors

Dipti Kapadia

Aditya Nair

Kirti Pai

Stuti Srivastava

Project Coordinator

Swati Kumari

Proofreaders

Simran Bhogal

Linda Morris

Indexer

Hemangini Bari

Production Coordinators

Pooja Chiplunkar

Manu Joseph

Cover Work

Manu Joseph

About the Author

Roberto Vespa has been passionate about programming computers since he was at high school, and he always wanted to do that for a living.

He has a degree in Electronic Engineering obtained in Italy, and has been working in the Information Technology industry since 1995, consulting on many different projects and for several customers. He is a software developer and architect with strong experience on the Windows platform, and in particular on the .NET Framework since Version 1.0, and on web technologies. He has always been working across a broad spectrum of responsibilities from distributed applications to complex user interfaces, from architecture and designing solutions to debugging server and client code, and from native Windows clients to web user interfaces.

He loves to learn, share, and communicate about technology. He wrote technical articles for an Italian magazine in the past, and now puts his effort into looking at the latest advances in programming or in contributing to open source projects such as ElmahR, a real-time error monitoring web dashboard built on top of SignalR. You can find out more about it at `http://elmahr.apphb.com/`.

Since 2011, he has been working in Switzerland where he lives with his wife, Cecilia, and their cats.

You can follow him on Twitter at `@wasp_twit`.

Roberto's blogs are at `http://www.robychechi.it/roby`.

Acknowledgments

I would like to thank Packt Publishing for the opportunity they gave me to write this book. This has been my first time as a writer and, for sure, it has been a challenging task, but their support has been constant and fundamental in making it possible. I would like to thank my colleague Duncan Moles and my old friend Emanuele "Lele" Rabino for accepting to review my first drafts. Their helpful advice comes from their skills and expertise. My gratitude goes also to Jean-Luc Marbot, Roberto Forno, Atif Aziz, and Gustavo Perez Leon. I learned so much from them in the last couple of years, and they have been very important to me during my adventure in Switzerland.

My biggest thanks go to my intercontinental family, without whom this book would not have been possible: my parents and brother in Italy, who have always supported and loved me; and my adoptive family in Paraguay, who received me as if I had always been one of them. And, of course I would like to thank my wife, Cecilia, who's at the center of my life: I love you.

About the Reviewers

Sriram Cherukumilli is a developer/architect with experience in architecture, design, and development across all application tiers with emphasis on enterprise real-time backend systems. He works at Argo Data Resources on Windows Workflow Foundation 4.5-based workflow management applications that abstract developer-specific and platform-specific details to present business-friendly orchestration services to build workflows across the company's different verticals. Sriram holds a Bachelor's degree in Engineering and a Master's degree in Information Technology. Before working at Argo, Sriram worked with many .NET SOA-based enterprise software systems at EMSI, Yahoo! Inc, and Verizon.

I would like to thank Packt Publishing for providing me with this opportunity.

Robin Karlsson is a Tech Lead and System Developer at Teleopti, with more than 10 years' experience of development in product companies in the HR area. The team at Teleopti successfully switched from a proprietary solution to a SignalR-based solution to enable monitoring the contact center performance in a web-based solution.

Duncan Mole is an experienced .NET architect/developer, and C# specialist with a focus on real-time, reactive programming. In recent years, Duncan has worked for a variety of investment banks and financial institutions, delivering solutions involving push-style messaging models on a variety of technologies such as SignalR.

Emanuele Rabino is a freelance developer specialized and passionate about everything related to the world of web development.

After working for many years on enterprise projects, using the ASP.NET stack in all its forms, he has been driving the development of e-commerce solutions using HTML5 and full-stack JavaScript environments.

Richard Seroter is the head of product management for CenturyLink Cloud, a Microsoft MVP, an instructor for the developer-centric training company Pluralsight, an InfoQ.com editor for cloud computing, and the author of multiple books on application integration strategies. He is a recognized public speaker and has spoken at events around the world. Richard maintains a regularly updated blog on the topics of architecture and solution design (`http://seroter.wordpress.com`), and can be found on Twitter as `@rseroter`.

www.PacktPub.com

Support files, eBooks, discount offers and more

You might want to visit `www.PacktPub.com` for support files and downloads related to your book.

Did you know that Packt offers eBook versions of every book published, with PDF and ePub files available? You can upgrade to the eBook version at `www.PacktPub.com` and as a print book customer, you are entitled to a discount on the eBook copy. Get in touch with us at `service@packtpub.com` for more details.

At `www.PacktPub.com`, you can also read a collection of free technical articles, sign up for a range of free newsletters and receive exclusive discounts and offers on Packt books and eBooks.

`http://PacktLib.PacktPub.com`

Do you need instant solutions to your IT questions? PacktLib is Packt's online digital book library. Here, you can access, read and search across Packt's entire library of books.

Why Subscribe?

- Fully searchable across every book published by Packt
- Copy and paste, print and bookmark content
- On demand and accessible via web browser

Free Access for Packt account holders

If you have an account with Packt at `www.PacktPub.com`, you can use this to access PacktLib today and view nine entirely free books. Simply use your login credentials for immediate access.

Table of Contents

Preface

The World Wide Web has been with us for the past 20 years and has become a fundamental part of our lives. Its distributed architecture has proven to be efficient and scalable. Thanks to it, nowadays, an incredible amount of information and services is available to all of us. We just have to connect, look for what we need, and pull it onto our devices to use it. However, it's also true that many scenarios would be more efficient if services themselves were able to determine the information that we need and then push it towards us at the right time. The contrast between these two ways of distributing content is clear and important, and according to the specific goals, there might be a clear advantage in using one or the other.

We already have several networking and application technologies that are ideal to build push systems, but the World Wide Web and its enabling protocol, HTTP, were not born for that. Traditional applications based on HTTP offer a request/response model where it's always the client's responsibility to initiate a connection, and it's always the client who has to ask the server for something. The server will send back the appropriate response on the same connection opened by the client to perform the request, and then, it will terminate the connection. According to this model, there is usually no natural way for the server to send any piece of information without a previous specific incoming request. Nevertheless, it would be a shame to miss the opportunity to leverage such a ubiquitous protocol in order to enable push scenarios for all its users.

This is how the Web has lately started to move towards enabling push scenarios, first with a series of technology tricks (Long Polling, Forever Frame, and Server-Sent Events) applied over the traditional HTTP/HTML stack, and then with the rise of a proper technology that introduces a way to establish persistent connections between clients and servers, which can then be used for fully bidirectional communications: WebSocket. Modern browsers and web servers bring full support to the latter option, while older systems can recur to the former tricks. So, now we have several ways to deliver a solution, but also the problem of having to decide which technology to use, or maybe the need to replicate our solution using all these techniques together to reach every potential user.

Enter SignalR! SignalR is a very interesting library that leverages all the strategies that we previously mentioned to deliver a real-time push platform. It enables a two-way communication model between the client and the server, and it achieves this goal simply by leveraging what HTTP and HTML5 have to offer. SignalR looks like magic because it transparently adapts itself to the available environment (the HTTP server and the web browser) and transforms a normally transient HTTP connection into a virtually persistent connection. The messaging API offered by SignalR succeeds in abstracting the low-level networking strategies, chosen according to what's supported by the involved counterparts, and offers us a simple and generic way to write code that remains unaware of the underlying complexity.

We briefly mentioned what SignalR does and how it does it, but the actual goal of this book is not to dig deep into the mechanics SignalR is built on top of. This is a practical, hands-on guide that provides you with a number of clear, step-by-step recipes that will gradually enable you to add SignalR as an innovative, effective, and useful item in your toolbox. It will move from simple examples down to complex use cases, going through a comprehensive overview of the library.

Although most of the recipes will give some information on how SignalR works behind the scenes to enable the proposed solution, these explanations shall not be too detailed. You should not expect otherwise from this book. The book will not go into deep architectural details. It will just provide you with a decent level of explanations to help the reader understand what is going on, while keeping the focus on bringing practical and synthetic solutions to specific questions.

In each recipe, we will be picking a problem and showing you a SignalR-based solution, or, if you prefer, we'll be choosing a specific feature from SignalR and matching it to the class of scenarios it helps tackle. This way you will gradually learn how to perform a set of common tasks, which the last chapter will combine to build complex applications.

At the time of writing this book, SignalR reached Version 2, and this is the one that we'll be using for our discussion. If you need to use Version 1, this book could still be used as a general reference. However, there are some differences that you would have to take care of, especially in areas related to hosting and bootstrapping a SignalR-based application. That said, these differences will not be treated throughout the text, and no particular attention will be paid to the older version.

You might also have to pay attention to the fast evolution of the minor version number of SignalR and of all its dependencies. SignalR is available on NuGet. It's constantly updated, and the same happens to the components that it depends on, such as jQuery or Newtonsoft.Json. This means that the actual version numbers that you might reference while writing your code are likely to be different from the ones you will find listed here. The recipes have been constantly revised, and have been updated to what's available at the time of the final technical reviews (February, 2014). You will have to take care of any further update that might be released later and act manually to fix any mismatch. This will probably result in having to change some JavaScript reference to a later version, or to add some `assemblyRedirect` directive in your configuration files to remap an older version of a required assembly to a newer one. Once done with that, the code will still be valid and fully working.

It's worth mentioning that SignalR is an open source project whose source code can be found at `https://github.com/SignalR/SignalR`.

Whenever there's anything unclear and you really need to shed some light on it, you can inspect its code and find the answers by yourself. It's a very clean and well-organized code base, and you should not get scared by the idea of going through it. The official documentation can be found at `http://www.asp.net/signalr`.

What this book covers

Chapter 1, *Understanding the Basics*, covers the basic steps to add the server and client portions of a SignalR application in the context of different hosting technologies. We will be writing the simplest code possible, and we'll perform the minimal steps that are required to have everything up and running.

Chapter 2, *Using Hubs*, illustrates the Hubs API from a server-side point of view.

Chapter 3, *Using the JavaScript Hubs Client API*, introduces the Hubs API from a client-side point of view, using the JavaScript client library.

Chapter 4, *Using the .NET Hubs Client API*, explains the Hubs API from a client-side point of view, using the .NET client library this time.

Chapter 5, *Using a Persistent Connection*, moves to the more low-level persistent connection API, illustrating its peculiar features and differences when compared to Hubs.

Chapter 6, *Handling Connections*, illustrates some advanced features that we can leverage to optimize and customize the way we handle the existing connections.

Chapter 7, *Analyzing Advanced Scenarios*, digs into more infrastructural features made available to fine-tune and extend SignalR's behaviors.

Chapter 8, *Building Complex Applications*, is all about full-fledged examples, illustrating how SignalR can be used as the foundation technology to solve real-world, bidirectional messaging problems.

Appendix A, *Creating Web Projects*, explains the steps to create each one of the various types of ASP.NET projects that we created in Visual Studio, in case you are not yet used to it.

Appendix B, *Insights*, discusses the different transport strategies that SignalR chooses to provide a logical persistent connection, according to the environment it runs on. It also talks about the basic concepts of asynchronous programming.

What you need for this book

All the code samples have been prepared and tested using Microsoft Visual Studio 2013, which brings the highest integration with Version 2 of SignalR with it. Microsoft Visual Studio 2012 could be used too, and you would be able to reach the same final result, but the experience inside the IDE might be slightly different. Again, the book will not try to fill any gap between the two environments, and it will explicitly only target the 2013 version.

Who this book is for

This book can be read by different types of developers.

Beginners will be able to learn all the fundamental concepts of SignalR, quickly becoming productive in a usually difficult arena that real-time, bidirectional communication normally is.

In this book, experienced programmers will find a handy and useful collection of ready-made solutions to common use cases, which they will then be able to enhance as needed. They will be able to use it as a quick reference to the most important SignalR features. No previous practical experience either in SignalR or real-time communication in general is required.

Conventions

In this book, you will find a number of styles of text that distinguish between different kinds of information. Here are some examples of these styles, and an explanation of their meaning.

Code words in text, database table names, folder names, filenames, file extensions, pathnames, dummy URLs, user input, and Twitter handles are shown as follows:
"To add a friendly name, we can use the `HubName` attribute."

A block of code is set as follows:

```
public class Startup
{
    public void Configuration(IAppBuilder app)
    {
        app.MapSignalR();
    }
}
```

When we wish to draw your attention to a particular part of a code block, the relevant lines or items are set in bold:

```
public class Startup
{
    public void Configuration(IAppBuilder app)
    {
        app.MapSignalR();
    }
}
```

Any command-line input or output is written as follows:

```
signalr.exe ghp
```

New terms and **important words** are shown in bold. Words that you see on the screen, in menus or dialog boxes for example, appear in the text like this: "Clicking on the **Ok** button creates a new file."

Warnings or important notes appear in a box like this.

Tips and tricks appear like this.

Reader feedback

Feedback from our readers is always welcome. Let us know what you think about this book—what you liked or may have disliked. Reader feedback is important for us to develop titles that you really get the most out of.

To send us general feedback, simply send an e-mail to `feedback@packtpub.com`, and mention the book title via the subject of your message.

If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, see our author guide on `www.packtpub.com/authors`.

Customer support

Now that you are the proud owner of a Packt book, we have a number of things to help you to get the most from your purchase.

Downloading the example code

You can download the example code files for all Packt books you have purchased from your account at `http://www.packtpub.com`. If you purchased this book elsewhere, you can visit `http://www.packtpub.com/support` and register to have the files e-mailed directly to you.

Errata

Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you find a mistake in one of our books—maybe a mistake in the text or the code—we would be grateful if you would report this to us. By doing so, you can save other readers from frustration and help us improve subsequent versions of this book. If you find any errata, please report them by visiting `http://www.packtpub.com/submit-errata`, selecting your book, clicking on the **errata submission form** link, and entering the details of your errata. Once your errata are verified, your submission will be accepted and the errata will be uploaded on our website, or added to any list of existing errata, under the Errata section of that title. Any existing errata can be viewed by selecting your title from `http://www.packtpub.com/support`.

Piracy

Piracy of copyright material on the Internet is an ongoing problem across all media. At Packt, we take the protection of our copyright and licenses very seriously. If you come across any illegal copies of our works, in any form, on the Internet, please provide us with the location address or website name immediately so that we can pursue a remedy.

Please contact us at `copyright@packtpub.com` with a link to the suspected pirated material.

We appreciate your help in protecting our authors, and our ability to bring you valuable content.

Questions

You can contact us at `questions@packtpub.com` if you are having a problem with any aspect of the book, and we will do our best to address it.

1
Understanding the Basics

In this chapter, we will cover:

- Adding a Hub to an ASP.NET project
- Adding a Hub to a self-hosting application
- Connecting to a Hub from a JavaScript client
- Connecting to a Hub from a .NET application

Introduction

SignalR is an amazing framework that delivers a real-time and bidirectional messaging platform. SignalR provides several options to reach its goal, but in this chapter we'll start simple and use the most basic API to set up a persistent and real-time channel: Hubs. A Hub is a special class that SignalR will expose to all the connected clients, allowing them to make **Remote Procedure Calls** (**RPC**) to it. Inside the Hub, the developer will also have a set of special objects to use in order to perform calls back onto the connected clients.

There is a very important detail to highlight: SignalR is composed of a server-side library and a set of client-side libraries. In every working solution, you will always need to use both; you will need to expose the server-side endpoints and connect to them using the most appropriate client library. SignalR will do the rest, and you will experience a very natural, simple, and bidirectional programming model.

All the recipes in this chapter will be classic "Hello World" applications. Nothing fancy or exciting will be happening, but all of them will clearly illustrate what can be achieved and how. The *Adding a Hub to an ASP.NET project* and *Adding a Hub to a self-hosting application* recipes will show you how to prepare a server portion of a SignalR application using the Hub type in different hosting contexts, whereas the *Connecting to a Hub from a JavaScript client* and *Connecting to a Hub from a .NET application* recipes will illustrate how to write client-side code to connect to it from different types of client processes. Each recipe has the goal to be fully functional, therefore all of them will in some way provide at least some hints about the missing counterparts. Server-side recipes will have minimal client code in place, and client-side ones will either contain a basic Hub to connect to or refer to one created earlier, but for all of them, the focus will remain on the actual topic of the recipe.

Adding a Hub to an ASP.NET project

SignalR sets a clear separation between the actual messaging runtime and the hosting environment. Although the host could be any plain old .NET-based process, the most natural context where you can add a SignalR Hub is inside an ASP.NET project, which is the topic of this recipe. Later in this chapter, we'll see how to host it in a different context.

This recipe will concentrate on the server-side; however, some minimal client-side code will also be added to be able to fully demonstrate a complete, although trivial, client-server connection.

Getting ready

There are three main types of ASP.NET projects:

- A Web Forms application
- An MVC application
- A website

The process of creating them is a fairly common task, so we are going to skip the details. If you want more information, you can refer to the *Appendix A*, *Creating Web Projects* at the end of the book and check how to generate them step by step. In this recipe, we will be covering all of them at once, highlighting the points where there's some difference across those types.

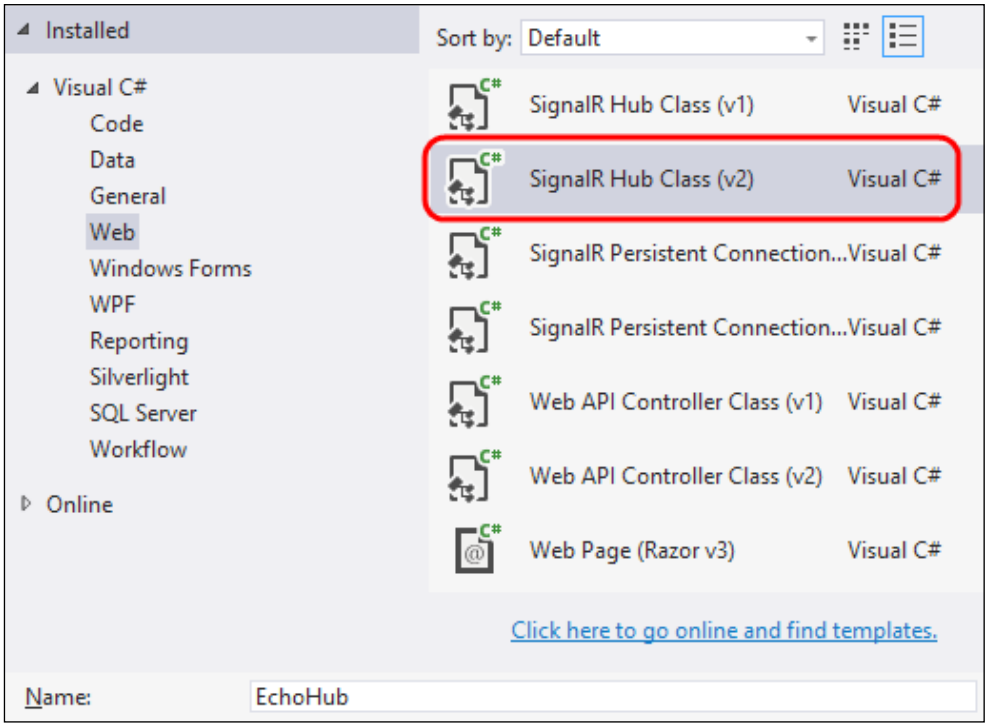
In order to show a complete sample for all three cases, the code that comes with this book will contain three separate projects, called `Recipe01_WF` (for the Web Forms sample), `Recipe01_MVC` (for the MVC project), and `Recipe01_WS` (for the website).

Before proceeding, please pick one of them and create your project in Visual Studio 2013.

How to do it...

We're ready to actually start adding the SignalR bits. Let's start with the Hub with the following steps.

From the **Project** menu, select **Add New Item** (you can also use the project context menu from **Solution Explorer** or the *Ctrl* + *Shift* + *A* keyboard shortcut), click on the **Web** folder, and then select the **SignalR Hub Class (v2)** template; specify `EchoHub` as the name and click on **OK** as shown in the following screenshot. Make sure you have selected the **v2** Version because we want to target **SignalR 2.0**.



Visual Studio will add a new file called `EchoHub.cs` with some boilerplate code inside.

1. Let's edit the file content to make it look like the following code snippet:

```
using System.Diagnostics;
using Microsoft.AspNet.SignalR;
using Microsoft.AspNet.SignalR.Hubs;
namespace Recipe01
```

```
{
    [HubName("echo")]
    public class EchoHub : Hub
    {
        public void Say(string message)
        {
            Trace.WriteLine(message);
        }
    }
}
```

Downloading the example code

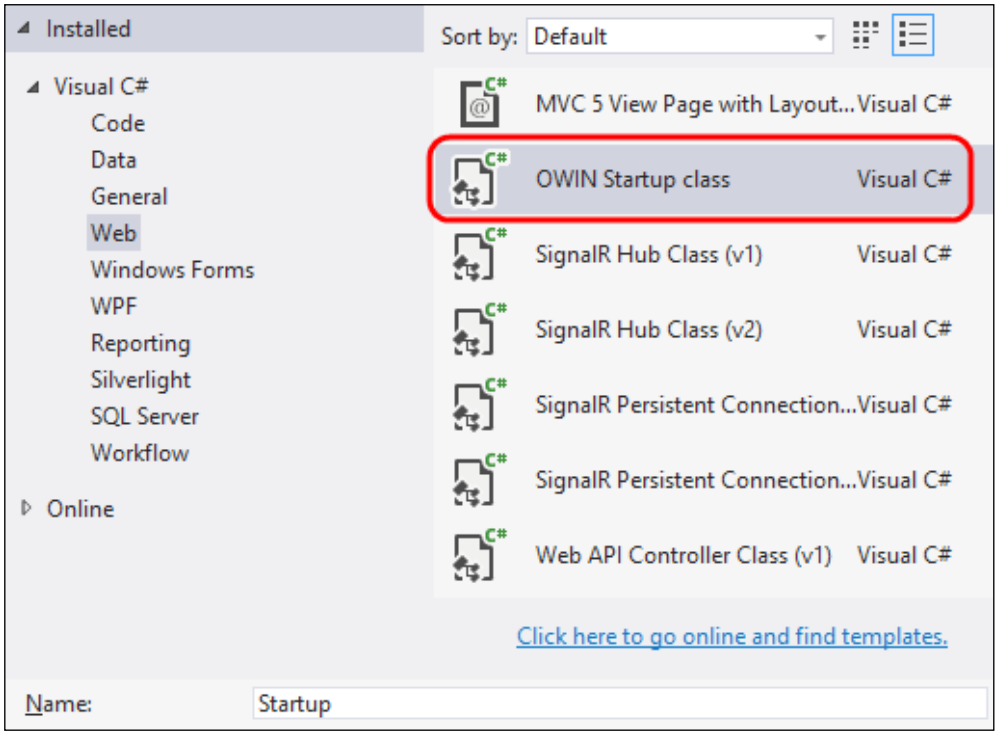
You can download the example code files for all Packt books you have purchased from your account at `http://www.packtpub.com`. If you purchased this book elsewhere, you can visit `http://www.packtpub.com/support` and register to have the files e-mailed directly to you.

The following lists the important points here:

- The necessary `using` directives are listed at the top of the file.
- The `EchoHub` class is derived from `Hub`, which comes from `Microsoft.AspNet.SignalR.Hubs` and makes the server-side SignalR API available to our class.
- The class is marked with the `HubName` attribute, which allows us to give the Hub a friendly name to be used by the clients; if we don't use the `HubName` attribute, the Hub name will be the same as the class name (in this case, it would be `EchoHub`).
- Our Hub contains a method called `Say()`. This is just a sample method we'll use to show how to expose Hub endpoints. On every call, it will just output the value of the `message` parameter in the debugger **Output** window, or in any **trace listener** we may want to configure.

The class namespace is not so important. Here, I'm choosing the same name as the project name; it's the recommended way, but it does not have to be like that.

2. From the **Project** menu, select **Add New Item** again, click on the **Web** folder, and then select the **OWIN Startup class** template. Specify `Startup` as the name and click on **OK**, as shown in the following screenshot:



Visual Studio will add a new file called `Startup.cs` with some code inside it.

3. Let's edit the file content to make it look like the following:

```
using Microsoft.Owin;
using Owin;
[assembly: OwinStartup(typeof(Recipe01.Startup))]
namespace Recipe01
{
    public class Startup
    {
        public void Configuration(IAppBuilder app)
        {
            app.MapSignalR();
        }
    }
}
```