SOAP web service tutorial for C#

Dr. Balázs Simon (sbalazs@iit.bme.hu), BME IIT, 2023

1 Introduction

This document describes how to create a Hello World web service in Visual Studio using ASP.NET core and SoapCore.

2 Code-first approach

Although, it is not recommended in a production environment, the easiest way to create a web service is to start with the C# code and publish the implementation as a web service.

It is advisable to choose a directory dedicated to your Visual Studio projects in the file system. The whole path of this directory should not contain any spaces or special characters. For example, you can choose **c:\Users\[user]\source\projects** where **[user]** is your Windows user name. In case this path contains special characters, choose a different directory.

Open a command prompt from your project directory and issue the following commands to create a new web application (the ↳ symbol means that the command is continued in the next line, and the whole command should be typed as a single line):

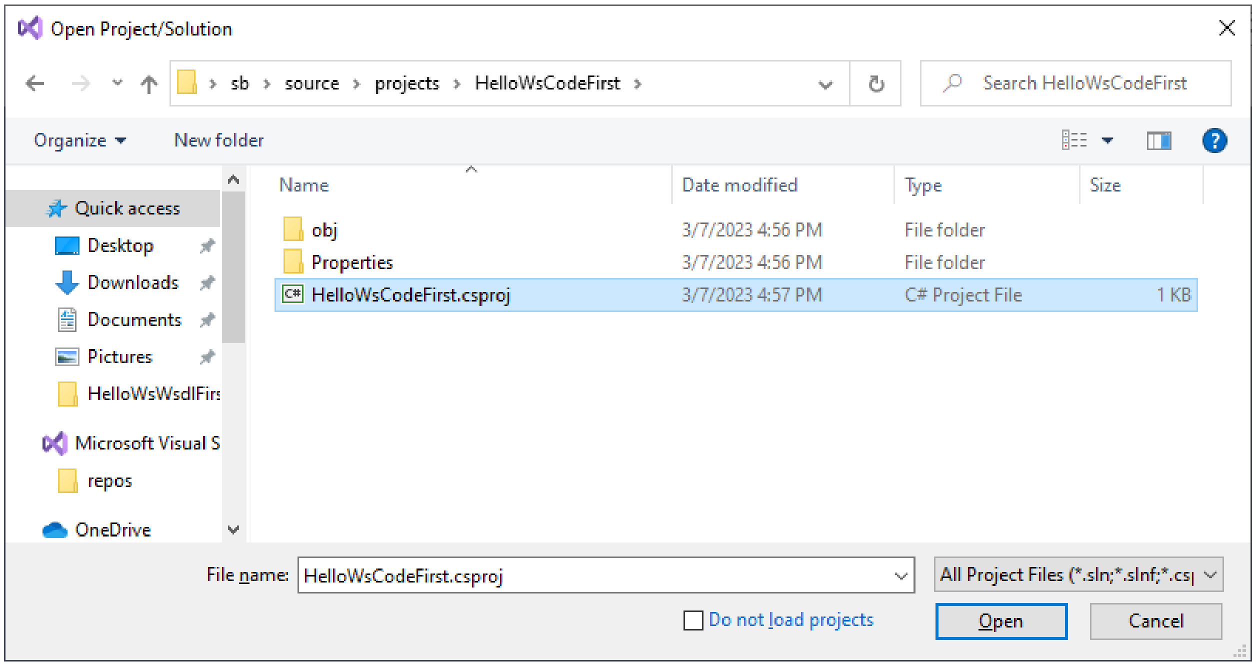
```
c:\Users\[user]\source\projects>mkdir HelloWsCodeFirst
```

```
c:\Users\[user]\source\projects>dotnet new web -n HelloWsCodeFirst ↳
  -o HelloWsCodeFirst
```

(Make sure to use the simple command prompt and not PowerShell, since PowerShell handles the dash switches differently.)

Replace the **`HelloWsCodeFirst`.csproj** in the newly created **HelloWsCodeFirst** folder with the one attached to this tutorial (**HelloWsCodeFirst\HelloWsCodeFirst.csproj**).

In Visual Studio, click on the menu **File > Open > Project/Solution...** and select the project file:

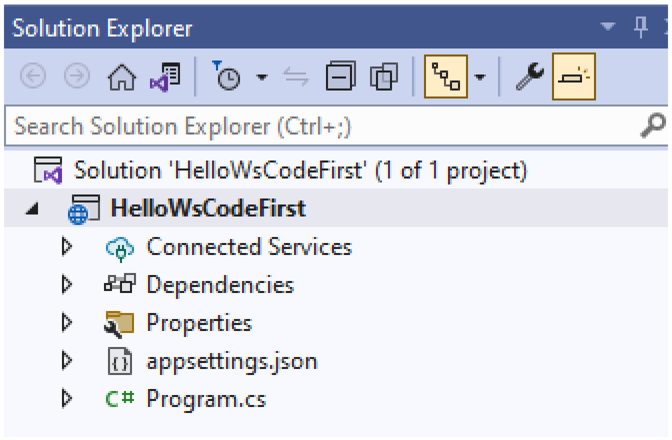


Name	Date modified	Type	Size
obj	3/7/2023 4:56 PM	File folder	
Properties	3/7/2023 4:56 PM	File folder	
HelloWsCodeFirst.csproj	3/7/2023 4:57 PM	C# Project File	1 KB

File name: HelloWsCodeFirst.csproj

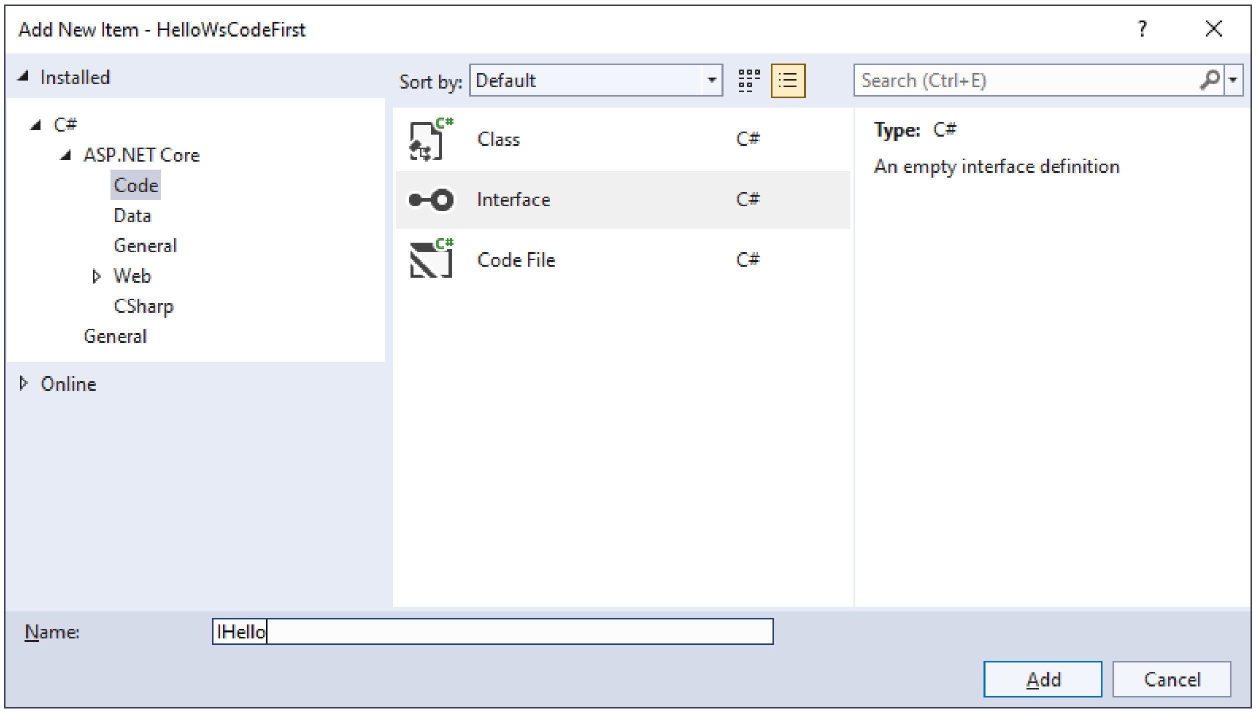
All Project Files (*.sln;*.slnf;*.cs|

Click **Open**, and the project should look like this:



Solution 'HelloWsCodeFirst' (1 of 1 project)
- **HelloWsCodeFirst**
  - Connected Services
  - Dependencies
  - Properties
  - appsettings.json
  - Program.cs

Right click on the **HelloWsCodeFirst** project and select **Add > Class**. Select **Interface** and specify **IHello** as a name:



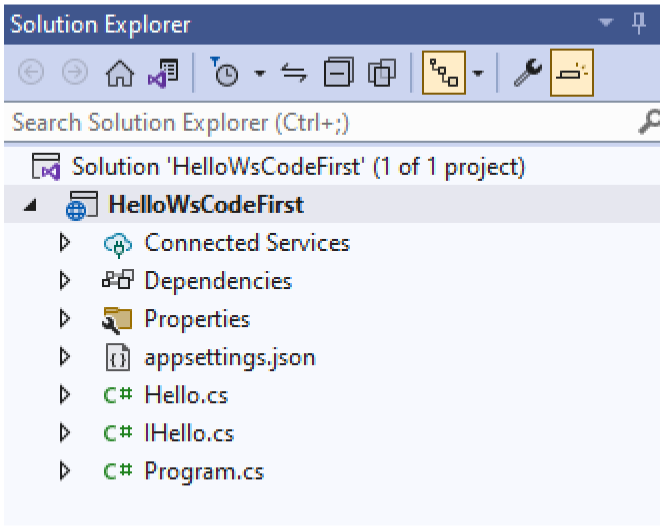
Type: C#
An empty interface definition

Name: IHello

Click **Add** to add the **IHello** interface it to the project.

Now, right click on the **HelloWsCodeFirst** project and select **Add > Class**. Select **Class** and specify **Hello** as a name. Click **Add** to add the **Hello** class to the project.

The project should look like this:



```
Solution 'HelloWsCodeFirst' (1 of 1 project)
  HelloWsCodeFirst
    Connected Services
    Dependencies
    Properties
    appsettings.json
    Hello.cs
    IHello.cs
    Program.cs
```

The **IHello.cs** should contain the following code:

```
using System.ServiceModel;

namespace HelloWsCodeFirst
{
    [ServiceContract]
    public interface IHello
    {
        [OperationContract]
        string SayHello(string name);
    }
}
```

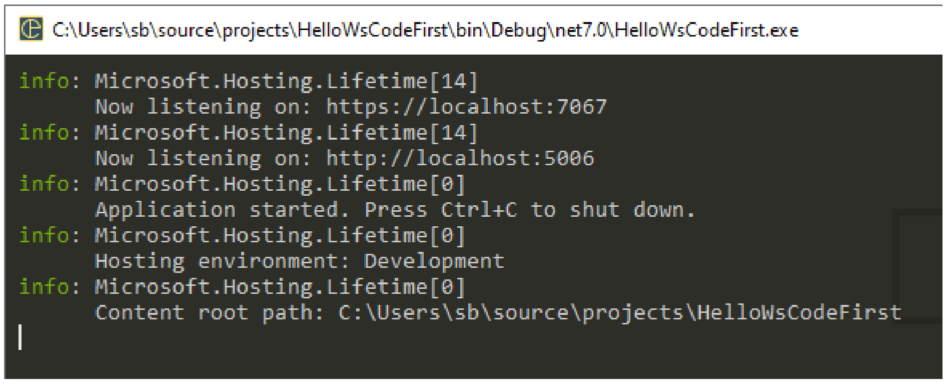
The **Hello.java** should contain the following code:

```
namespace HelloWsCodeFirst
{
    public class Hello : IHello
    {
        public string SayHello(string name)
        {
            return $"Hello: {name}";
        }
    }
}
```

Update the **Program.cs** file with the following content:

```
using HelloWsCodeFirst;
using Microsoft.Extensions.DependencyInjection.Extensions;
using SoapCore;

var builder = WebApplication.CreateBuilder(args);

// Register the SoapCore library:
builder.Services.AddSoapCore();
// Register our Hello service implementing the IHello interface:
builder.Services.TryAddSingleton<IHello, Hello>();

var app = builder.Build();

// Specify the endpoint of our Hello service:
app.UseRouting();
app.UseEndpoints(endpoints => {
    endpoints.UseSoapEndpoint<IHello>("/HelloService", new SoapEncoderOptions(),
SoapSerializer.DataContractSerializer);
});

app.Run();
```

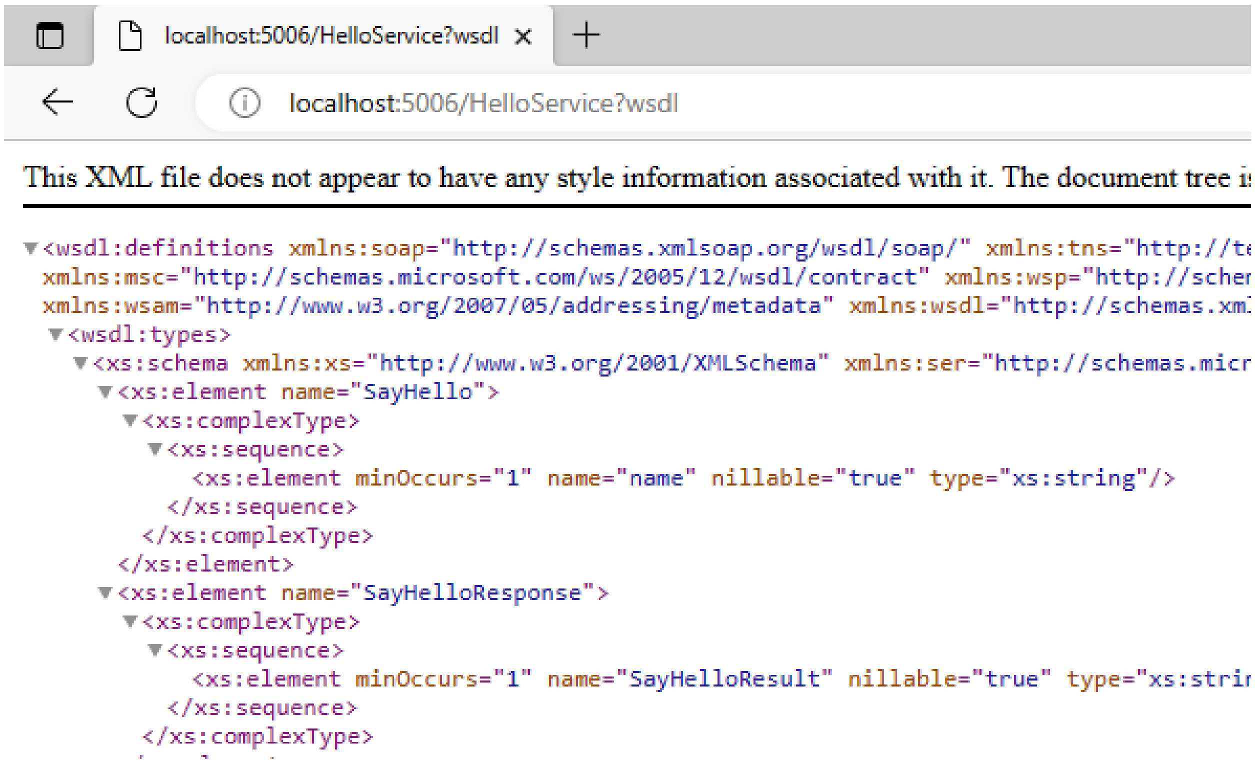
Press **Ctrl+F5** to run the project. It should start with a similar log written to the command line:



```
C:\Users\sb\source\projects\HelloWsCodeFirst\bin\Debug\net7.0\HelloWsCodeFirst.exe

info: Microsoft.Hosting.Lifetime[14]
      Now listening on: https://localhost:7067
info: Microsoft.Hosting.Lifetime[14]
      Now listening on: http://localhost:5006
info: Microsoft.Hosting.Lifetime[0]
      Application started. Press Ctrl+C to shut down.
info: Microsoft.Hosting.Lifetime[0]
      Hosting environment: Development
info: Microsoft.Hosting.Lifetime[0]
      Content root path: C:\Users\sb\source\projects\HelloWsCodeFirst
```

Based on the port number in the log, type the appropriate URL in a browser by appending **?wsdl** at the end to see the WSDL generated for the service. In the example above it is either **http://localhost:5006/HelloService?wsdl** or **https://localhost:7067/HelloService?wsdl**

For example:



```
localhost:5006/HelloService?wsdl

This XML file does not appear to have any style information associated with it. The document tree i

▼<wsdl:definitions xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:tns="http://te
 xmlns:msc="http://schemas.microsoft.com/ws/2005/12/wsdl/contract" xmlns:wsp="http://scher
 xmlns:wsam="http://www.w3.org/2007/05/addressing/metadata" xmlns:wsdl="http://schemas.xml
  ▼<wsdl:types>
    ▼<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:ser="http://schemas.micr
      ▼<xs:element name="SayHello">
        ▼<xs:complexType>
          ▼<xs:sequence>
              <xs:element minOccurs="1" name="name" nillable="true" type="xs:string"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      ▼<xs:element name="SayHelloResponse">
        ▼<xs:complexType>
          ▼<xs:sequence>
              <xs:element minOccurs="1" name="SayHelloResult" nillable="true" type="xs:strir
            </xs:sequence>
          </xs:complexType>
```

(Note that the port numbers are also available in the **launchSettings.json** file under the **Properties** folder inside the project.)

You can stop the application by pressing **Ctrl+C** in the command line window.

3 WSDL-first approach

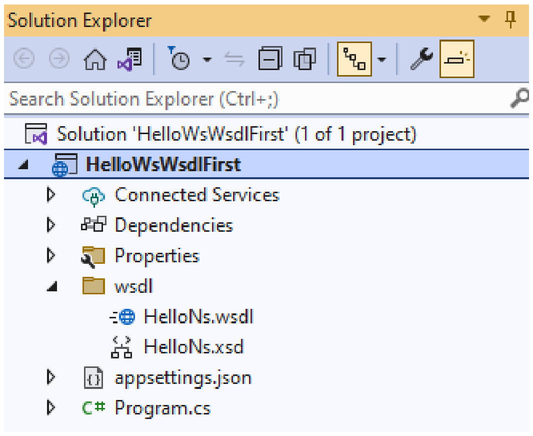
The recommended approach for creating web services is to start from the WSDL interface, because the WSDL description is stable (the service implementation does not affect it).

We need the **svcutil** tool to generate C# code from the WSDL description. Install this tool with the following command issued from any directory using the command prompt:

```
dotnet tool install --global dotnet-svcutil
```

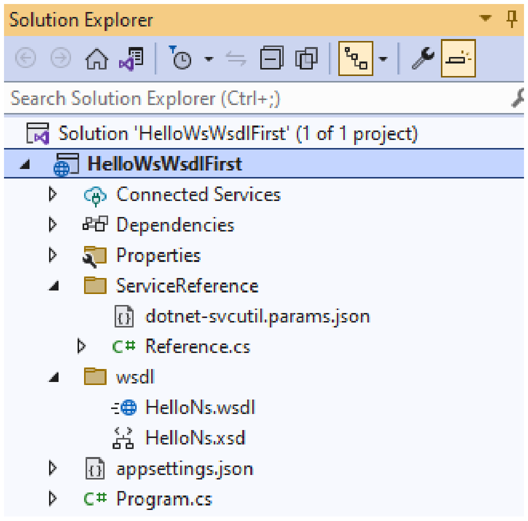
Similarly to the previous project, create a new **web project** called **HelloWsWsdlFirst**. Use the attached **HelloWsWsdlFirst/HelloWsWsdlFirst.csproj** as project configuration. Create a folder called **wsdl** inside the project, and copy the **HelloNs.wsdl** and **HelloNs.xsd** into this folder. Open the project in Visual Studio.

The project should look like this:



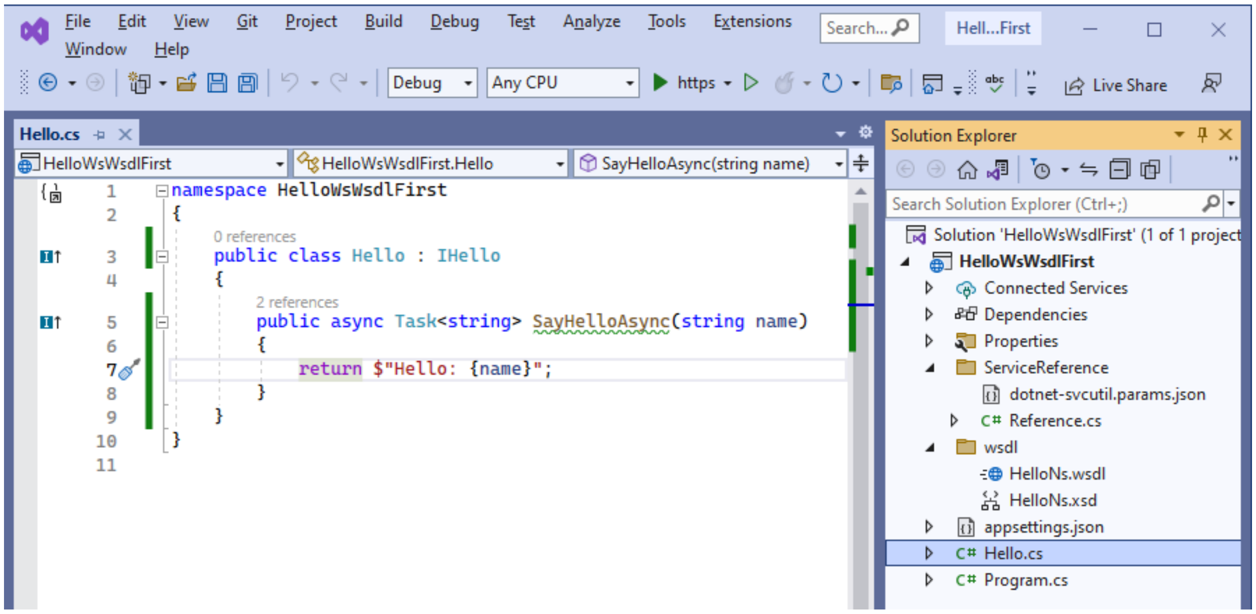
```
Solution 'HelloWsWsdlFirst' (1 of 1 project)
  HelloWsWsdlFirst
    Connected Services
    Dependencies
    Properties
    wsdl
      HelloNs.wsdl
      HelloNs.xsd
    appsettings.json
    Program.cs
```

Press **Ctrl+Shift+B** to build the project. This should generate the C# code from the WSDL into the **ServiceReference** folder:



```
Solution 'HelloWsWsdlFirst' (1 of 1 project)
  HelloWsWsdlFirst
    Connected Services
    Dependencies
    Properties
    ServiceReference
      dotnet-svcutil.params.json
      Reference.cs
    wsdl
      HelloNs.wsdl
      HelloNs.xsd
    appsettings.json
    Program.cs
```

Add a new class called **Hello** to the project with the following content:


File Edit View Git Project Build Debug Test Analyze Tools Extensions Search... Hell...First
Window Help
Debug Any CPU https Live Share
Hello.cs
HelloWsWsdlFirst HelloWsWsdlFirst.Hello SayHelloAsync(string name)
Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'HelloWsWsdlFirst' (1 of 1 project
HelloWsWsdlFirst
Connected Services
Dependencies
Properties
ServiceReference
dotnet-svcutil.params.json
Reference.cs
wsdl
HelloNs.wsdl
HelloNs.xsd
appsettings.json
Hello.cs
Program.cs

```
namespace HelloWsWsdlFirst
{
    0 references
    public class Hello : IHello
    {
        2 references
        public async Task<string> SayHelloAsync(string name)
        {
            return $"Hello: {name}";
        }
    }
}
```

This class implements the **IHello** interface generated from the WSDL.

Update the **Program.cs** with the following content:

```
using HelloWsWsdlFirst;
using Microsoft.Extensions.DependencyInjection.Extensions;
using SoapCore;

var builder = WebApplication.CreateBuilder(args);

// Register the SoapCore library:
builder.Services.AddSoapCore();
// Register our Hello service implementing the IHello interface:
builder.Services.TryAddSingleton<IHello, Hello>();

var app = builder.Build();

// Specify the endpoint of our Hello service:
app.UseRouting();
app.UseEndpoints(endpoints => {
    endpoints.UseSoapEndpoint<IHello>("/HelloService", new SoapEncoderOptions(),
SoapSerializer.DataContractSerializer);
});

app.Run();
```

Press **Ctrl+F5** to run the project. You can access the WSDL in the browser similarly to the one in the previous project.

You can stop the application by pressing **Ctrl+C** in the command line window.

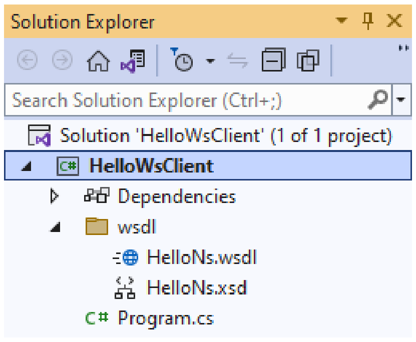
4 Client for the service

Open a command prompt from your project directory and issue the following commands to create a new console application:

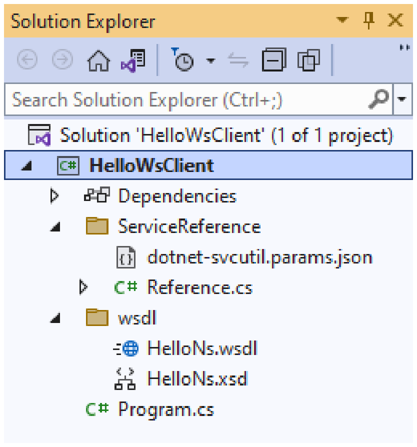
```
c:\Users\[user]\source\projects>mkdir HelloWsClient

c:\Users\[user]\source\projects>dotnet new web -n HelloWsClient ↪
  -o HelloWsClient
```

Replace the **HelloWsClient.csproj** in the newly created **HelloWsClient** folder with the one attached to this tutorial (**HelloWsClient\HelloWsClient.csproj**). Create a folder called **wsdl** inside the project, and copy the **HelloNs.wsdl** and **HelloNs.xsd** into this folder. Open the project in Visual Studio.

The project should look like this:



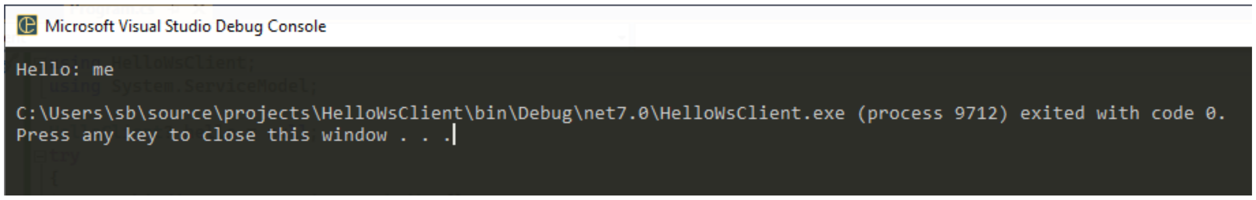
Press **Ctrl+Shift+B** to build the project. This should generate the C# code from the WSDL into the **ServiceReference** folder:



Update the **Program.cs** with the following content (make sure that the endpoint address refers to the same port number as the one used by the **HelloWsWsdlFirst** application):

```
using HelloWsClient;
using System.ServiceModel;

HelloClient? hello = null;
try
{
  var binding = new BasicHttpBinding();
  var endpoint = new EndpointAddress(new Uri("http://localhost:5006/HelloService"));

  hello = new HelloClient(binding, endpoint);
  var response = await hello.SayHelloAsync("me");
  Console.WriteLine(response);
}
finally
{
  if (hello is not null) await hello.CloseAsync();
}
```

Make sure that the **HelloWsWsdlFirst** application is running, and then run the **HelloWsClient** application by pressing **Ctrl+F5**. The result should be the following:



```
Hello: me

C:\Users\sb\source\projects\HelloWsClient\bin\Debug\net7.0\HelloWsClient.exe (process 9712) exited with code 0.
Press any key to close this window . . .
```

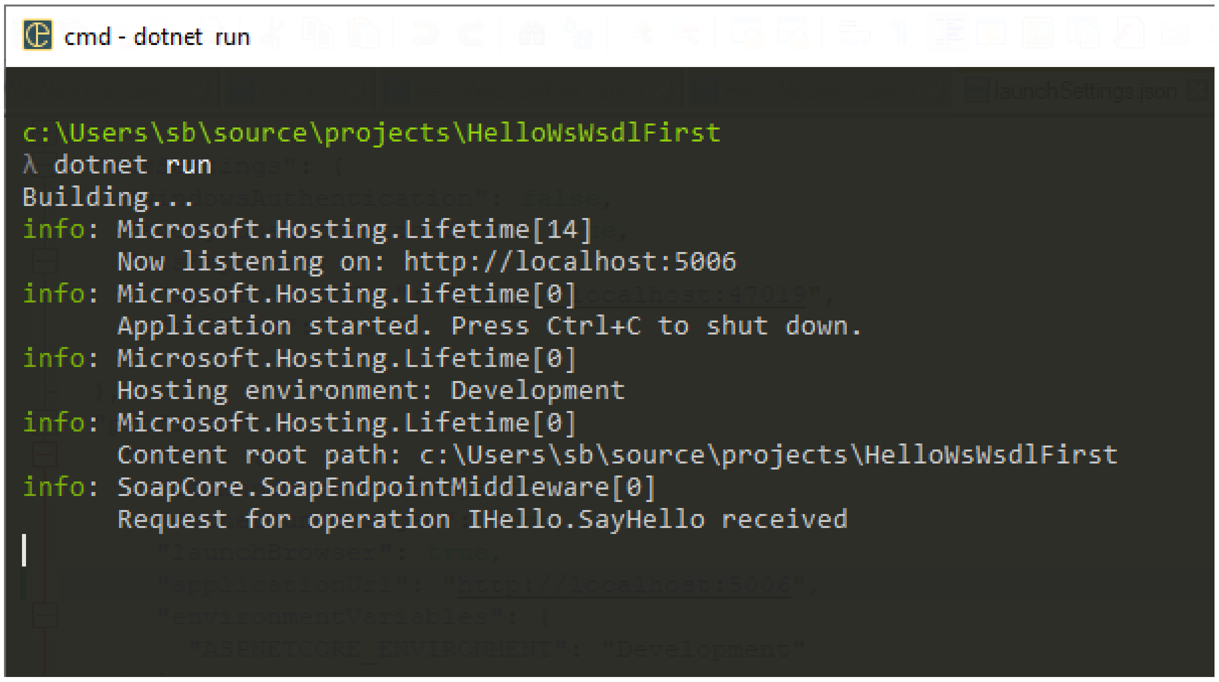
5 Building and running from the command line

You can build the projects created in the previous chapters from the command line if you enter the project's directory and issue the **dotnet build** command. Examples:

```
c:\Users\[user]\source\projects\HelloWsCodeFirst>dotnet build

c:\Users\[user]\source\projects\HelloWsWsdlFirst>dotnet build

c:\Users\[user]\source\projects\HelloWsClient>dotnet build
```

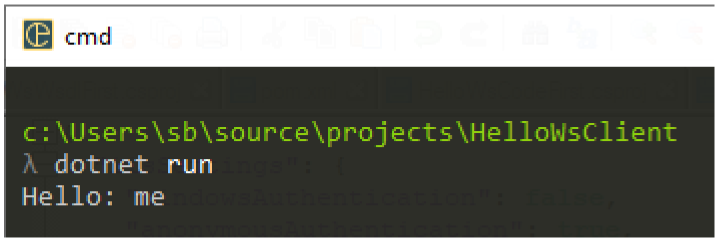
You can run the projects created in the previous chapters from the command line if you enter the project's directory and issue the **dotnet run** command.

Run the **HelloWsWsdlFirst** server application from one command line window (make sure it is not running elsewhere already):



```
c:\Users\sb\source\projects\HelloWsWsdlFirst
λ dotnet run
Building...
info: Microsoft.Hosting.Lifetime[14]
      Now listening on: http://localhost:5006
info: Microsoft.Hosting.Lifetime[0]
      Application started. Press Ctrl+C to shut down.
info: Microsoft.Hosting.Lifetime[0]
      Hosting environment: Development
info: Microsoft.Hosting.Lifetime[0]
      Content root path: c:\Users\sb\source\projects\HelloWsWsdlFirst
info: SoapCore.SoapEndpointMiddleware[0]
      Request for operation IHello.SayHello received
```

Run the **HelloWsClient** server application from another command line window:



```
c:\Users\sb\source\projects\HelloWsClient
λ dotnet run
Hello: me
```

Go back to the **HelloWsWsdlFirst** server console and stop the server by pressing **Ctrl+C**.