Exercise 4.1

WCF – basics and service configuration

Author: Mariusz Fraś

1 Objectives of the exercise

The purpose of the exercise is:

1. Understanding the basic architecture of the WCF application..
2. Getting acquainted with the basics of creating a WCF service and client service
3. Understanding the options for configuring services - endpoints, transport, the service behavior.

- **The first part of the exercise** is to be carried out according to the given instruction and possible class teacher’s additional instructions.
- **The second part of the exercise** is to prepare and present or to complete **on the next class** according to teacher’s orders.

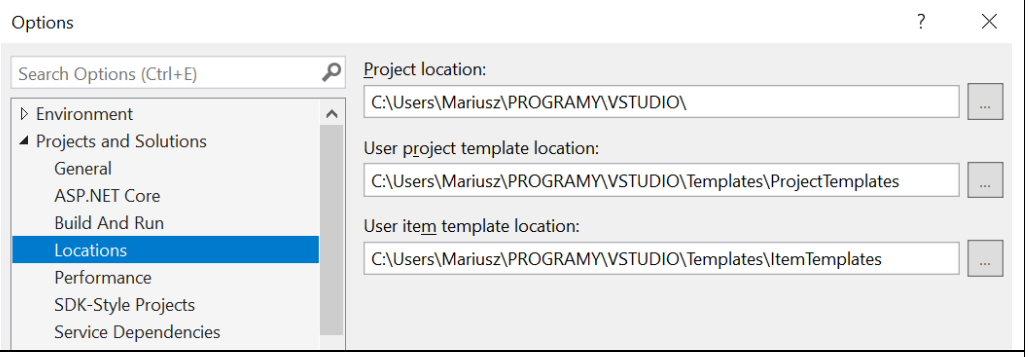
2 Exercise – Part I The basics of creating WCF application

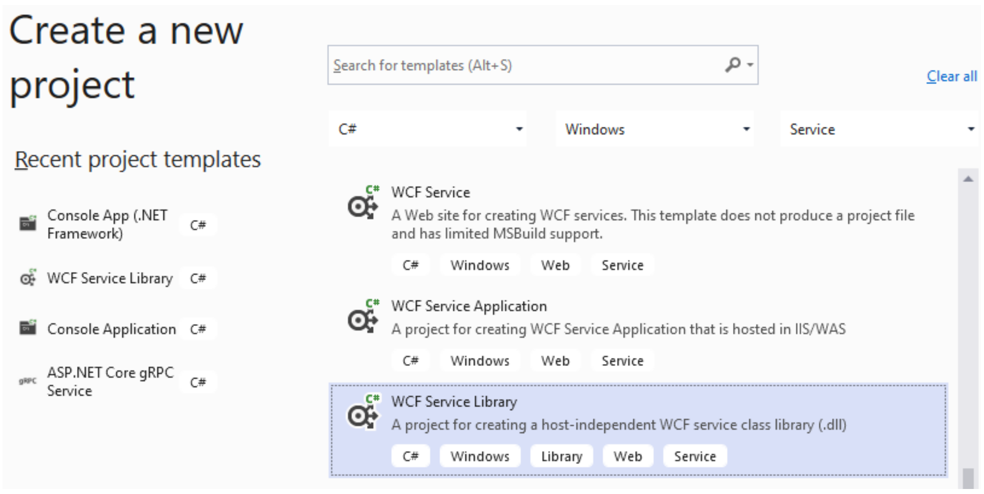
The exercise will include a solution including: **a)** a service running as a separate console application and **b)** a client using this service. The service and the client will be developed with use of Visual Studio (VS for short). The task of implementing the client-server WCF application consists of several stages:

1. Defining a service contract.
2. Implementation of the service contract.
3. Creating an application hosting the service
   - here: it is a console application – this approach is so-called self hosting service.
4. Creating a client application with a client proxy (WCF proxy client) component.
5. Configuration of the client.
6. Implementation of the client application.

ATTENTION: ***any changes*** *(e.g. class names, etc.)* ***in the initially generated*** *by the platform* ***code*** *should be implemented* ***using platform refactoring options***.*.*

- *In VStudio 2013, the* ***Refactor option*** *in the context menu (after clicking the right mouse button.*
- *In VStudio 2015÷2019 the* ***appropriate option*** *in the context menu (eg Rename) - there is no Refactor option specified.*

1. Initial preparation of the environment	• Prepare a working directory in which the WCF application will be created. • Run the Visual Studio platform. • Check and if applicable set up the folders in which projects will be created in the option: **TOOLS→Options…→Projects and Solutions→Locations**. (or **General** in earlier versions of VS).  Options Search Options (Ctrl+E) Environment Projects and Solutions General ASP.NET Core Build And Run Locations Performance SDK-Style Projects Service Dependencies Project location: C:\Users\Mariusz\PROGRAMY\VSTUDIO\ User project template location: C:\Users\Mariusz\PROGRAMY\VSTUDIO\Templates\ProjectTemplates User item template location: C:\Users\Mariusz\PROGRAMY\VSTUDIO\Templates\ItemTemplates
2. Defining a WCF service contract	• Create a new solution and application project with use of **Visual C# WCF Service Library** project template giving own solution name and project name (here: Exercise4 i WcfService1).


Create a new project

Recent project templates

Console App (.NET Framework) C#
WCF Service Library C#
Console Application C#
ASP.NET Core gRPC Service C#

Search for templates (Alt+S)

Clear all

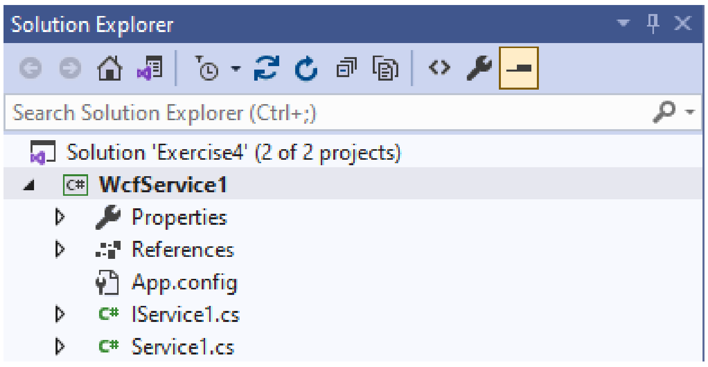
C# | Windows | Service

WCF Service
A Web site for creating WCF services. This template does not produce a project file and has limited MSBuild support.
C# Windows Web Service

WCF Service Application
A project for creating WCF Service Application that is hosted in IIS/WAS
C# Windows Web Service

WCF Service Library
A project for creating a host-independent WCF service class library (.dll)
C# Windows Library Web Service

- Look at the content of the project. Draw attention at:
  - **IService1.cs** –the declaration file (interface) of the contract,
  - **Service1.cs** – implementation of the contract,
  - **App.config** – configuration of properties and accessibility of contract implementation.


Solution Explorer

Search Solution Explorer (Ctrl+;)

Solution 'Exercise4' (2 of 2 projects)
WcfService1
Properties
References
App.config
IService1.cs
Service1.cs

Note: in the latest version of VS, file names may change after changing the interface or class name from the default (as above) to your own - such as your own class or interface name. This behavior is configurable (it can be disabled).

- Open the **IService1.cs** file and define the service contract - the **ICalculator** interface containing Add, Sub, and Multiply methods:
  - Delete unused code.
  - Add or process the code as below.

Note: the namespace is set such as the name of the project - it's worth not to change it

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Runtime.Serialization;
using System.ServiceModel;
using System.Text;
```

	```
	namespace WcfService1
	{
	  [ServiceContract(ProtectionLevel = ProtectionLevel.None)]
	  public interface ICalkulator
	  {
	    [OperationContract]
	    double Add(double n1, double n2);
	    [OperationContract]
	    double Sub(double n1, double n2);
	    [OperationContract]
	    double Multiply(double n1, double n2);
	} }
	```
	 ***Note**: after changing the name from Iservice1 to ICalculator (using the menu option - not manually!) the name of the file may change in the project.*
	• The property (behavior) `ProtectionLevel` (set to `None`) has been added to avoid the issue of authorization when accessing services.

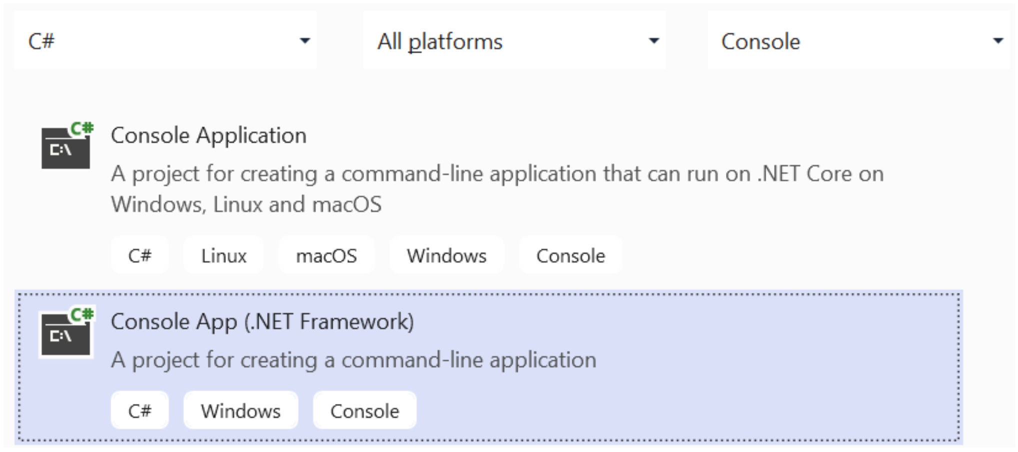
3. Implementacja kontraktu usługi WCF	• Open the **Service1.cs** file. Enter the code of **MyCalculator** class implementing the **ICalculator** interface: Implement each of the required methods:
	```
	using System;
	using System.Collections.Generic;
	using System.Linq;
	using System.Runtime.Serialization;
	using System.ServiceModel;
	using System.Text;

	namespace WcfServiceContract1
	{
	  public class MyCalculator : ICalculator
	  {
	    public double Add(double val1, double val2) {
	      ...
	    }
	    public double Sub(double val1, double val2) {
	      ...
	    }
	    public double Multiply(double val1, double val2) {
	      ...
	    }
	  }
	}
	```
	 • In place of the dots **...** add the appropriate code for each method:
	- performing the appropriate action,
	- displaying in the console information what is called, what was received in the call, and what is returned,
	- returning the calculated value.

4. WCF service host.

Create a console application running the service (Service Host).

- Add to the previous solution a second project of the console application giving it own name (here: WcfServiceHost1) – option: **Add…** → **New Project**.

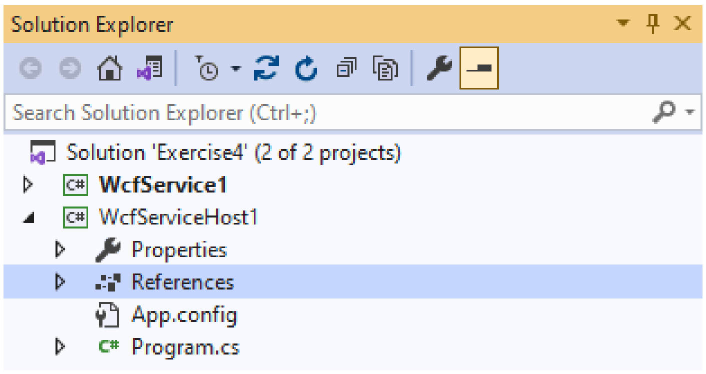


C# | All platforms | Console

Console Application
A project for creating a command-line application that can run on .NET Core on Windows, Linux and macOS
C# Linux macOS Windows Console

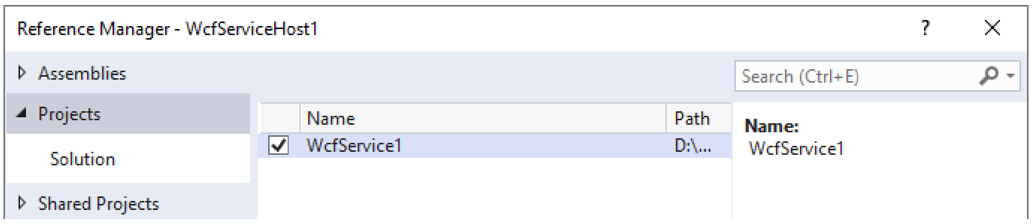
Console App (.NET Framework)
A project for creating a command-line application
C# Windows Console

- Add in the project a reference to the WCF contract project:
  - Select the **References** node in Solution Explorer and select the **Add Reference** option



Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'Exercise4' (2 of 2 projects)
WcfService1
WcfServiceHost1
Properties
References
App.config
Program.cs

  - In the Reference Manager window, select **Solution/Project**, mark the WCF contract project and confirm:

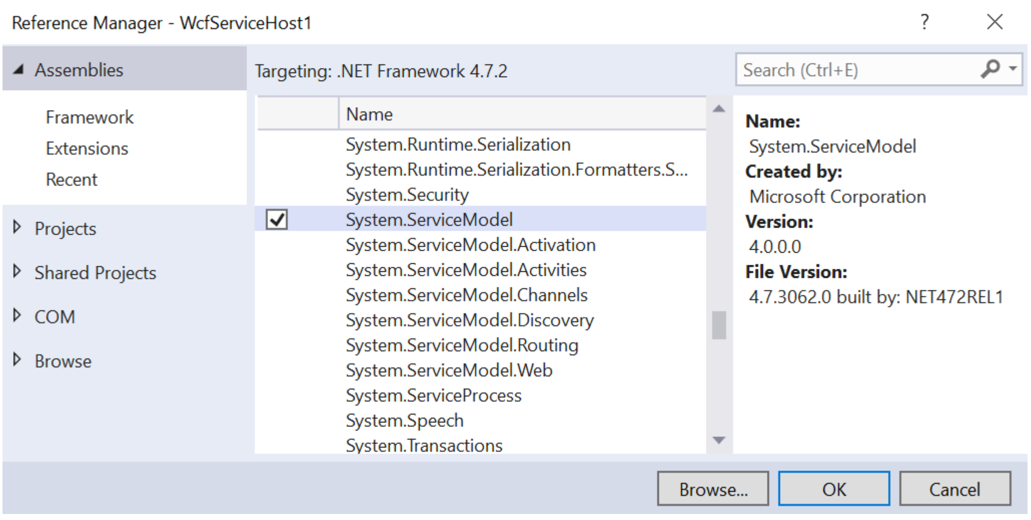


Reference Manager - WcfServiceHost1
Assemblies
Projects
Solution
Shared Projects
Search (Ctrl+E)

	Name	Path
✓	WcfService1	D:\...

Name: WcfService1

- Add in the project a reference to **System.ServiceModel**:
  - Right-click the **References** node in Solution Explorer and select the **Add Reference** option
  - In the Reference Manager window, select **Assemblies** / **Framework**, mark **System.ServiceModel**, and confirm:



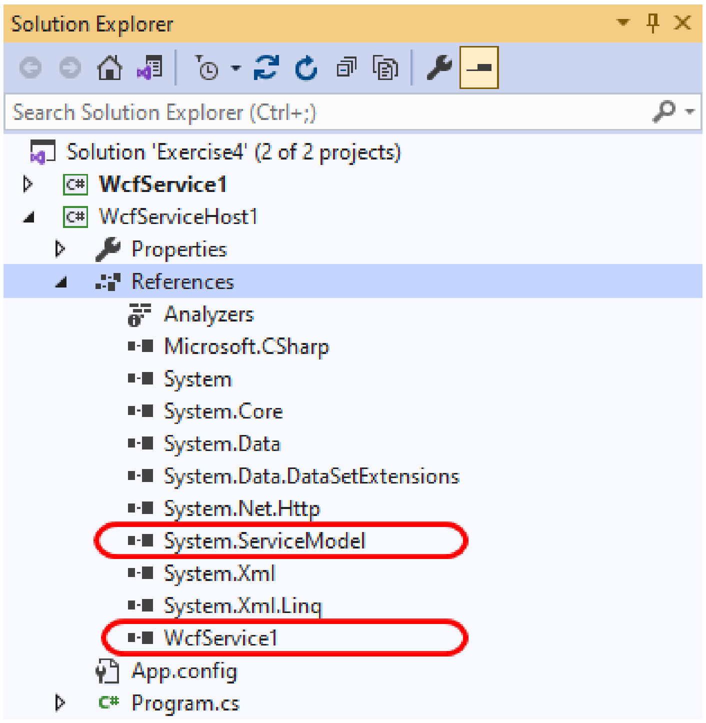
Reference Manager - WcfServiceHost1

Targeting: .NET Framework 4.7.2

	Name
	System.Runtime.Serialization
	System.Runtime.Serialization.Formatters.S...
	System.Security
✓	System.ServiceModel
	System.ServiceModel.Activation
	System.ServiceModel.Activities
	System.ServiceModel.Channels
	System.ServiceModel.Discovery
	System.ServiceModel.Routing
	System.ServiceModel.Web
	System.ServiceProcess
	System.Speech
	System.Transactions

Name: System.ServiceModel
Created by: Microsoft Corporation
Version: 4.0.0.0
File Version: 4.7.3062.0 built by: NET472REL1

- Verify the appearance of additional references in the project as in the figure below



```
Solution 'Exercise4' (2 of 2 projects)
  WcfService1
  WcfServiceHost1
    Properties
    References
      Analyzers
      Microsoft.CSharp
      System
      System.Core
      System.Data
      System.Data.DataSetExtensions
      System.Net.Http
      System.ServiceModel
      System.Xml
      System.Xml.Linq
      WcfService1
    App.config
    Program.cs
```

- Open the **Program.cs** file and enter the code that performs the following functions:
  - Creation of the URI with the service base address,
  - Creating a service instance.
  - Adding the service endpoint.
  - Activation metadata exchange (service information).
  - Running the service (and finally closing the service).

  Instead of **xxx** enter port number of value: **10000 + workstation number in the laboratory**. Instead of the ***BaseName*** (service name), enter your own name of the service.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace WcfServiceHost
{
  class Program
  {
    static void Main(string[] args)
    {
      // Step 1 Create the URI of service base address
      Uri baseAddress = new Uri("http://localhost:xxx/BaseName");
      // Step 2 Create service instance.
      ServiceHost myHost = new
               ServiceHost(typeof(MyCalculator), baseAddress);
      // Step 3 Add the endpoint.
      BasicHttpBinding myBinding = new BasicHttpBinding();
      ServiceEndpoint endpoint1 = myHost.AddServiceEndpoint (
                             typeof(ICalculator),
                             myBinding,
                             "endpoint1");
      // Step 4 Set up metadata and publish service metadata
      ServiceMetadataBehavior smb = new ServiceMetadataBehavior();
      smb.HttpGetEnabled = true;
      myHost.Description.Behaviors.Add(smb);

      // Step 5 Run the service.
      try
      {
        myHost.Open();
        Console.WriteLine("-->Service started.");
        Console.WriteLine("-->Press <ENTER> to STOP service...");
        Console.WriteLine();
        Console.ReadLine();    // to not finish app immediately:
        myHost.Close();
      }
      catch (CommunicationException ce)
      {
        Console.WriteLine("-->Exception occurred: {0}", ce.Message);
        myHost.Abort();
      }
    }
  }
}
```

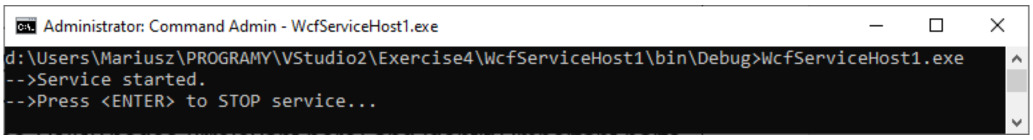
- Remove errors by adding the import of appropriate libraries - when pointing with cursor, the **Quick Actions and Refactorings…** option in the context menu or **Show potential fixes**.
  - Most often, this will be adding the **using** import directive

5. Testing the service

NOTE: **to run the service, you must have the administrator's rights in Windows. Otherwise, the system must be properly configured.**

- Check the correct operation of the application

- Build the executable code of the application.
- Run from the console the application hosting the WCF service.



```
Administrator: Command Admin - WcfServiceHost1.exe
d:\Users\Mariusz\PROGRAMY\VStudio2\Exercise4\WcfServiceHost1\bin\Debug>WcfServiceHost1.exe
-->Service started.
-->Press <ENTER> to STOP service...
```

- Examine the service metadata and service dwscription.
  - Start the web browser and connect to the address:
    http://localhost:xxx/*BaseName*
  - Familiarize with service information displayed in the browser.

  Connection to the host and displaying the page with the appropriate description indicates the correct operation of the application.

  - Go to the service WSDL description page using link of the first page.
    identify important parts of the service description: types, messages, operations, endpoint, etc.

Running the service (contract – not the host) from Visual Studio platform automatically starts the built-in client that allows you to test the operation of the service.

- Run the service from VS,
  Notice that in such case the service is run on specific reserved for VS port number (other than defined in the host).
  - Click on any operation in this client and see the form of the XML message (SOAP message) to send.
  - Enter some values of parameters and call the operation.
  - Examine the XML content of request and response messages.
- Run the Postman (or SOAPUI) application to test operation from other client.
  - In the new HTTP request of the application configure the following:
    – method: POST,
    – address: endpoint address of the service (you may verify it in WSDL file),

    Additional headers:

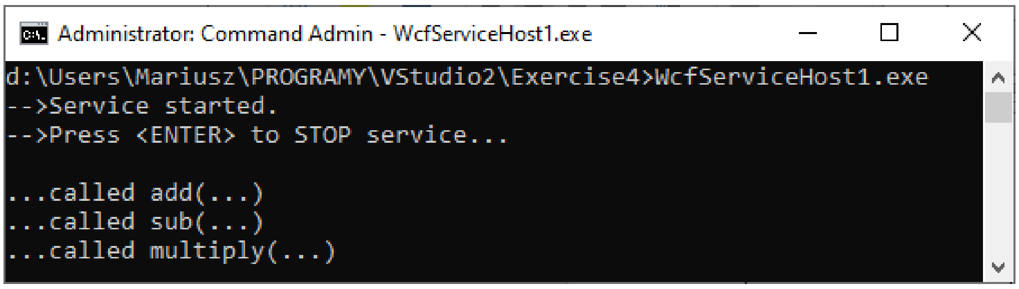
    – Content-type header: text/xml
    Note: for WSHttpBinding it would be application/soap+xml – but not in our simpler case,
    – SOAPAction header: enter here value of **soapAction** attribute of the operation specified in WSDL file – usually it is of the following form:

	http://tempuri.org/*service_interface_name*/*operation_name* (e.g. http://tempuri.org/ICalculator/add),
	o In the body enter the simplest form of SOAP request message
	– – you may copy it from test client run by VS platform,
	– leave the header empty as it is not well supported by Postman,
	– The request would looks like:
	`<?xml version="1.0" encoding="utf-8"?>`
	`<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/">`
	`    <s:Header/>`
	`  <s:Body>`
	`    <Add xmlns="http://tempuri.org/">`
	`      <val1>-111</val1>`
	`      <val2>13.24</val2>`
	`    </Add>`
	`  </s:Body>`
	`</s:Envelope>`
	o Run the request and verify the response.
	Note: in most cases the errors that "*server was unable to process the request due to an internal error*" are induced by mistakes in names of methods, parameters, SOAPAction, using small instead capital letters (or reversely) etc.

6. Implementing the client	**Creating a service client application and client proxy in code.** • Add to the solution a third project of the application using the **Visual C # Console App (.Net Framework)** template. • Check (and if necessary correct) the version of the Application Framework (same as in the second project). • Add in the project a reference to **System.ServiceModel** (same as in the second project) – you can also omit this step now and add the reference by using VS hints for code corrections. • Add to the project the interface item – right-click project and option: *Add/New Item…/Interface*. Define in the file exactly the same interface as the service contract interface (here: **ICalculator** interface). • Open the **Program.cs** file and enter the client code that performs the following operations: o Creating a client instance (WCF client proxy). – Creating base address Uri – Creating binding – Creating endpoint – Creating client proxy with use of Channel factory o Invoking service operations from the client (proxy). o Closing the client.

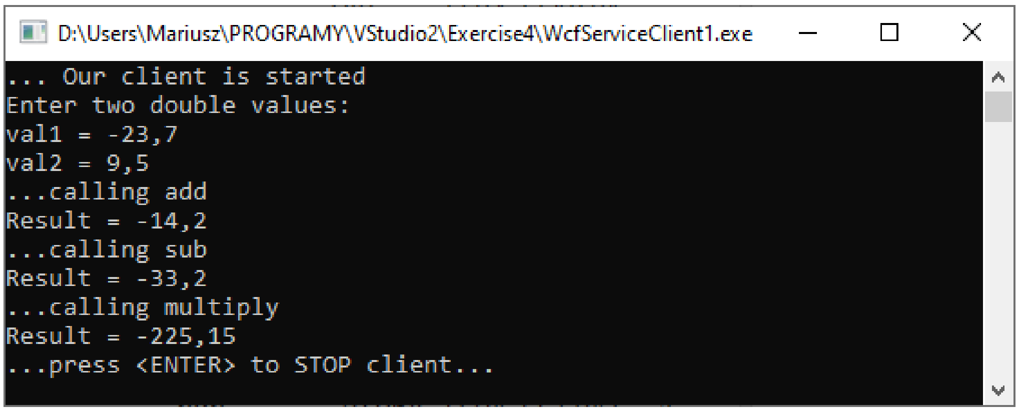
	```
	namespace WcfServiceClient1 {
	  class Program {
	    static void Main(string[] args) {
	      Console.WriteLine("... The client is started");
	      // Step 1: Create client proxy based on communication channel.
	      // base address:
	      Uri baseAddress;
	      /* using channel: */
	      // binding:
	      BasicHttpBinding myBinding = new BasicHttpBinding();
	      baseAddress = new
	              Uri("http://localhost:10000/BaseName/endpoint1");
	      // endpoint:
	      EndpointAddress eAddress = new EndpointAddress(baseAddress);
	      // channel factory:
	      ChannelFactory<ICalculator> myCF = new
	                   ChannelFactory<ICalculator>(myBinding, eAddress);
	      // client proxy (here myClient) based on channel
	      ICalculator myClient = myCF.CreateChannel();
	      // Step 2: service operations call.
	      Console.WriteLine("...calling add");
	      double result = myClient.add(-3.7, 9.5); //just example values
	      Console.WriteLine("Result = "+result);
	      [...] // here other operations
	      Console.WriteLine("...press <ENTER> to STOP client...");
	      Console.WriteLine();
	      Console.ReadLine();   // to not finish app immediately:
	      // Step 3: Closing the client - closes connection and clears
	         resources.
	      ((IClientChannel)myClient).Close();
	      Console.WriteLine("...Client closed - FINISHED");
	    }
	  }
	}
	```
	 ○ Insert the appropriate messages in place of the dots
	○ Fix errors by adding import of the appropriate names.

7. Testing the operation of the application	• Run the service (application hosting the WCF service) in one Windows console. • Run the client in the second Windows console. • Check the results of the operation. • The effect of service's and client's activities should be similar to the illustrations shown below.


Administrator: Command Admin - WcfServiceHost1.exe

```
d:\Users\Mariusz\PROGRAMY\VStudio2\Exercise4>WcfServiceHost1.exe
-->Service started.
-->Press <ENTER> to STOP service...

...called add(...)
...called sub(...)
...called multiply(...)
```


D:\Users\Mariusz\PROGRAMY\VStudio2\Exercise4\WcfServiceClient1.exe

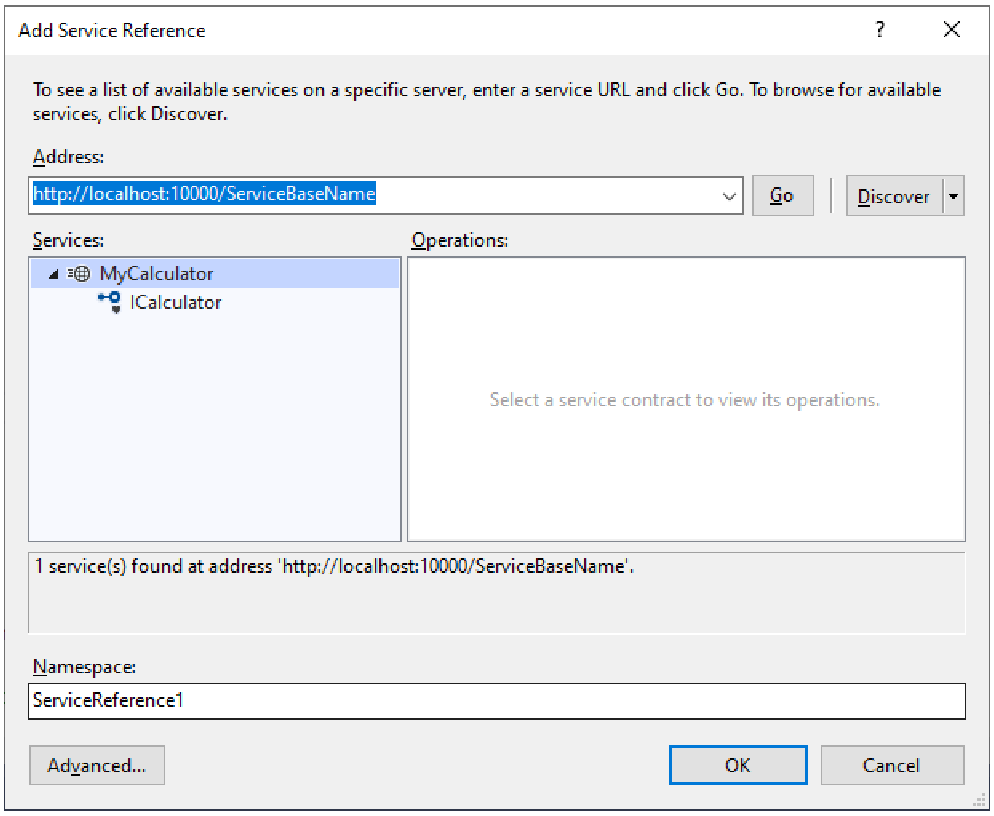
```
... Our client is started
Enter two double values:
val1 = -23,7
val2 = 9,5
...calling add
Result = -14,2
...calling sub
Result = -33,2
...calling multiply
Result = -225,15
...press <ENTER> to STOP client...
```

- Finish all applications.

8. Creating client proxy using VS platform tools

Creating a service client application and *client proxy* using Visual Studio: *Add Service Reference* option.

- Add service reference to the defined service:
  (the illustration continues in the picture)


Add Service Reference

To see a list of available services on a specific server, enter a service URL and click Go. To browse for available services, click Discover.

Address: http://localhost:10000/ServiceBaseName

Services:
- MyCalculator
  - ICalculator

Operations:

Select a service contract to view its operations.

1 service(s) found at address 'http://localhost:10000/ServiceBaseName'.

Namespace: ServiceReference1

  - **Run the application hosting the WCF service first!**

	○ Right-click the **References** node in Solution Explorer and select the **Add Service Reference** option. ○ In the Add Service Reference window, enter the address of the service (endpoint) in the **Address** field: **http://localhost:*xxx*/*BaseName*** Instead of ***xxx***, enter the appropriate port number ○ Press **Go** and the available service at the specified access point should appear. Choosing a contract (interface) will additionally show the available operations (methods). ○ Confirm the choice with the **OK** button. **In the above way, the client proxy code is generated – an attached module that performs service calls.**

9. Client configuration.	The client configuration is included in the **App.config** configuration file created by adding service reference operation. No additional code to build client proxy is necessary now (like in previous implementation) but one line (also specified in metadata).
	• Open the client's **Program.cs** file and modify the client code that that create client proxy and closes it:
	○ Comment lines of creating: binding, address, endpoint, and channel factory.
	○ Instead previous creation of client proxy and closing it enter the following code:
	`// Step 1: Create client proxy`
	`CalculatorClient myClient = new CalculatorClient();`
	`// Step 2: service operations call.`
	`[...] // here calling operations`
	`// Step 3: Closing the client`
	`myClient.Close();`
	`Console.WriteLine("...Client closed - FINISHED");`
	• Open the client's **App.config** file and analyze its content.
	• Pay particular attention to the section its name, and type of **binding**, section and name of the endpoint, and contract specification.
	• Its content should be similar to the following.
	`<?xml version="1.0" encoding="utf-8" ?>`
	`<configuration>`
	`  <startup> ... </startup>`
	`  <system.serviceModel>`
	`    <bindings>`
	`      <basicHttpBinding>`
	`        <binding name="BasicHttpBinding_ICalculator" />`
	`      </basicHttpBinding>`
	`    </bindings>`

	`<client>`
	`  <endpoint`
	`    address=http://localhost:10000/ServiceBaseName/endpoint1`
	`    binding="basicHttpBinding"`
	`    bindingConfiguration="BasicHttpBinding_ICalculator"`
	`    contract="ServiceReference1.ICalculator"`
	`    name="BasicHttpBinding_ICalculator" />`
	`</client>`
	`  </system.serviceModel>`
	`</configuration>`

10. Testing the operation of the application	• Run the service (application hosting the WCF service) in one Windows console, run the client in the second console, and check the results of the operation. • Finish all applications.

11.Modyfikacja kontraktu	• Add another operation in the contract interface:
	`[OperationContract]`
	`double Summarize(double n1);`
	○ Add a global variable in the class implementing the contract
	`double sum=0;`
	• Define in contract implementation the "global" sum operation:
	`public double Summarize(double n1) {`
	`  sum += n1;`
	`  return sum;`
	`}`

12.Service Host modification	• Before starting the service (before calling `Open(…)` method): create additional WSHttpBinding object and add additional endpoint:
	`WSHttpBinding binding2 = new WSHttpBinding();`
	`binding2.Security.Mode = SecurityMode.None;`
	`ServiceEndpoint endpoint2 = myHost.AddServiceEndpoint(`
	`                                typeof(ICalkulator),`
	`                                binding2, "endpoint2");`
	The `SecurityMode.None` is set to simplify code in configuration file.
	• Next add the code displaying information about endpoints:
	`Console.WriteLine("\n---> Endpoints:");`
	`// copy below code for each endpoint:`
	`Console.WriteLine("\nService endpoint {0}:", endpoint1.Name);`
	`Console.WriteLine("Binding: {0}", endpoint1.Binding.ToString());`
	`Console.WriteLine("ListenUri: {0}", endpoint1.ListenUri.ToString());`
	Optional additional code:
	In the project configuration file, define additional endpoint of service:
	• Open the **App.config** file and enter in the **<configuration>** node, after the **<startup>** node, the code that defines the service (node **<service>**), and inside it the service endpoint (node **<endpoint>**):

	`<system.serviceModel>`
	`  <services>`
	`    <service name="WcfService1.MyCalculator">`
	`      <endpoint name="myEndpoint3" address="/endpoint3"`
	`        binding="basicHttpBinding"`
	`        contract="WcfService1.ICalculator">`
	`      </endpoint>`
	`    </service>`
	`  </services>`
	`</system.serviceModel>`
	• Open the **Program.cs** file and enter the code that retrieve endpoint defined in the app.config file:
	`ServiceEndpoint endpoint3 = myHost.Description.Endpoints.Find(`
	`             new Uri("http://localhost:xxx/BaseName/endpoint3") );`
	• Next add the code displaying information also about endpoint3.

13. Testing the operation of the service	• Rebuild service contract and service host. • Run the service from the console and check the operation. • Familiarize with the displayed data in the host console.

14. Client modification for operation with the second host	• Make sure that the host works and update the service references in the client application:
	  ○ right-click in the solution the node **Service References** → **ServiceReference1** and choose **Update Service Reference** option.
	• Check what changes have appeared in the **App.config** file.
	The code below is to do because a client proxy can be created ***without an endpoint specification only if there is one endpoint*** *of a given type.* ***Otherwise****, the endpoint* ***must be specified*** *for the proxy.*
	• In the **Program.cs** file, comment the instruction of client proxy creation without the endpoint specification in the constructor.
	`//CalculatorClient myClient = new CalculatorClient();`
	• Instead, add instructions of creating additional client proxies for using various endpoints.
	When creating client proxies, specify the name of the endpoint corresponding to those generated in the **App.config** file.
	`CalculatorClient client1 = new`
	`                   CalculatorClient("WSHttpBinding_ICalculator");`
	`CalculatorClient client2 = new`
	`                 CalculatorClient("BasicHttpBinding_ICalculator");`
	`CalculatorClient client3 = new CalculatorClient ("myEndpoint3");`
	Endpoint names (client 3) and binding names (clients 1 and 2) must fit the names used in client **App.config** file.

	• Similarly, comment out the instruction of calling the Add (...) operation, Sub(...) operation, etc. of the previous client proxy of `myClient` and instead add calls for new clients:
	`result = clientx.operation(…);`
	where **x**- client number, *operation(...)* – is Add (...), Sub(...), Multiply(...), ... operation.
	• Add a call of **Summarize** operation twice for each client proxy (that is for all endpoints):
	`result = clientx.Summarize(some_value);`
	where **x**- client number, *some_value* - a numerical value.
	• Add displaying of relevant comments and results in the client console.

15. Testing the operation of the service	• Run the client in the Windows console (*the service must be run*). • Pay attention to the individual values of the summation results (Summmarize(…) operation). • Stop the service and the client. • In the **Service1.cs** service contract file (contract implementation), add the instruction describing the service behavior as a single instance for all clients – just before the class definition: `[ServiceBehavior(InstanceContextMode = InstanceContextMode.Single)]` • Rebuild contract and service host. • Run the service from the console. • Upgrade the service reference in the client and rebuild the client. • Run the client (twice) and again pay attention to the individual values of the summarize results - a change compared to the previous version of the service.

3 Exercise – Part II

The detailed and final requirements for Part II are set by the teacher.

A. Practice the technique of creating WCF services and clients:

1. Defining and implementing contracts.
2. Creating a service host.
3. Defining endpoints with specific addresses.
4. Defining the communication method (BasicHttp, WSHttp, NetTCP).
5. Creating a client, binding and calling service operations.

B. Brace for writing a program in the classroom, containing elements with practiced functionalities, according to the teacher's instructions.