Exercise 2

WCF - basics - defining services and clients, synchronous and asynchronous operations

Author: Mariusz Fras

1 Objectives of the exercise

The purpose of the exercise is:

1. Getting to know the basic architecture of WCF applications
2. Getting acquainted with the basics of creating a service with a WSDL description and available via the SOAP protocol (here: WCF services), and the client of such a service (here: WCF client).
3. Understanding options for configuring services - endpoints, transport, how the service works.

- **The first part of the task**is to be performed according to the given instructions and any instructions of the laboratory teacher.
- **The second part of the task is to be prepared and handed over or to be performed according to**
  instructor's instructions in the next class.

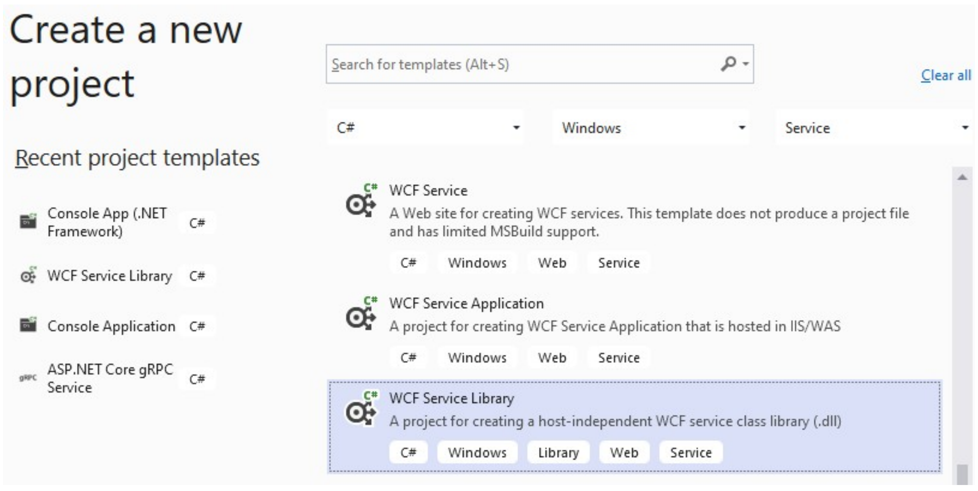
2 Task - part I
Construction of a basic WCF application

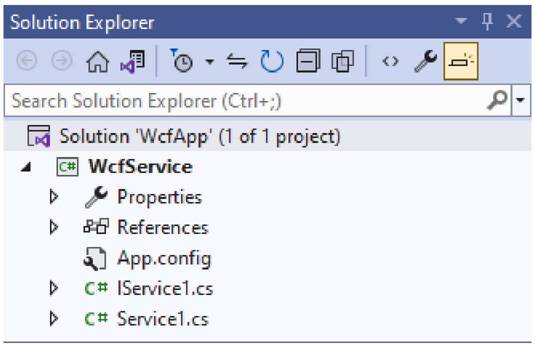
In the task, a solution will be implemented that includes: a) a service run as a separate application and b) a client using this service. The service and the client will be implemented in the Visual Studio tool (hereinafter abbreviated as VS) as a WCF service. The task of implementing a basic WCF client-server application consists of several steps:

1. Define the service contract.
2. Implementation of the service contract (service implementation).
3. Create a service hosting application
   – here: it is a console application – the so-called self-hosted service.
4. Client application implementation (including proxy client).
5. Expansion of the service and client for asynchronous operations.

ATTENTION*: changes (e.g. class names, etc.) in the code automatically generated by the platform are best implemented through the refactoring option of the platform*

- *Usually the Refactor option in the context menu or the corresponding option in the menu* ***contextual****(e.g. Rename - when the Refactor option is not highlighted).*

1. Defining the service contractWCF	• Create a new solution and application project from the Visual C# WCF Service Library template giving your own solutions and project names (here: WcfApp and WcfService).  Create a new project Search for templates (Alt+S) Clear all C# Windows Service Recent project templates Console App (.NET Framework) C# WCF Service Library C# Console Application C# ASP.NET Core gRPC Service C# 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 • Review the content of the project. Pay attention to the following: ○ **IService1.cs**– contract declaration file (interface), ○ **Service1.cs**– contract implementation, ○ **app.config**– configuration of ownership and accessibility of the contract implementation.



Solution Explorer

Search Solution Explorer (Ctrl+;)

```
Solution 'WcfApp' (1 of 1 project)
◢ WcfService
    ▷ Properties
    ▷ References
      App.config
    ▷ IService1.cs
    ▷ Service1.cs
```

***Attention**: file names may change after renaming the interface or class from the default (as in the picture above) to your own name. This behavior is configurable (can be turned off).*

- Open the IService1.cs file and define the service contract - interface **iCalculator**containing Add, and Multiply methods:
  - Remove unused code.
  - Add or modify the code to the form:

```
[ServiceContract(ProtectionLevel
=ProtectionLevel.none)]public interfaceiCalculator
{
  [OperationContract]
  doubleAdd(doubleval1,doubleval2);
  [OperationContract]
  doublemultiple(doubleval1,doubleval2);
}
```

***Attention**: here after changing the name from Iservice1 to ICalculator (menu option – not manually!) the file name may change in the project.*

- propertyProtectionLevel (set to None) added to simplify the service.

2. Service contract implementation WCF

- Open the Service1.cs file. Enter the code of the MyCalculator class that implements the ICalculator interface: Implement each of the required methods:

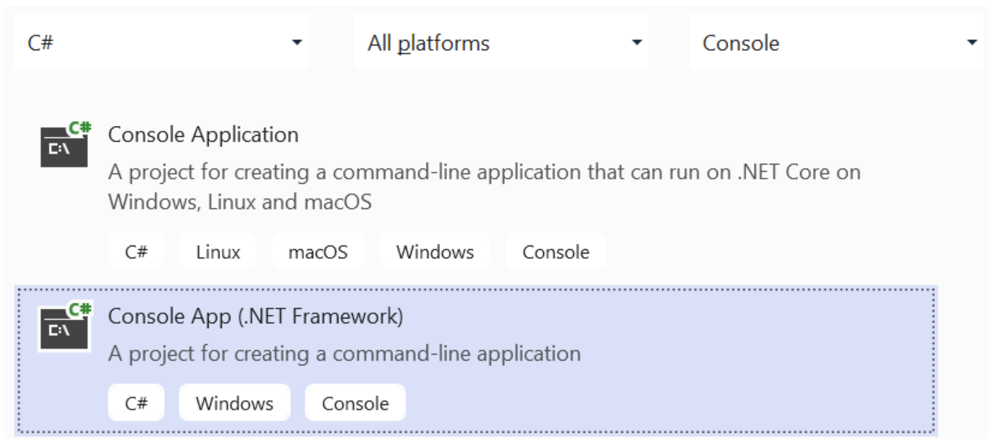
```
public classMyCalculator:iCalculator{public
  doubleAdd(doubleval1,doubleval2) {
    ...
  }
  public doublemultiple(doubleval1,doubleval2) {
    ...
  }
}
```

- In place of the dots ... add the appropriate code for each method:
  - performing the appropriate action,
  - displaying information in the console what is called, what was received in the call and what is returned,
  - return the appropriate value.

3. Hosting a WCF service

Create a console application that hosts a WCF service (Service Host).

- Add a second console application project to the existing walkthrough, giving it your own name (here: WcfServiceHost) – option: Add… □ New Project.

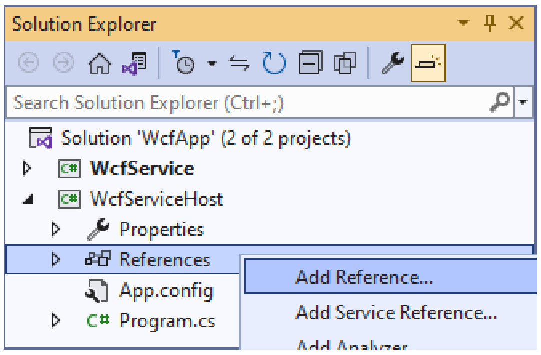


C# | All platforms | Console

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

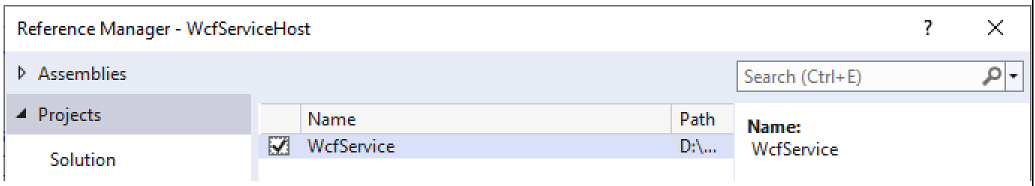
Console App (.NET Framework)
A project for creating a command-line application
C# Windows Console

- Check (and possibly set) the version of the Application Framework.
  - In Solution Explorer context menu Properties, option Application □ Target Framework.
- Add a reference to the WCF service contract project in the project:
  - In Solution Explorer, select the References folder and select an option
    AddReference.



Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'WcfApp' (2 of 2 projects)
WcfService
WcfServiceHost
Properties
References
App.config
Program.cs

  - In the credential manager window, select Solution/Project , select the WCF service contract project and commit:



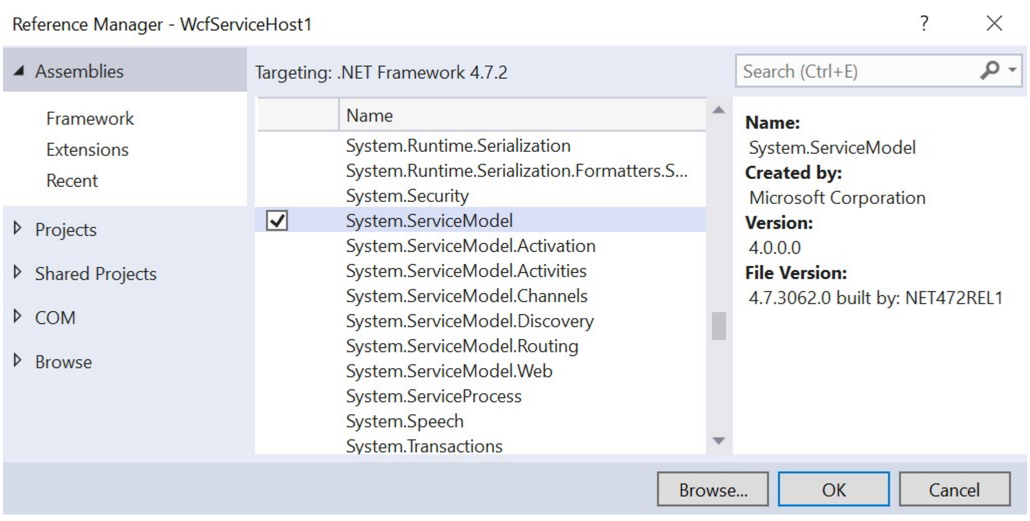
Reference Manager - WcfServiceHost

Assemblies
Projects
Solution
Search (Ctrl+E)

	Name	Path
☑	WcfService	D:\...

Name:
WcfService

- Add a reference to System.ServiceModel in your project:
  - In Solution Explorer, right-click the References folder and select Add Reference.
  - In the reference manager window, select Assemblies/Framework, select System.ServiceModel and confirm.

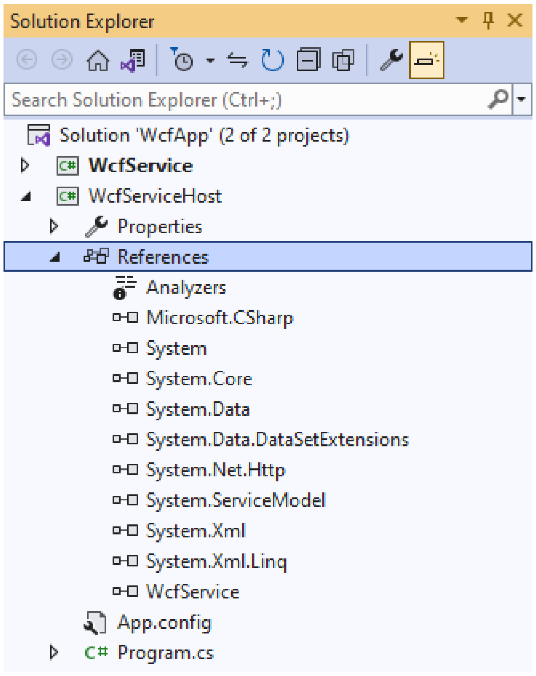

Reference Manager - WcfServiceHost1

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 picture below:



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

- Open the Program.cs file and enter the following code:
  - Create a URI with the base site address.
  - Create a service instance.
  - Adding a site endpoint.
  - setting metadata (providing information about the website).
  - Launching the website (and finally closing the website).

  Instead of xxx, enter the network port number (e.g. 10000 + lab station number).
  Replace ServiceBaseName (service name) with your own service name.

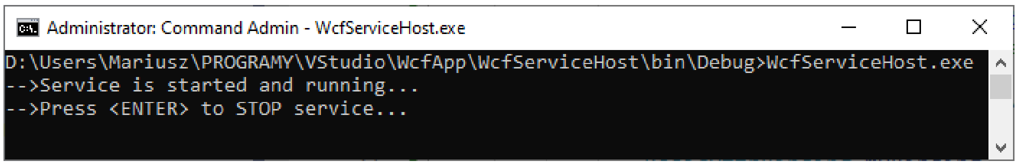
```
static voidmain(string[] arguments)
{
  // Step 1 URI for the base site address
  UribaseAddress =newUri("http://localhost:xxx/ServiceBaseName");
  // Step 2 Service
  instanceServiceHostmyHost
  =new
           ServiceHost(typeof(MyCalculator), baseAddress);
  // Step 3 Service endpoint
  BasicHttpBindingmyBinding
  =newBasicHttpBinding();ServiceEndpointendpoint1 =
  myHost.AddServiceEndpoint (
                                typeof(iCalculator),my
                                Binding,"endpoint1");
  // Step 4 Set metadata
  ServiceMetadataBehaviorsmb
  =newServiceMetadataBehavior();smb.HttpGetEnabled =true;
  myHost.Description.Behaviors.Add(smb);
  try{
    // Step 5 Launching the
    website.myHost.Open();
    Console.WriteLine("Service is started and
    running.");Console.WriteLine("Press <ENTER> to STOP
    service...");Console.WriteLine();
    Console.ReadLine();    // not to terminate
    immediately:myHost.Close();
  }
  catch(CommunicationExceptionce)
    {Console.WriteLine("Exception occurred: {0}",
    ce.Message); myHost.Abort();
  }
}
```

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

4.Testing the operation of the application

***ATTENTION**: to run the service outside the VS platform (e.g. from the console) you must have administrator rights in the system. Otherwise, the system must be additionally configured accordingly.*

- Test the correct operation of the application
  - Build the application's executable code.
  - Run the WCF service hosting application from the command line



Administrator: Command Admin - WcfServiceHost.exe

```
D:\Users\Mariusz\PROGRAMY\VStudio\WcfApp\WcfServiceHost\bin\Debug>WcfServiceHost.exe
-->Service is started and running...
-->Press <ENTER> to STOP service...
```

- Check the site metadata and service description
  - Launch your browser and connect to the address:

    http://localhost:xxx/ServiceBaseName

	○ Read the description of the site.

Connecting to the host and displaying the page with the appropriate description means that the application works correctly.

- Go to the service WSDL description page.
  Identify the important parts of the service description: types, messages, operations, access point, etc.
- Browse the content of the page:
  http://localhost:xxx/ServiceBaseName?xsd=xsd0

Starting the service (not the host) from the VS (Visual Studio) level automatically launches the built-in client that allows you to test the operation of the service.

- Start the service from within VS
  Note that in this case the service is available (works) on a special port reserved by VS (other than defined in the host).
  - Click on an operation in this client (e.g. Add)
  - Enter some data for the parameters and call the operation
  - Check the form of the XML message (SOAP messages) that is sent and received.
  - Shut down the service running from VS. Leave the service running from the console.
- Run the Postman program (alternatively, you can SOAPUI) to test the operation of the site.
  - Create an HTTP request by configuring:
    - POST method
    - address as for the service endpoint - check it in the WSDL in the service->port->address section.

    HTTP headers:
    - **Content-type**= test/xml (this is the case here)
      Note: for WSHttpBinding the type is different: application/soap+xml
    - **SOAPAction**– set here the value of the soapAction attribute of the called operation defined in WSDL – here usually in the following form:http://tempuri.org/service_interface_name/operation_name(eghttp://tempuri.org/ICalculator/Add)
      Note: in WSDL, the operation name may start with a lowercase letter instead of an uppercase one.
  - In the body of the request, enter the simplest form of the SOAP message/request
    - you can copy it from the VS test client
    - leave the <Header> section empty (Postman has trouble handling it)

– the request should look similar to the following:

```
<?xmlversion="1.0"encoding="utf-8"?>
<s:Envelopexmlns:s="http://schemas.xmlsoap.org/soap/envelope/">
    <s:Header/></s:Header>
  <s: Body>
    <Addxmlns="http://tempuri.org/">
      <val1>-98.76</val1>
      <val2>12.34</val2>
    </Add>
  </s: Body>
</s:Envelope>
```

o Run/send request and check response.

Attention: Possible errors, e.g. "*server was unable to process the request due to an internal error*"are usually caused by errors/typos in method names, parameters, SOAPAction, usage lowercase instead of uppercase (or vice versa), etc.

5. Implementation of the service client - version I

Creation of a service client application and a client proxy (client proxy)a separate code.

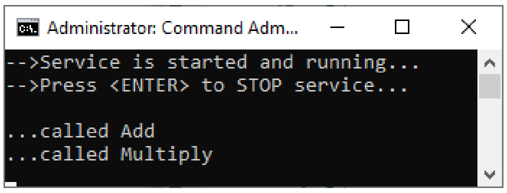
- Create a separate walkthrough with the third application project from the C# Console App template (.NetFramework) giving it its own name (here WcfClient).
- Check (and possibly set) the application's Framework version (same as in the second project).
- Add a reference to System.ServiceModel in the project (same as in the second project (for the host)) - it can also be done as a result of platform prompts when entering the code.
- Add an interface to the project (Add/New Item…/Interface option) Define the service contract interface (ICalculator) exactly the same way.
- In the client file Program.cs, enter code that:
  - Creates a client instance (client proxy)
    - creating a Uri object of the base address of the service.
    - creating a binding
    - creating an endpoint
    - creating a proxy client using a channel factory
  - Invokes a service operation using a proxy client
  - Closes the client

```
static voidmain(string[] args) {
  Console.WriteLine("... The client is started");
// Step 1: Create client proxy based on communication channel.
  // base address:
  Uri baseAddress;

  // binding, address, endpoint
  address:BasicHttpBinding myBinding
  =newBasicHttpBinding(); baseAddress =new
        Uri("http://localhost:10000/ServiceBaseName/endpoint1");
```

```
       EndpointAddress eAddress =newEndpointAddress(baseAddress);
       // channel
       factory:ChannelFactory<ICalculator> myCF
       =new
                    ChannelFactory<ICalculator>(myBinding, eAddress);
       // client proxy (here myClient) based on
       channelICalculator myClient = myCF.CreateChannel();
     // Step 2: service operations call.
       Console.Write("...calling Add (for entpoint1) ");
       doubleresult = myClient.add(-3.7, 9.5);//just example
       valuesConsole.WriteLine("Result = "+result);
       [...]// here possible 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");
     }
```

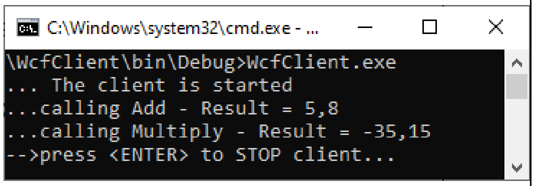
6.Testing the operation of the application

- Start the service (the application hosting the WCF service) in a single console window.
- Start the client in a second console window.
- Check the operating results.
- The effect of the customer and the service should be similar to the illustrations.



Administrator: Command Adm...

```
-->Service is started and running...
-->Press <ENTER> to STOP service...

...called Add
...called Multiply
```



C:\Windows\system32\cmd.exe - ...

```
\WcfClient\bin\Debug>WcfClient.exe
... The client is started
...calling Add - Result = 5,8
...calling Multiply - Result = -35,15
-->press <ENTER> to STOP client...
```

- Quit all applications.

7. Modification of the website host

- Open the host's Program.cs file and add the following code:
  - Adding another endpoint (for WSHttp transport).
  - View contract information.
- <u>Before starting the service</u>(before the Open() function) create a WSHttpBinding object (for the WS Http transport) and add an additional endpoint/endpoint:

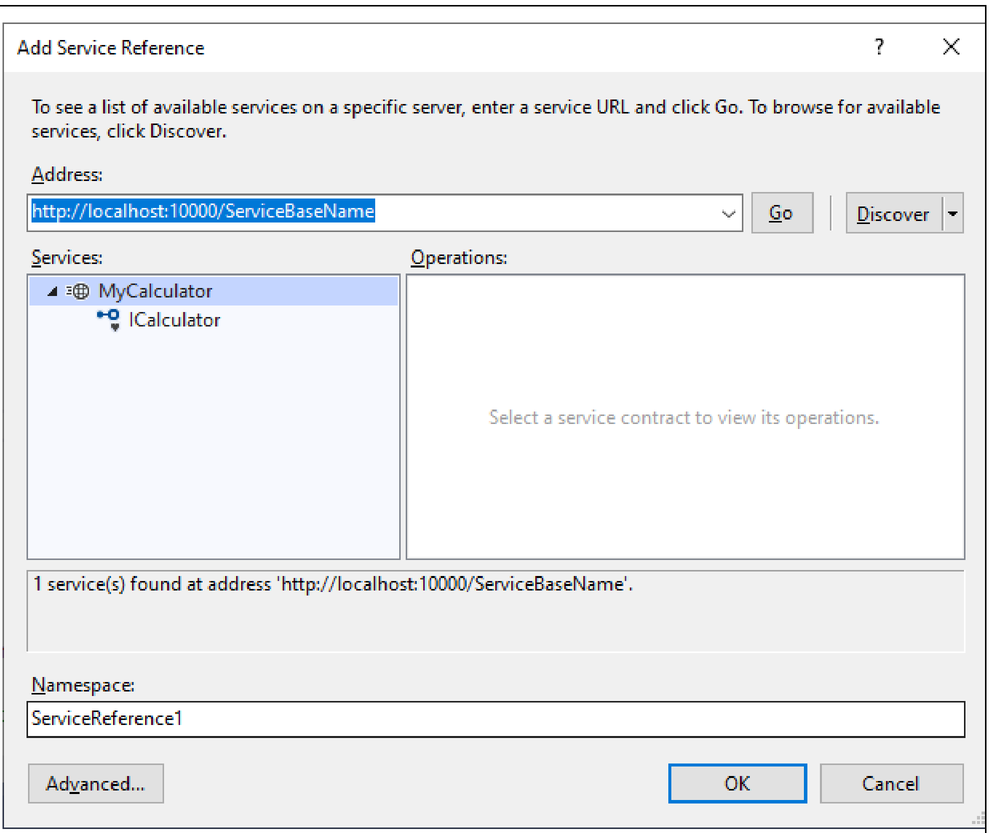
```
WSHttpBindingbinding2 =newWSHttpBinding();
binding2.Security.Mode =
SecurityMode.None;ServiceEndpointendpoint2 =
myHost.AddServiceEndpoint(
                                        typeof(iCalculator),
                                        binding2,"endpoint2");
```

- Then add the code displaying information about endpoints (as below for endpoint 1), duplicating it for endpoint2:

	`Console.WriteLine("\n---> Endpoints:");`
	`Console.WriteLine("\nService endpoint {0}:", endpoint1.Name);Console.WriteLine("Binding: {0}", endpoint1.Binding.ToString());`
	`Console.WriteLine("ListenUri: {0}", endpoint1.ListenUri.ToString());`
	Extra note*:*
	many site elements, including additional endpoints, can also be defined in the App.config host project configuration file.

8.Testing the operation of the service	• Rebuild (Rebuild) the service contract and service host.
	• Start the service from the console and check the operation.
	  ○ Review the data displayed in the host console.
	• Test it with Postman
	  ○ change request address to endpoint2
	  ○ change the Content-Type header to application/soap+xml
	  ○ include a reference to SOAP standards in the envelope and attributes in the header:
	  - **action**– such as soapAction attribute in WSDL
	  - **This**– such as endpoint address:
	`<s:Envelopexmlns:a="http://www.w3.org/2005/08/addressing"`
	`          xmlns:s="http://www.w3.org/2003/05/soap-envelope">`
	`  <s:Header>`
	`    <a:Actions:mustUnderstand="1">http://tempuri.org/ICalculator/Add`
	`    </a:Action>`
	`    <and this>http://localhost:10000/MyService/endpoint2</and this>`
	`  </s:Header>`
	`  <s: Body>[...]</s: Body>`
	`</s:Envelope>`
	  ○ Send a request and check the response.

9. Implementation of the service client - version II - configuring the client's proxy	Creationclient proxy (client proxy) using Visual Studio functions: **Add Service Reference**. • Add a service reference to the defined service in the client's project (Illustration further in the figure): ○ **Start the WCF service hosting application first!** ○ Right-click a folder in Solution Explorer **References**and select Add Service Reference. ○ In the Add Service Reference window, enter the service address (endpoint) in the Address field: **http://localhost:xxx/ServiceBaseName** Replace xxx with the appropriate port number. ○ Press the Go button and an available service should appear on the given access point (see figure next). Selecting a contract (interface) will additionally show available operations (methods). ○ Confirm your selection with the OK button.


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

In the above manner, the client proxy code is generated that performs service calls - the code of the additional application module.

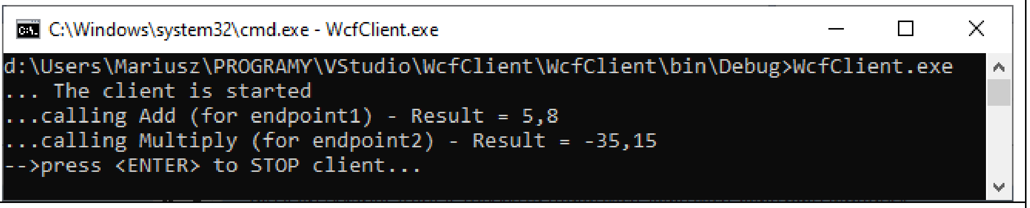
The client configuration is contained in the project's App.config configuration file created by adding a service reference.

- Open the client's App.config file and examine its contents.
- Pay particular attention to the section and name and type of binding (binding), the section and name of the access point (endpoint) and the contract specification (contract).
- Its contents should be similar to the one in the picture.

```
<?xmlversion="1.0"encoding="utf-8"?>
<config>
  <startup> ...</startup>
  <system.serviceModel>
    <bindings>
      <basicHttpBinding>
        <bindingname="BasicHttpBinding_ICalculator"/>
      </basicHttpBinding>
      <wsHttpBinding>
        <bindingname="WSHttpBinding_ICalculator">
          <securitymode="None"/>
        </binding>
      </wsHttpBinding>
    </bindings>
```

	```
	<client>
	  <endpointaddress="http://localhost:10000/MyService/endpoint1"
	    binding="basicHttpBinding"bindingConfiguration="BasicHttpBinding_ICalculator"contracts="ServiceReference1.ICalculator"name="BasicHttpBinding_ICalculator"/>
	  <endpointaddress=http://localhost:10000/MyService/endpoint2binding="wsHttpBinding"bindingConfiguration="WSHttpBinding_ICalculator"contracts="ServiceReference1.ICalculator"name="WSHttpBinding_ICalculator"/>
	</client>
	</system.serviceModel>
	</config>
	```

10. Implementation of the service client - version II -proxy creation and call	• In the client file Program.cs, add the following code:
	○ Creating a client proxy instance (client proxy).
	○ Calling a service operation from the client.
	`CalculatorClient myClient2 =new CalculatorClient("WSHttpBinding_ICalculator"); Console.Write("...calling Multiply (for endpoint2) - "); result = myClient2.Multiply(-3.7, 9.5); //just example valuesConsole.WriteLine("Result = "+ result);`
	The proxy client object (here myClient2) is created according to hints from the service description page:
	– the class name is the service class name plus "Client",
	– however, if the service has more than one access point (endpoint), the constructor must specify one of them,
	– the name of the endpoint from the file is used for the specification **app.config**(here: "WSHttpBinding_ICalculator").
	○ Fix errors by adding imports of appropriate classes including ServiceReference1.

11.Testingapplication operation	• Run the service (the application hosting the WCF service) in one console window, the client in the other console window, and check the results.
	• The effect should be similar to the one below.
	
	C:\Windows\system32\cmd.exe - WcfClient.exe
	`d:\Users\Mariusz\PROGRAMY\VStudio\WcfClient\WcfClient\bin\Debug>WcfClient.exe`
	`... The client is started`
	`...calling Add (for endpoint1) - Result = 5,8`
	`...calling Multiply (for endpoint2) - Result = -35,15`
	`-->press <ENTER> to STOP client...`

12. Asynchronous operations – version I	There are several ways to perform an operation asynchronously. The new version of WCF automatically generates asynchronous methods that return Task<T> according to the ATM (Asynchronous Task Model). They have names with Async added. This is why it is usually recommended to use this approach.

Service:

- Add another HMultiply operation to the contract (interface and implementation) - like Multiply, but with added sleep for 5 seconds. (something like simulation of long calculations (Heavy Multiply)).
- For the service contract, define the behavior to multithread the service instance ConcurencyMode=Multiple.

```
  [ServiceBehavior(InstanceContextMode =InstanceContextMode.Single,
                   ConcurrencyMode =ConcurrencyMode.Multiple)]
public classMyCalculator: ICalculator {[...]}
```

[...]– denotes already existing code fragments.

- Rebuild the site from scratch and run it.

Client:

In the client, we will define a separate method in which we will wait for the result of the previously called asynchronous service operation (which returns a Task<T> promise)

- Update the service reference (so the customer can see changes to the site).
- Define a method in the client that will invoke the asynchronous version of HMultiply - HmultiplyAsync with expectation:

```
static asyncTask<double> callHMultiplyAsy(doublen1,doublen2) {
  Console.WriteLine("2......called callHMultiplyAsync");
  doublereply =waitmyClient2.HMultiplyAsync(n1, n2);
  Console.WriteLine("2......finished
  HMultipleAsync");returnreply;
}
```

- In the Main method add:
  - behind the call to the Multiply method, the method call **callHMultiplyAsync**:

    ```
    Console.WriteLine("2...calling HMultiply ASYNCHRONOUSLY !!!");
    Task<double> asyResult = callHMultiplyAsync(1.1, -3.3);
    ```

  - after this call, add a pause (Thread.Sleep) for about 100ms (to better catch the order of activities), and then another add call (synchronous request).
  - after this call, at the end, before closing the application (proxy clients), add getting the result from the async method and print the result:

    ```
    result = asyResult.Result;
    Console.WriteLine("2...HMultiplyAsync Result = "+ result);
    ```

    Attention: here, if the result is not yet available, the client will be suspended.

- Run the application and check the operation.
  Pay attention to the order in which operations are called

	and the results are printed.

13. Asynchronous operations – version I – preliminary preparation	The first approach, consistent with the SOA pattern of performing operations asynchronously using one-way requests - without a response (One-way) - is to define a Callback Contract - one one-way request calls the operation, the second request (from the service to the client) returns the result . **Service**: The service will be implemented in a separate project. • Add the WCF Library project of the third service (e.g. called CallbackService) to the solution - the project of the second contract. • Add a reference to this project in the host.

14. Defining a service contract with callback operations	Define in the project a new Callback service contract with one operation (method). For this purpose, the following are defined:
	- OneWay operations,
	- behavior of the CallbackContract service specifying the type of callback interface (here: we specify it as ISuperCalcCallback) – this is the client interface for callback handling – this interface must be implemented in the client,
	- behavior can be defined as an attribute of the service contract (v **[ServiceContract]**),
	- additionally, we will specify the requirement for the service instance to run within the session.
	• Define the ISuperCalc service contract interface code containing the Factorial (callback) asynchronous (callback) method/operation, additionally defining the CallbackContract service attribute and requiring the session mode:
	`[ServiceContract(SessionMode =SessionMode.required,`
	`                    CallbackContract=typeof(ISuperCalcCallback))]public interfaceISuperCalc`
	`{`
	`  [OperationContract(IsOneWay =true)]voidFactorial(doublen);`
	`  [OperationContract(IsOneWay =true)]voidDoSomething(intseconds);`
	`}`
	• In this file, also define the ISuperCalcCallback interface containing a description of the methods called at the client in order to transfer the results of the Factorial operation - here containing the FactorialResult method for the factorial calculation result. Add a second interface in the same file:
	`public interfaceISuperCalcCallback{`
	`  [OperationContract(IsOneWay =true)]voidFactorialResult(doubleresult);`
	`}`

15. Implementation service contract	Implement contract - a class that implements each of the required ones methods of the ISuperCalc interface.
	• In the Service1.cs file. enter the MySuperCalc class code implementing the ISuperCalc interface.
	o Behavior is also defined for the service **InstanceContextMode=PerSession**meaning creation object instance (service instance) for each session.
	o The handler for the callback is taken in the constructor.
	`[ServiceBehavior(InstanceContextMode =InstanceContextMode.PerSession,`
	`ConcurrencyMode =ConcurrencyMode.Multiple)]`
	`public classMySuperCalc:ISuperCalc{`
	`  doubleresult;`
	`  ISuperCalcCallbackcall back =null;`
	`  publicMySuperCalc() {`
	`    call back =OperationContext.Current.GetCallbackChannel`
	`              <ISuperCalcCallback>();`
	`  }`
	`  public voidFactorial(doublen) {`
	`    Console.WriteLine("...called Factorial({0})", n);`
	`    Thread.Sleep(1000);`
	`    result = 1;`
	`    for(inti = 1; i <= n; i++ )`
	`      result *= i;`
	`    callback.FactoryResult(result);`
	`  }`
	`}`
	o Finally, we call the callback method in the client.

16. Expansion host for third service	Add in the code of the application hosting the launch of the second website.
	• In the Program.cs file, add the code where appropriate performing the following functions:
	o Create a URI with the base address of the second site.
	o Create a second site host object.
	o Adding an endpoint with a WSDualHttpBinding.
	o Define site metadata.
	o Launching a second site.
	[...]denotes existing code snippets.
	`static voidmain(string[] args) {`
	`  [...]`
	`  UribaseAddress3`
	`  =newUri(...);ServiceHostmyHost3`
	`  =new`
	`     ServiceHost(typeof(MySuperCalc),`
	`  baseAddress3);WSDualHttpBindingmyBinding3`
	`  =newWSDualHttpBinding();ServiceEndpointendpoint3 =`
	`     myHost3.AddServiceEndpoint(typeof(ISuperCalc),`
	`                               myBinding3,"endpoint3");`
	`  myHost3.Description.Behaviors.Add(smb);`
	`  try{`

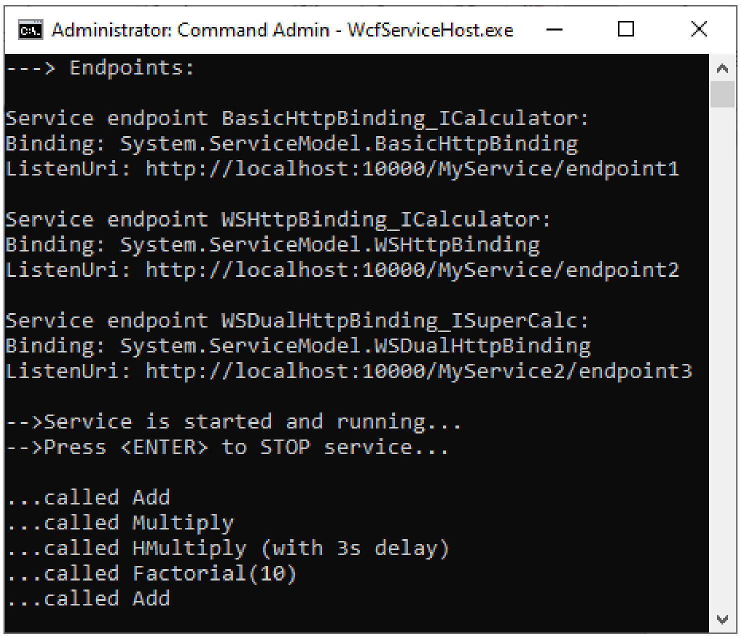
	`[...]`

	```
	     myHost3.Open();
	    Console.WriteLine("--> Service SuperCalc is running.");
	     [...]
	     myHost3.Close();
	   }catch(CommunicationExceptionce) {
	     [...]
	     myHost3.Abort();
	   }
	 }
	```
	 Start the service (host) from the console and check the operation.

17.Expansion of the client to use the second service	• Add a service reference to the second service in the client:Note: remember to run the service hosting app first!
	• Add a new class to the client's project (here called SuperCalcCallback) in which operations called back by the service will be defined to send back the results of its service operations.
	```
	     classSuperCalcCallback:ISuperCalcCallback{
	     public voidFactorialResult(doubleresult) {
	     //here the result is
	     consumedConsole.WriteLine(" Factorial = {0}",
	     result);
	   }
	 }
	```
	• Open the Program.cs file and add code in the Main function to do the following:
	  ○ Creation of a handle object (handler) with the operations of receiving results from the service.
	  ○ Creating a proxy client instance.
	  ○ Calling a service operation from the client (proxy).
	  ○ Client closures
	Add this code after calling the callHMultiplyAsync method. Receiving and writing results will be asynchronous - initiated by the service.
	```
	static voidmain(string[] args) {
	  [...]
	  SuperCalcCallbackmyCbHandler
	  =newSuperCalcCallback();InstanceContextinstanceContext =
	                                 new
	InstanceContext(myCbHandler);SuperCalcClien
	  tmyClient3 =new
	SuperCalcClient(instanceContext);dou
	  blevalue1 = 10;
	  Console.WriteLine("...calling Factorial({0})...", value1);
	  myClient3.Factory(value1);
	  [...]
	  client3.Close();Console.WriteLine("CLIENT
	  3 - STOP");
	}
	```

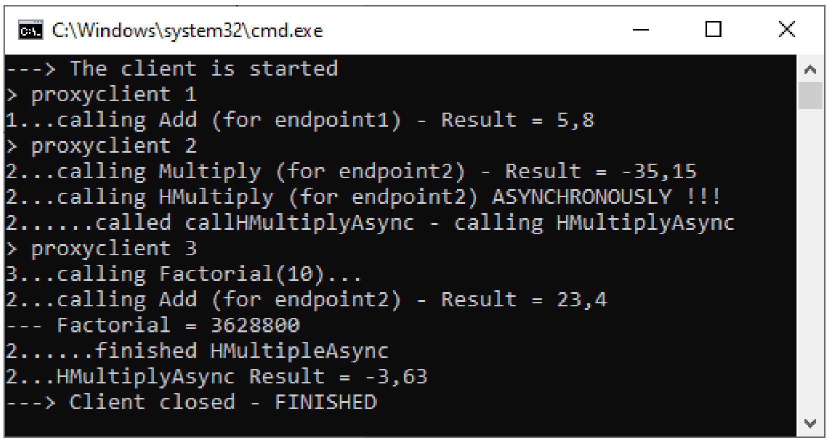
18.Testingapplication operation

- Start the service (service hosting application) in one console window.
- Start the client in a second console window.
- Check the operating results. Pay attention to the times of service and customer operation.
- The final output should be similar to the following: Host window:



```
Administrator: Command Admin - WcfServiceHost.exe
---> Endpoints:

Service endpoint BasicHttpBinding_ICalculator:
Binding: System.ServiceModel.BasicHttpBinding
ListenUri: http://localhost:10000/MyService/endpoint1

Service endpoint WSHttpBinding_ICalculator:
Binding: System.ServiceModel.WSHttpBinding
ListenUri: http://localhost:10000/MyService/endpoint2

Service endpoint WSDualHttpBinding_ISuperCalc:
Binding: System.ServiceModel.WSDualHttpBinding
ListenUri: http://localhost:10000/MyService2/endpoint3

-->Service is started and running...
-->Press <ENTER> to STOP service...

...called Add
...called Multiply
...called HMultiply (with 3s delay)
...called Factorial(10)
...called Add
```

client window:



```
C:\Windows\system32\cmd.exe
---> The client is started
> proxyclient 1
1...calling Add (for endpoint1) - Result = 5,8
> proxyclient 2
2...calling Multiply (for endpoint2) - Result = -35,15
2...calling HMultiply (for endpoint2) ASYNCHRONOUSLY !!!
2......called callHMultiplyAsync - calling HMultiplyAsync
> proxyclient 3
3...calling Factorial(10)...
2...calling Add (for endpoint2) - Result = 23,4
--- Factorial = 3628800
2......finished HMultipleAsync
2...HMultiplyAsync Result = -3,63
---> Client closed - FINISHED
```

3 Task - part II

A. Practice the technique of creating WCF services and clients according to the manual.
1. Defining, configuring and implementing contracts.
2. Create a service host. Defining endpoints (endpoint).
3. Creating a client, binding and invoking service operations.
4. Asynchronous operations according to the ATM and CallbackContract models.

B. Prepare to write an application with similar functionalities or modify the application during classes. according to the instructor's instructions.