IRA A. FULTON SCHOOLS OF engineering

school of computing, informatics, & decision systems engineering

CSE 446/598

Software Integration and Engineering

Unit 1

Service Standards and Service Development

Lecture 1-2
Raw Services and Self-Hosting

Dr. Yinong Chen

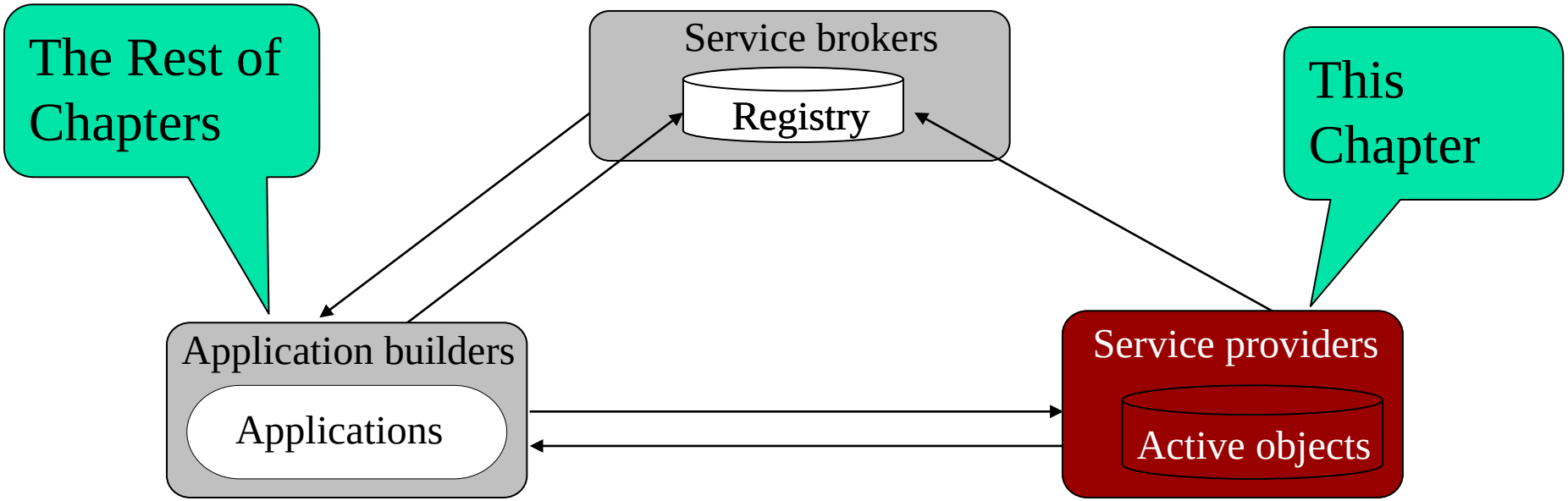
https://myasucourses.asu.edu/

Lecture Outline

- WCF Basic Service Development and Consuming (Review)
- Raw Service Development and Self-Hosting
  - Raw Service development: Develop the interface without using the service tools
  - Host (worker process) development
  - Endpoint creation
  - Proxy creation
- Different Clients Consuming Self-Hosting Services

Model of Service-Oriented Software Development

Three-Party Development Model

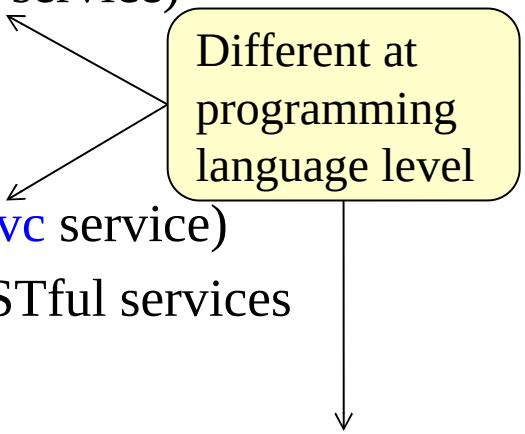


Service/Application Development Environments

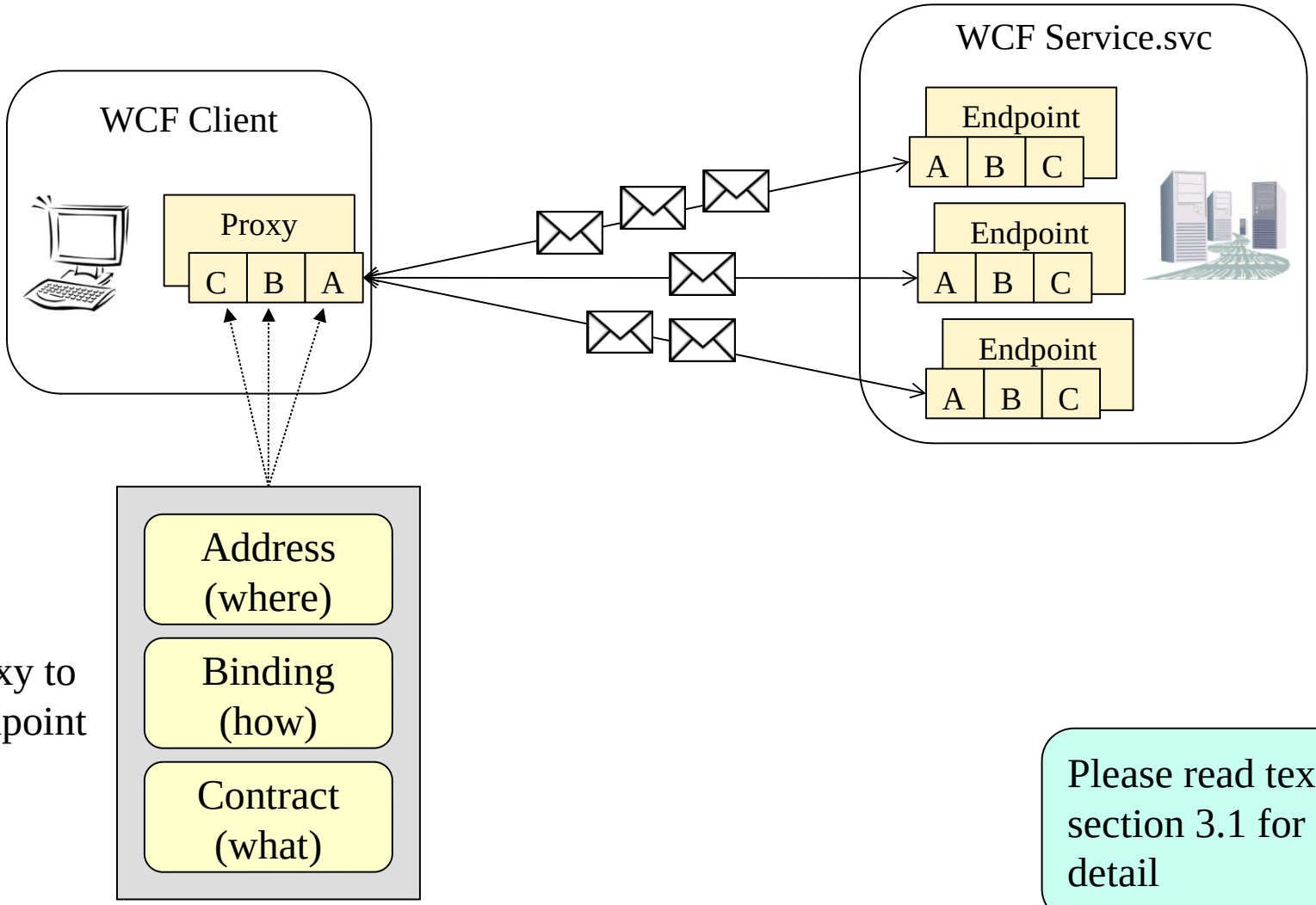
- **ASP.NET and Windows Communication Foundation (WCF)** (Textbook Chapter 3)
- Workflow Foundation (Text Chapter 7)
- Oracle SOA Suite/ JDeveloper (Text Chapter 8)
- Java-based SOA Development: Netbeans, Eclipse/Axis/Tomcat
- JAX-RPC 1.1 API
- J2EE
- BEA WebLogic
- IBM WebSphere
- SAP Enterprise Service Architecture (ESA) Development Framework

WCF Service Development and Consuming

- ASP .Net is a component in Visual Studio for service and application development.
  - It can be used for service development (.asmx service)
  - It is mainly used for application development
- WCF is a component in Visual Studio
  - WCF is used for service development only (.svc service)
  - It is used for developing SOAP as well as RESTful services
  - WCF service hosting: with or without IIS
- Service Interfaces: Both .asmx and .svc services use WSDL
- WCF service consuming
  - Console application
  - ASP .Net Windows forms application (Desktop application)
  - ASP .Net Website (Web application) / HTML5 Website
  - Workflow Foundation
  - BPEL, and many others



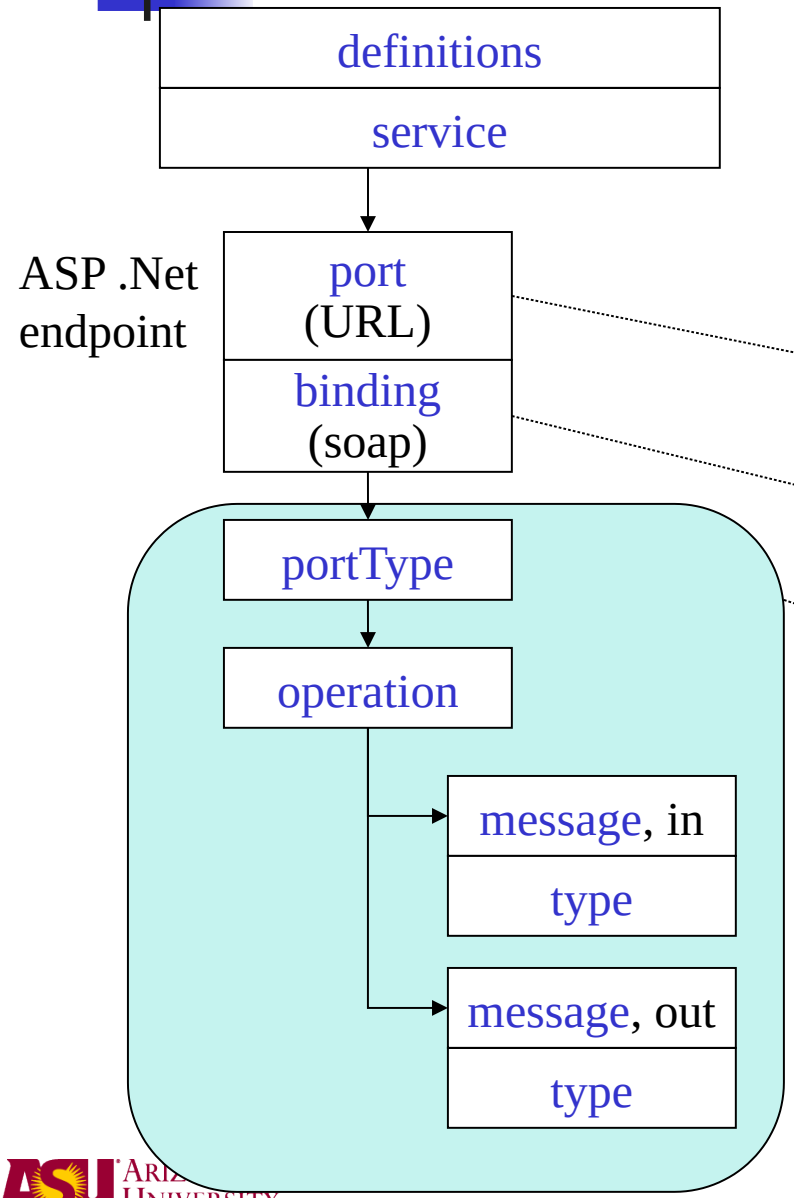
WCF Interface between Client and Service



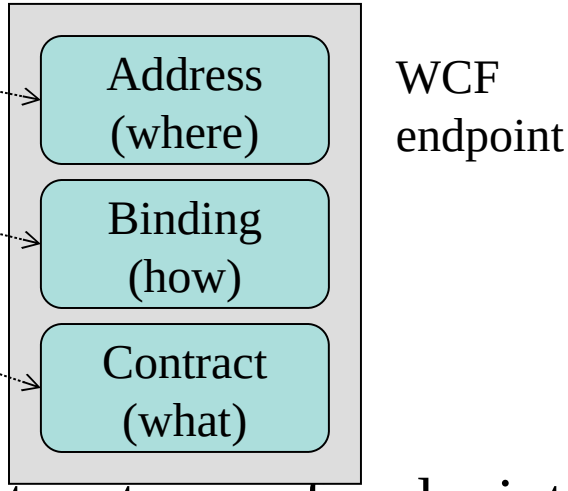
Proxy to Endpoint

Please read text section 3.1 for more detail

Endpoint Coded in WSDL



- An ASP .Net endpoint is the port to the method exposed through a markup of contract in program
- WCF endpoint contains ABC:

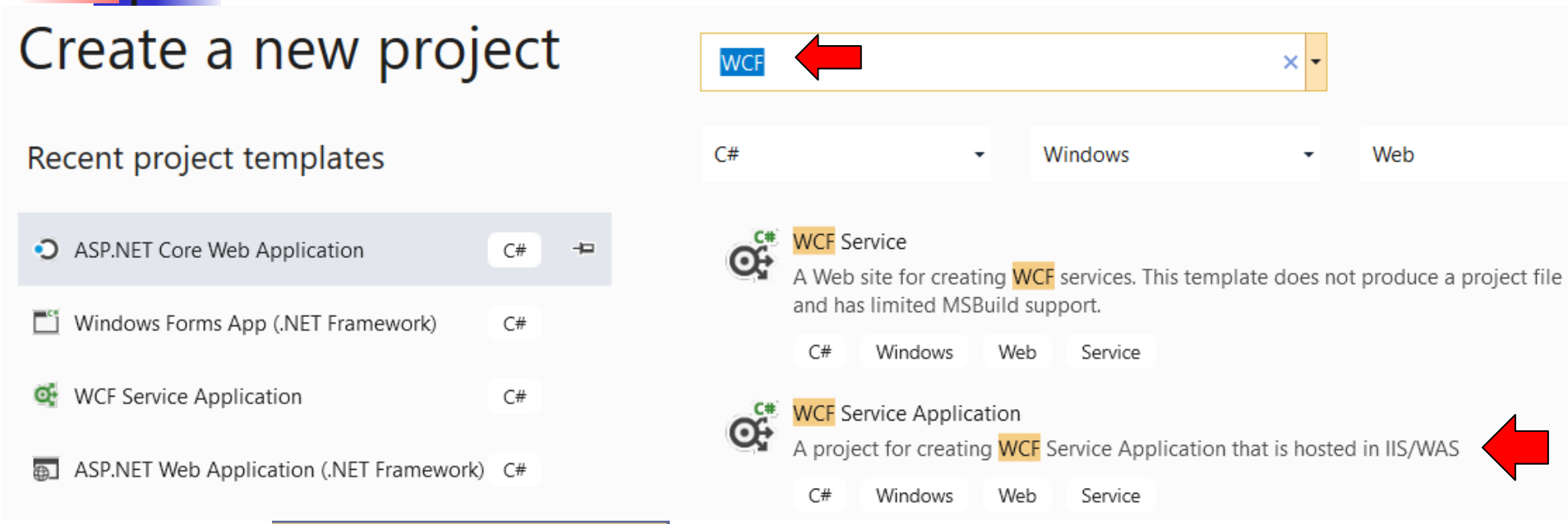


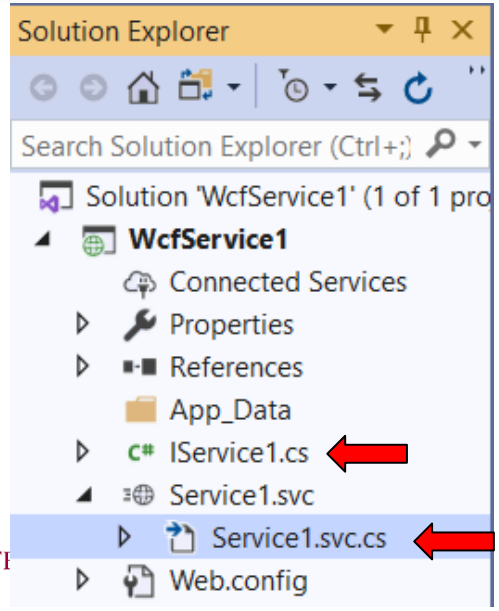
- The two types of endpoints (.asmx and.svc) have different structures, but contain the same basic information
- Both are coded in WSDL

Developing WCF Basic Services & Clients

- Developing a Service using WCF tools (Review)
  - Developing a WCF service using template
  - Hosting the service using IIS Express Server
  - Developing a console application to consume the WCF service
- Developing WCF Service with Self-Hosting

Develop Web Services Using WCF

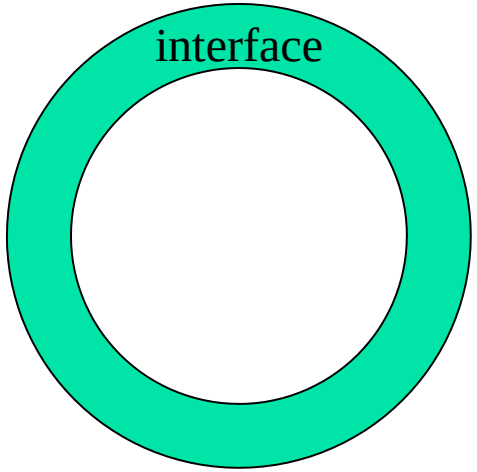




Text Chapter 3, section 3.2.1
We did this exercise in CSE445 Assignment 1

Y. Chen

Step 2: Define Service Contract / Interface



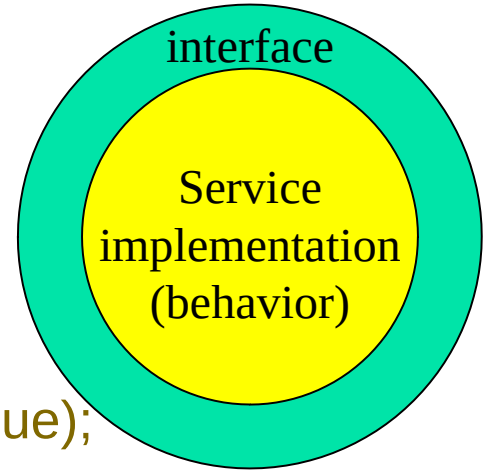
1

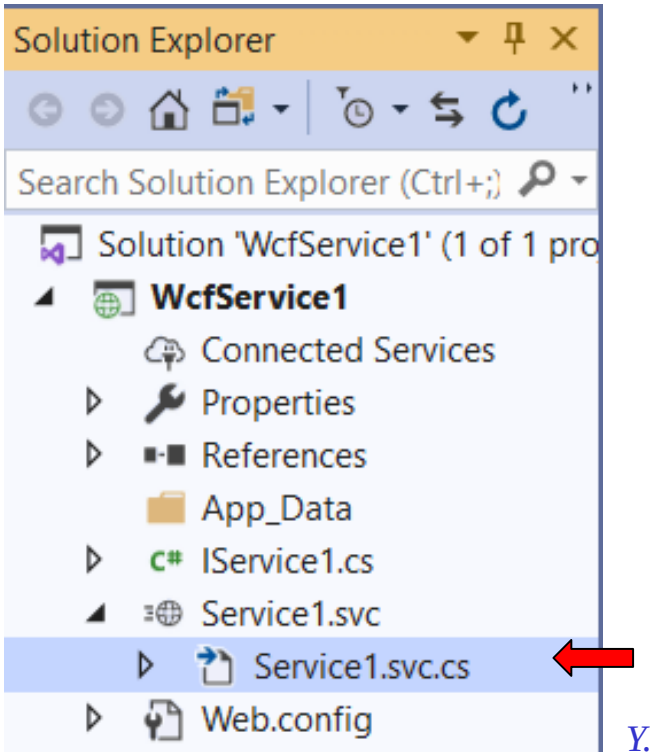
```
// IService.cs file
using System.ServiceModel;

[ServiceContract]
public interface IService {
    [OperationContract]
    string GetData(int value);    // generated code, rename it

    [OperationContract]
    double PiValue();    // my operation 1

    [OperationContract]
    int absValue(int intVal);      // my operation 2
}
```

Step 3: Implement Service Contract (behavior)

2

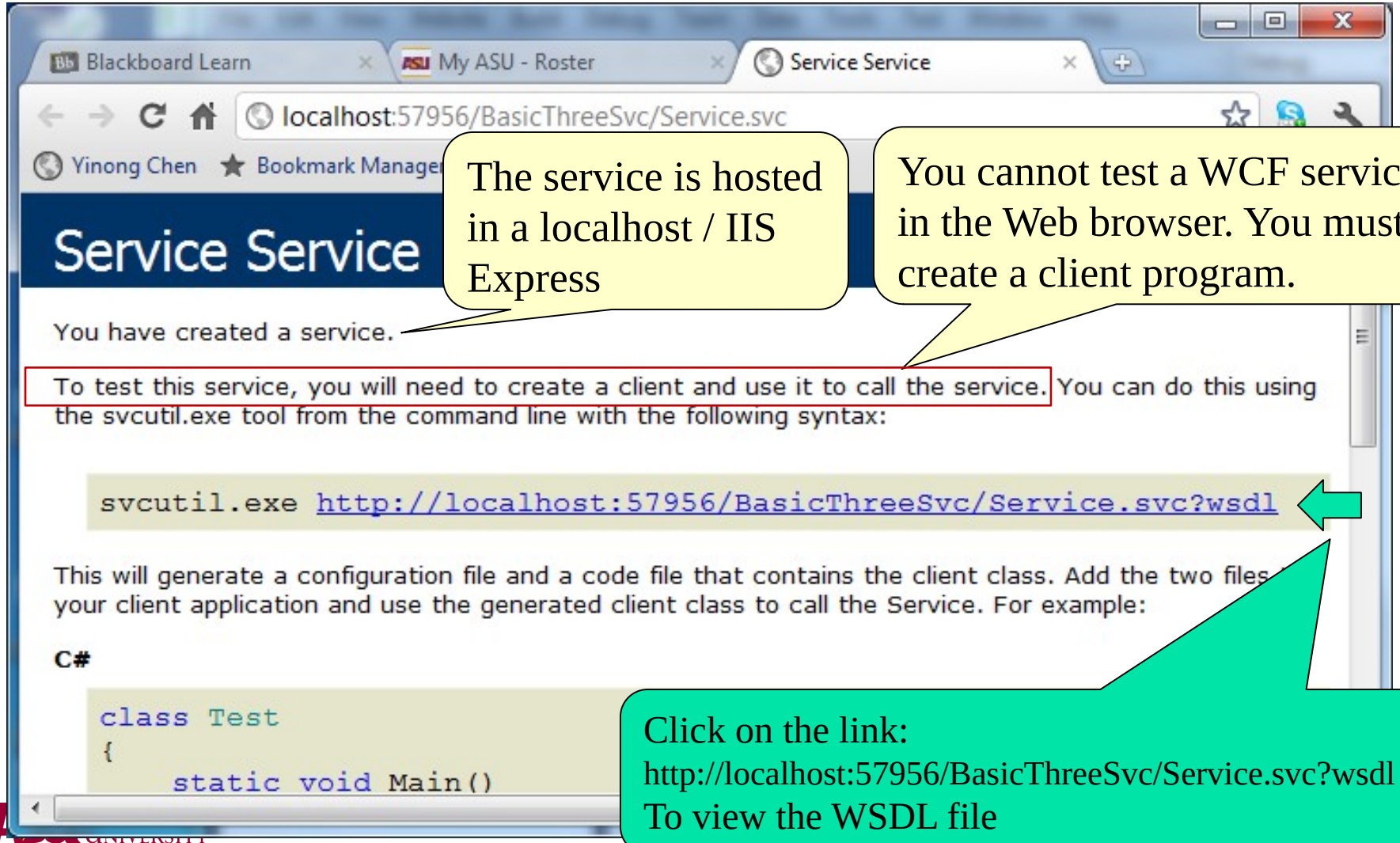
```
// Service.svc.cs file
using System.ServiceModel;

public class Service : Iservice {
    public string GetData(int value) {
        return string.Format("You entered: {0}", value);
    }
    public double PiValue() {
        double pi = System.Math.PI;
            return (pi);
    }
    public int absValue(int x) {
        if (x >= 0) return (x);
            else return (-x);
    }
}
```





Step 4: Build and View in Browser

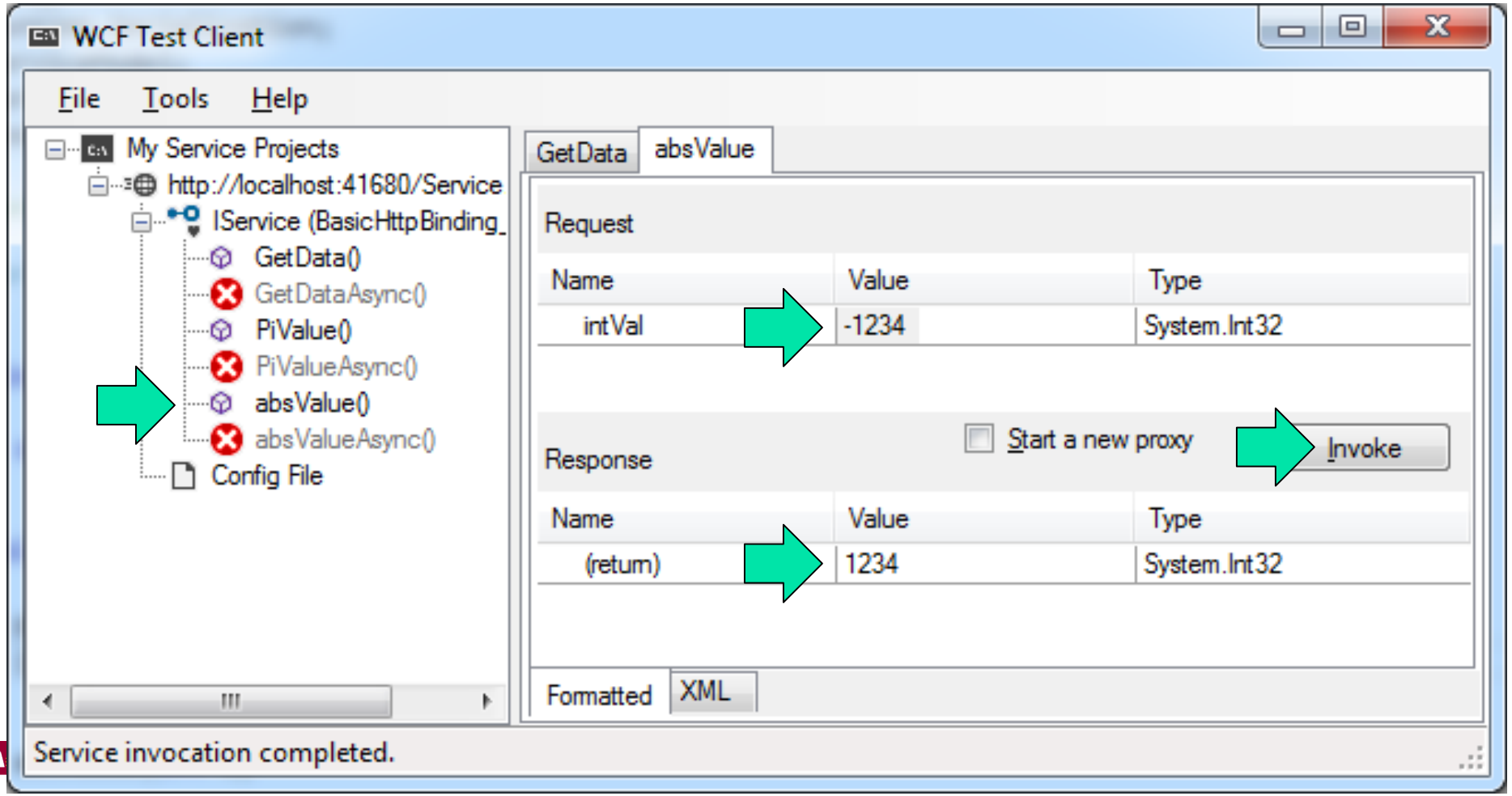
Right-click file Service.svc and choose “View in Browser”



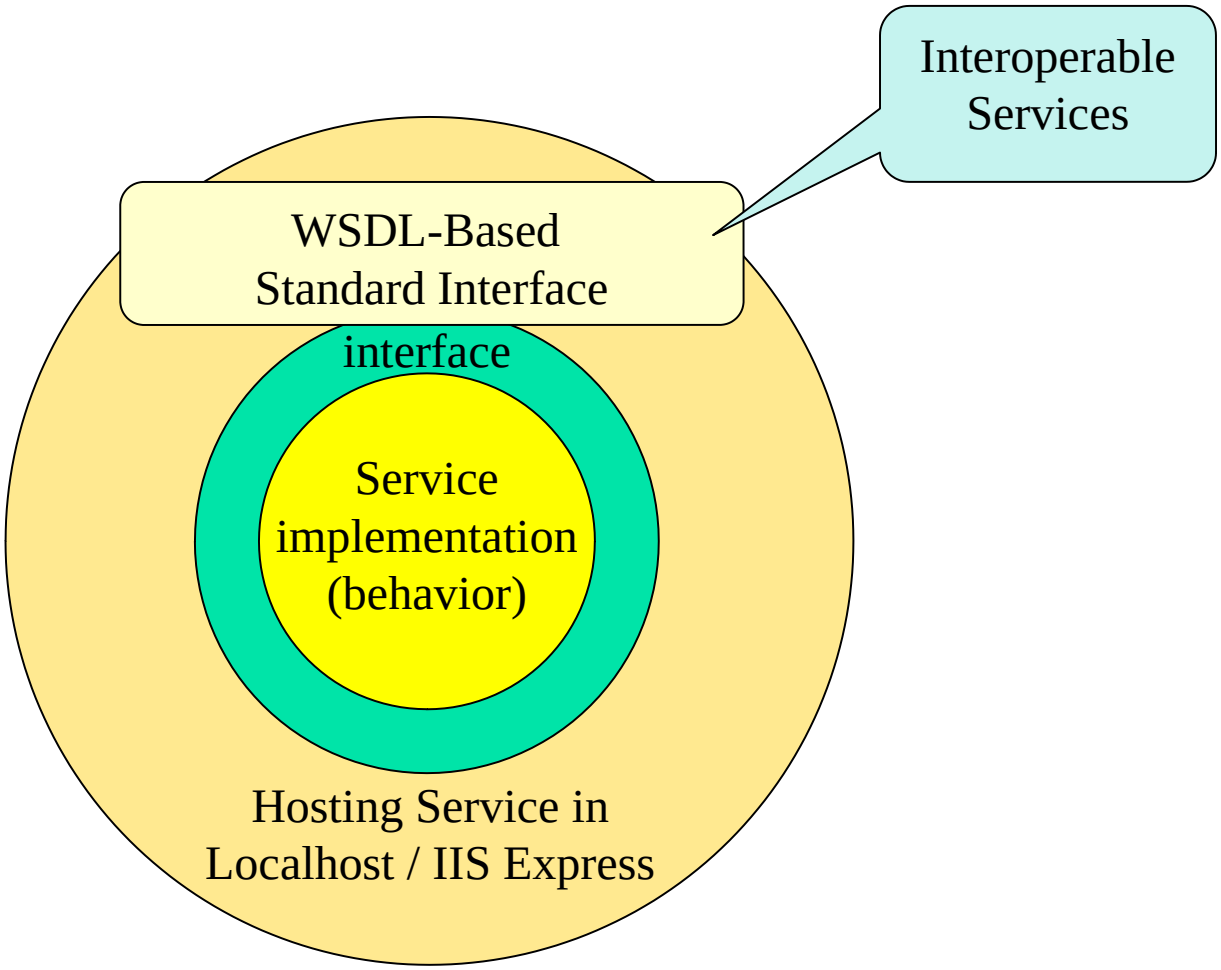
Built-in Test Client

Select the service
Menu: DEBUG Start Without Debugging
The Test Client Window will be open:

Read text for other ways of starting the Test Client: From CMD

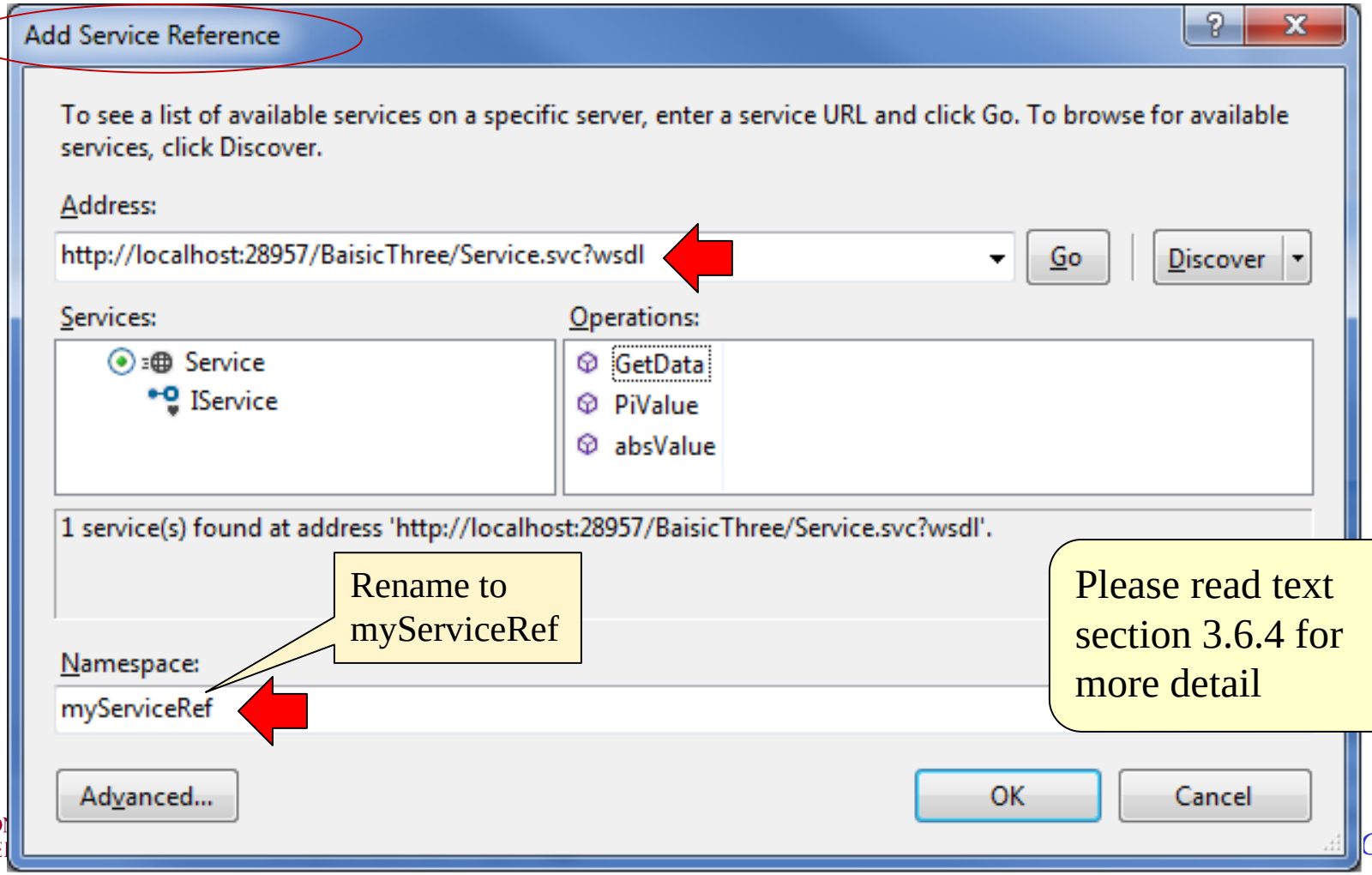


Hosting Using Localhost / IIS Express



Step 5: Create a Client to Test the Service

- Add a Console Application into the same solution
- Add Service Reference: `http://localhost:28957/BaisicThree/Service.svc?wsdl`

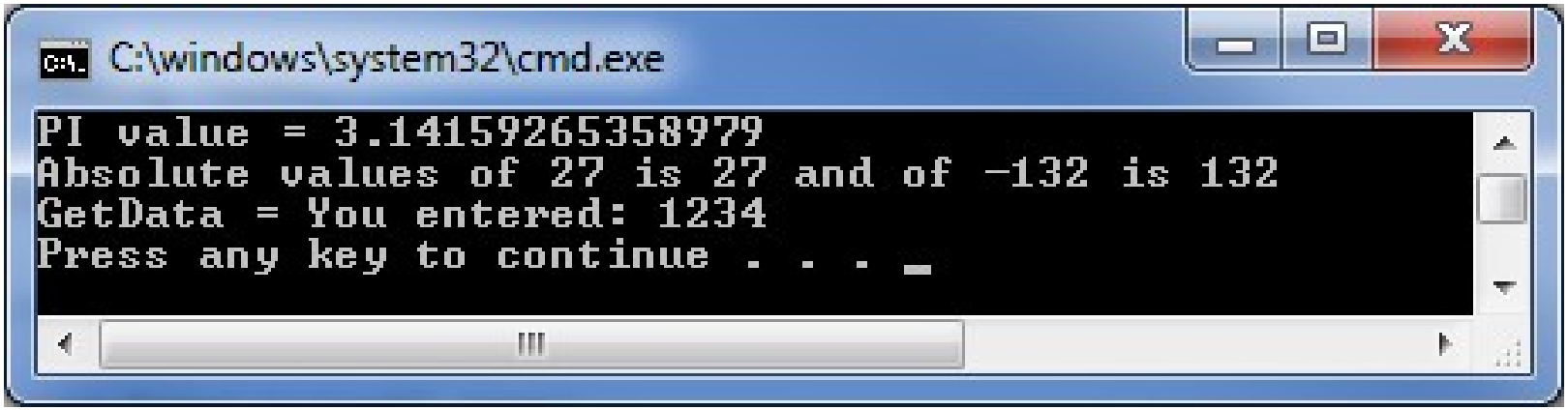


Step 6: Write Client Code to Invoke Service

```
static void Main(string[] args) {   // console application
    myServiceRef.ServiceClient myProxy = new myServiceRef.ServiceClient();
        double pi = myProxy.PiValue();  // call PiValue operation
        Int32 test1 = 27;   // Create test input 1
        Int32 test2 = -132; // Create test input 2
        Int32 result1 = myProxy.absValue(test1); // call operation
        Int32 result2 = myProxy.absValue(test2); // call operation
        Console.WriteLine("PI value = {0}", pi);
        Console.WriteLine("Absolute values of {0} is {1} and of {2} is {3}", test1,
result1, test2, result2);
        Console.WriteLine("GetData = {0}", myProxy.GetData(1234));
        // Close the proxy, the channel to the service
        myProxy.Close();
}
```

Step 7: Test the WCF Client and Service

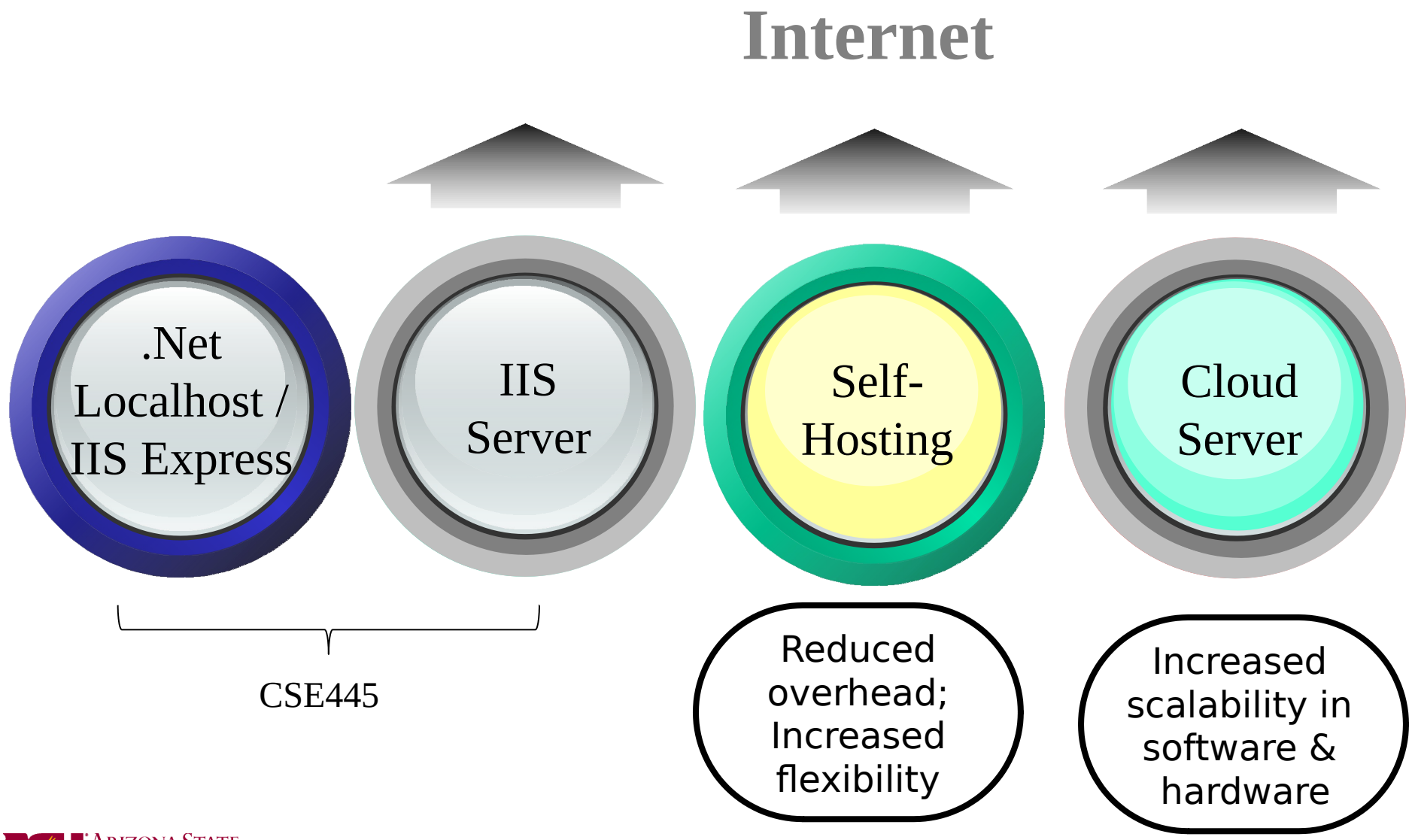
- First, start the service by right-click the **Service.svc** and choose “View in Browser” → Keep service running
- Right-click the **client** project and choose: Set as StartUp Project
- Start without Debugging



Developing Self-Hosting Services and Clients

- WCF Basic Service Development (Review)
  - The process is largely automated
- Developing a Raw Service and Self-Hosting
  - Developing raw service without using template tools;
  - Developing the hosting service;
  - Creating endpoint and proxy
  - Consuming the service in a command line client;
  - Consuming the service in ASP .Net website

Different Ways of Hosting WCF Services



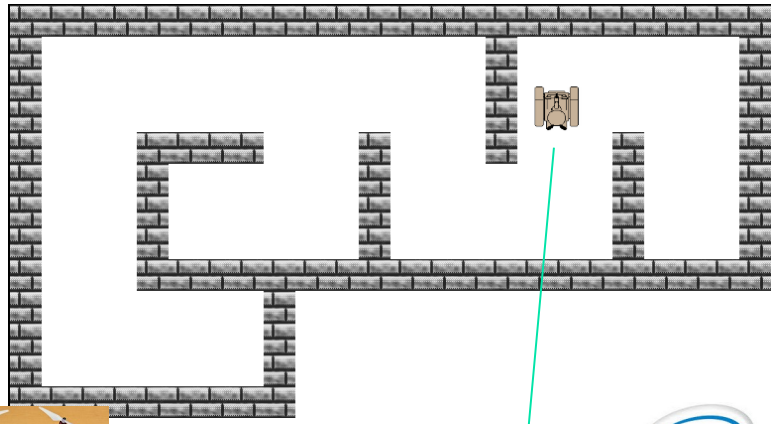
Reasons of Using Self-Hosting

- Reduced overhead for efficiency
  - IIS (Internet Information Services) and WAS (Windows Process Activation Services) are designed to deal with all possible scenarios of service hosting
  - Self-hosting can avoid many overheads
- Increased flexibility/controllability
  - IIS and WAS decide when a service is open (started) and closed (terminated).
  - With self-hosting, you can decide when to start and when to close a service.

Case Study of Using Self-Hosting Service for IoT and Robotics Apps

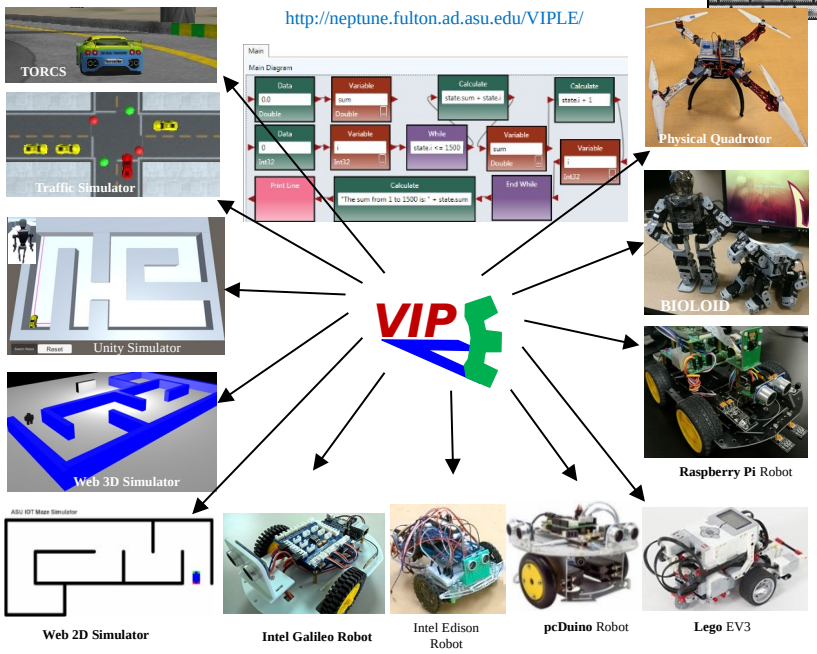
http://neptune.fulton.ad.asu.edu/WSRepository/apps/RaaS/main/

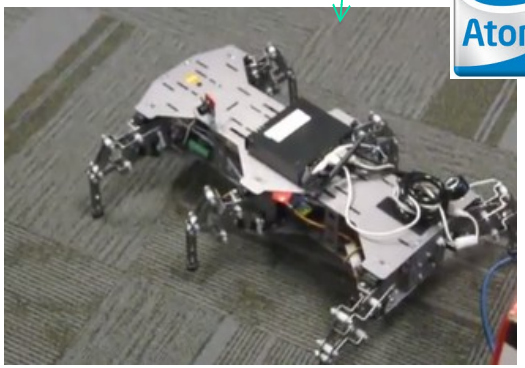
Web Simulation Environment



Visual IoT/Robotics Programming Language Environment

http://neptune.fulton.ad.asu.edu/VIPLE/



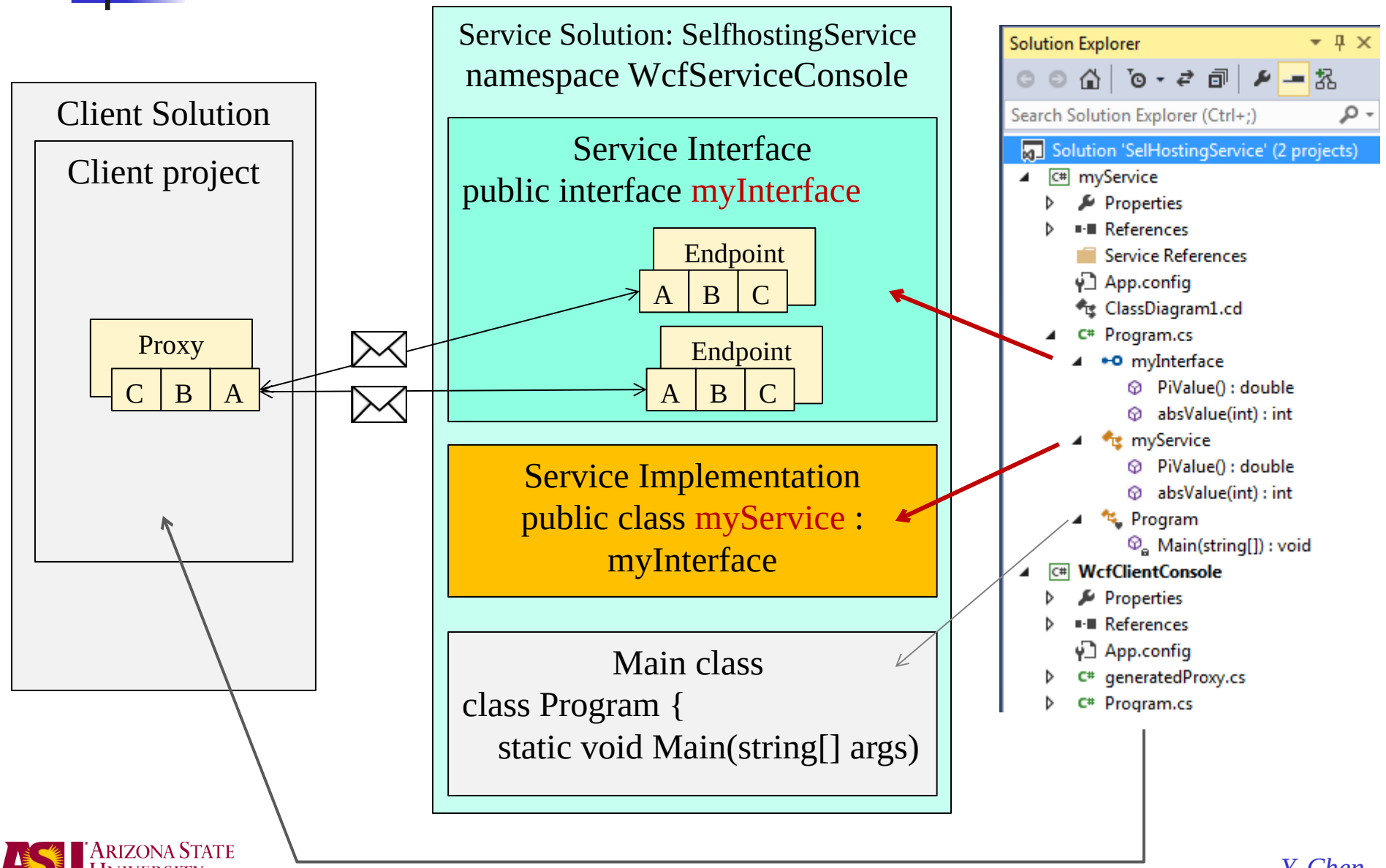


Robot as a Service at ASU, with self-hosting service

Self-Hosting Using Multithreading

- Write your own Worker process, which can run in an infinite loop, or until a condition is met;
- Write your service methods
- Worker process activates each service method as a thread – active object (design pattern)
- Without implementing service management, one client can call the service at a time;
- To implement multiple accesses at the same time, implement service queue, as discussed in Text chapter 2.

Self-Hosting Project Overview



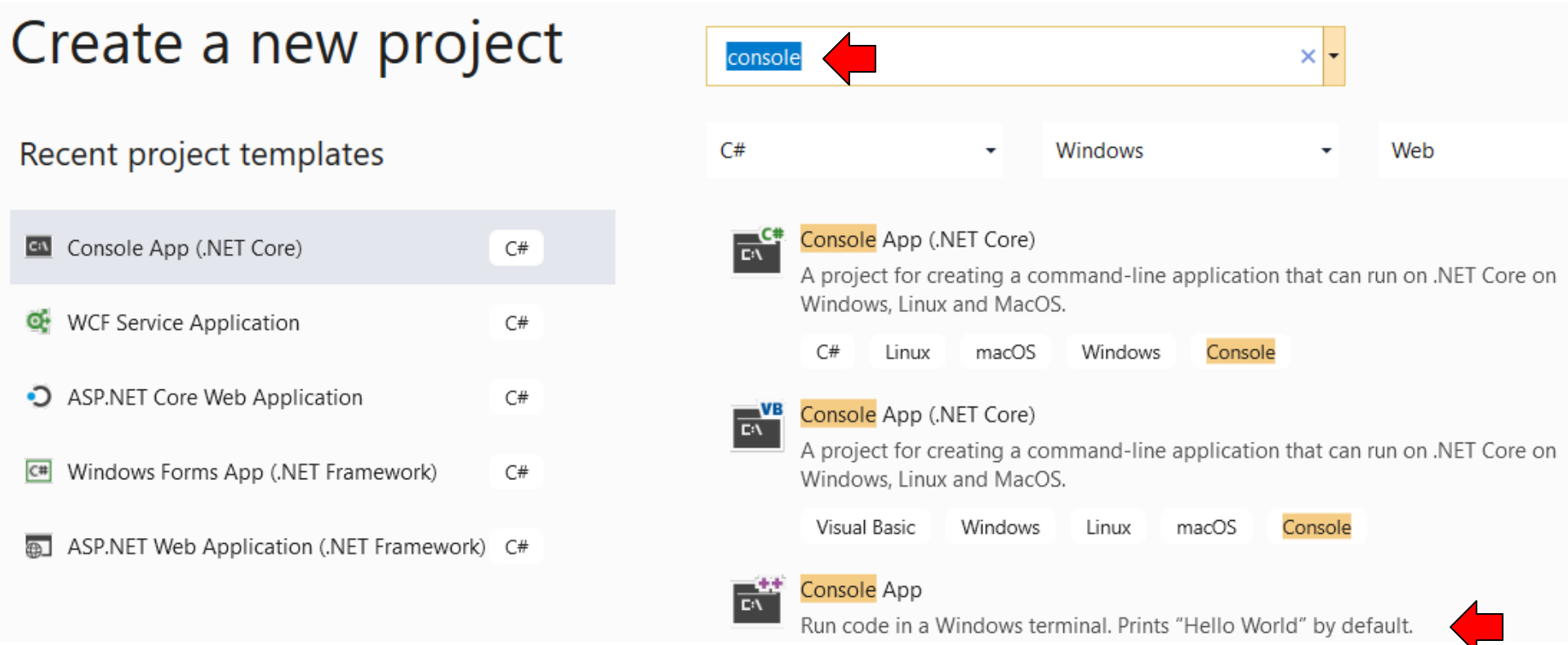
Developing Services with Self-Hosting

Reading: Text Section 7.1

1. Start Visual Studio as an administrator: right-clicking the program in the Start menu and selecting "Run as administrator"; It is necessary as the service needs to register an HTTP channel for client to access;
2. Create a new Solution, with a new Project of Console Application type and name the service "myService";
3. Create Contract Through Interface;
4. Implement the service behavior in C#
5. **Adding Hosting Service and Endpoint**

Starting a New Solution of Console Application Type

We use console application to create a service



Console application is a basic template without advanced tools

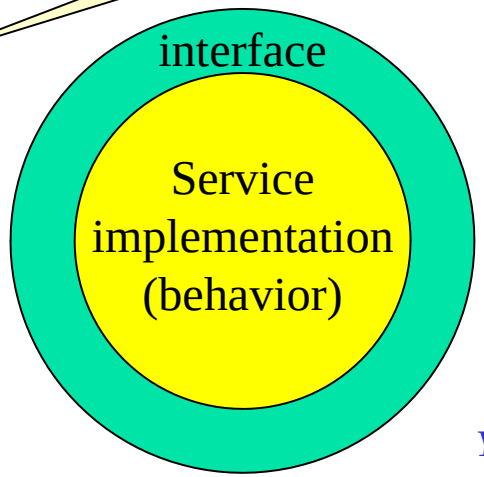
Service Code in the Console Application: Interface and Implementation

```
using System;
using System.ServiceModel;
using System.ServiceModel.Description;
namespace WcfServiceConsole {
    [ServiceContract]
    public interface myInterface { // Use interface to define contracts
        [OperationContract] double PiValue();
        [OperationContract] int absValue(int intVal);
    }
    public class myService : myInterface  // implementation of interface
    {     public double PiValue() {
                double pi = System.Math.PI;  return (pi);
          }
          public int absValue(int x) {
                if (x >= 0) return (x); else return (-x);
          }
    }  // not yet complete, to add hosting code next
```

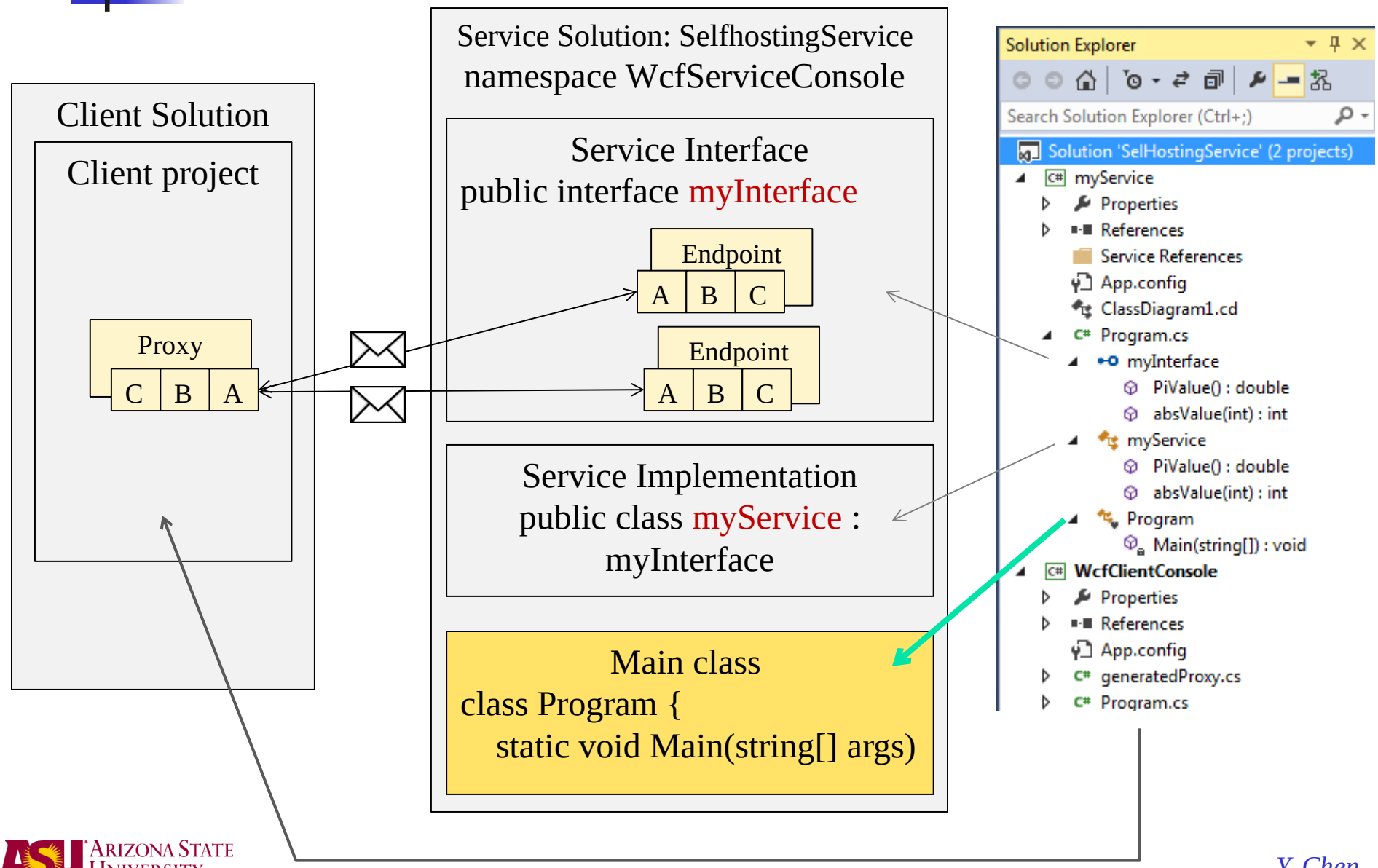
Service Contract is based on interface definition

Operation Contract is based on method

Implementation



Project Overview



Create Hosting Service and Endpoint

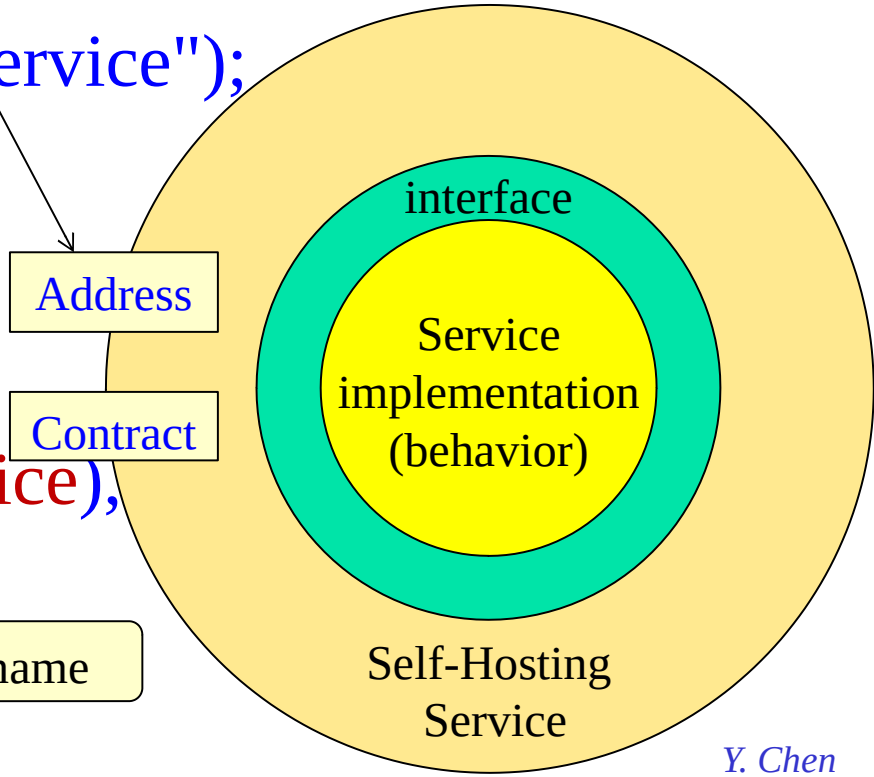
In the Console Application, add in the Main method;

1. Using Uri library class to create a URI instance to myService as the base address;

```
Uri baseAddress = new
    Uri("http://localhost:8000/Service");
```

2. Create a new ServiceHost instance to host the service

```
ServiceHost selfHost = new
    ServiceHost(typeof(myService),
    baseAddress);
```



Create Hosting Service and Endpoint

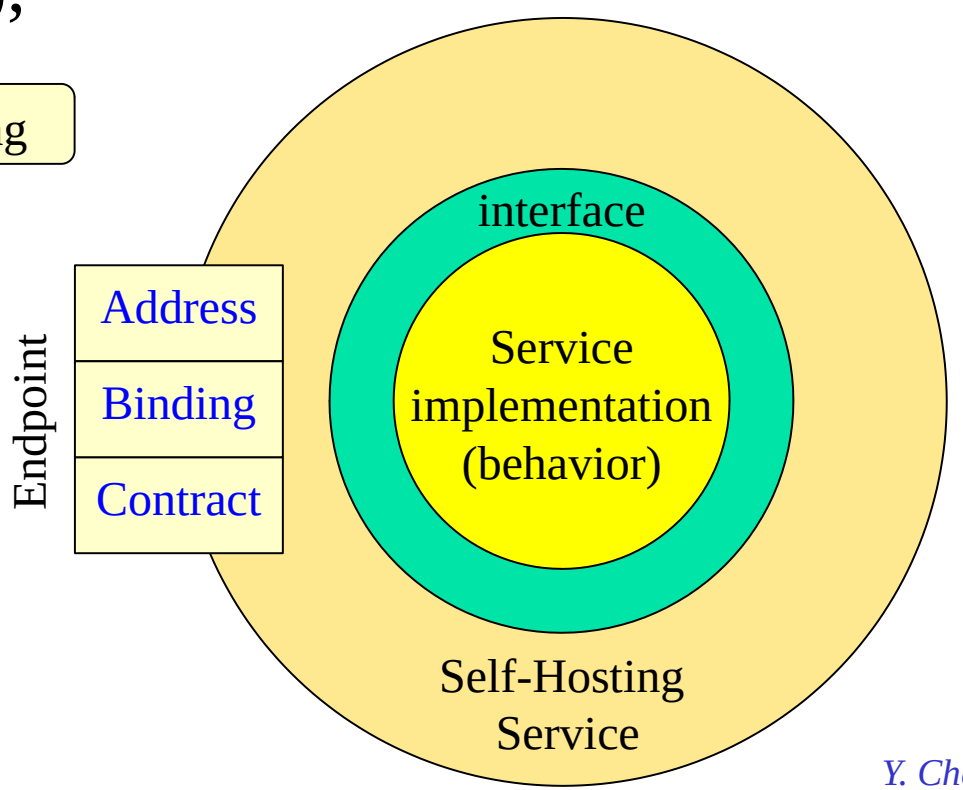
3. Add contract (myInterface) and Binding (WsHttpBinding)

```
selfHost.AddServiceEndpoint(typeof(myInterface),
    new WsHttpBinding(),
    "myService");
```

Contract

binding

behavior

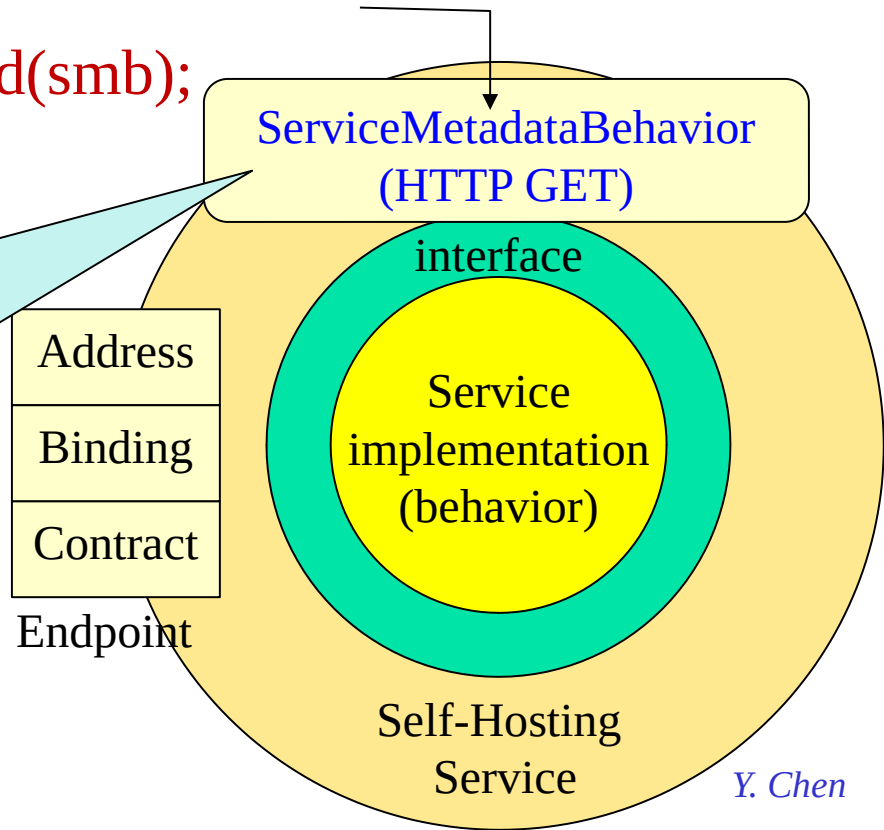


Adding HTTP GET for Public Access

4. Add metadata for public access

```
System.ServiceModel.Description.ServiceMetadataBehavior
smb = new
System.ServiceModel.Description.ServiceMetadataBehavior( );
smb.HttpGetEnabled = true;
selfHost.Description.Behaviors.Add(smb);
```

A different class in the namespace System.ServiceModel.Description will result in a different interface for a different kind of clients to access. As both SOAP/WSDL services and RESTful services use HTTP GET, the interface can potentially support both types of services.



Public Properties of ServiceMetadataBehavior Class

Property Name	Description
ExternalMetadataLocation	Gets or sets a value that is the location of service metadata
HttpGetEnabled	Gets or sets a value that indicates whether to publish service metadata for retrieval using an HTTP/GET request.
HttpGetUrl	Gets or sets the location of metadata publication for HTTP/GET requests.
HttpsGetEnabled	Gets or sets a value that indicates whether to publish service metadata for retrieval using an HTTPS/GET request.
HttpsGetUrl	Gets or sets the location of metadata publication for HTTPS/GET requests.
MetadataExporter	Gets or sets the internal MetadataExporter object used to publish the service metadata.

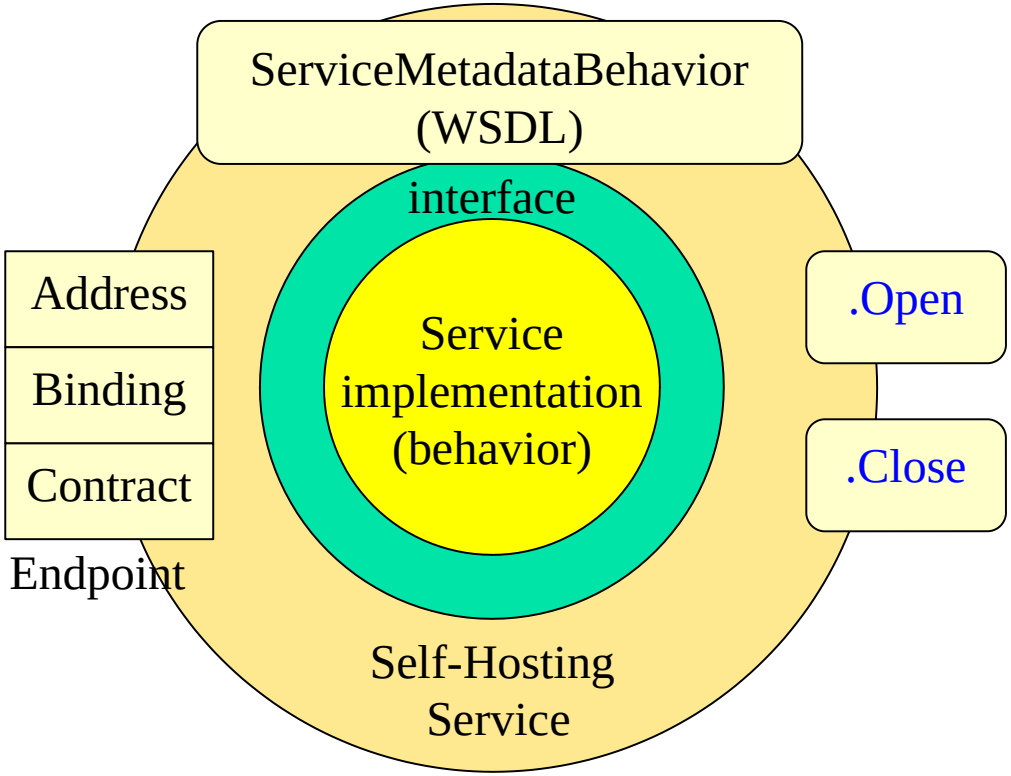
Starting and Terminating the Service

5. Start service and waiting for request.

   selfHost.Open();

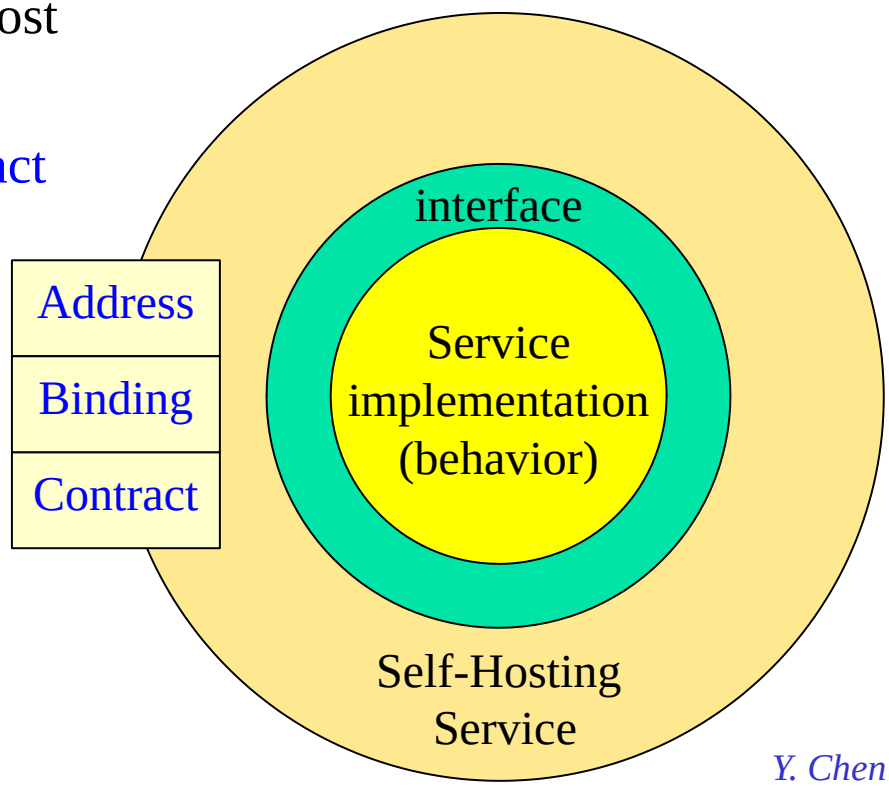
6. Close the ServiceHostBase to shutdown the service.

   selfHost.Close();



Code Together: Hosting Part

```
class Program { // more detail see text section 7.1
    static void Main(string[] args) {
        // (1) Create a URI instance to myService as the base address.
        Uri baseAddress = new Uri("http://localhost:8000/Service");
        // (2) Create a new ServiceHost instance to host the service
        ServiceHost selfHost = new ServiceHost
                (typeof(myService),
                baseAddress);  // (3) Add contract
        try {
          // (4) Add binding
            selfHost.AddServiceEndpoint(
                typeof(myInterface),
                new WSHttpBinding(),
                "myService");
```



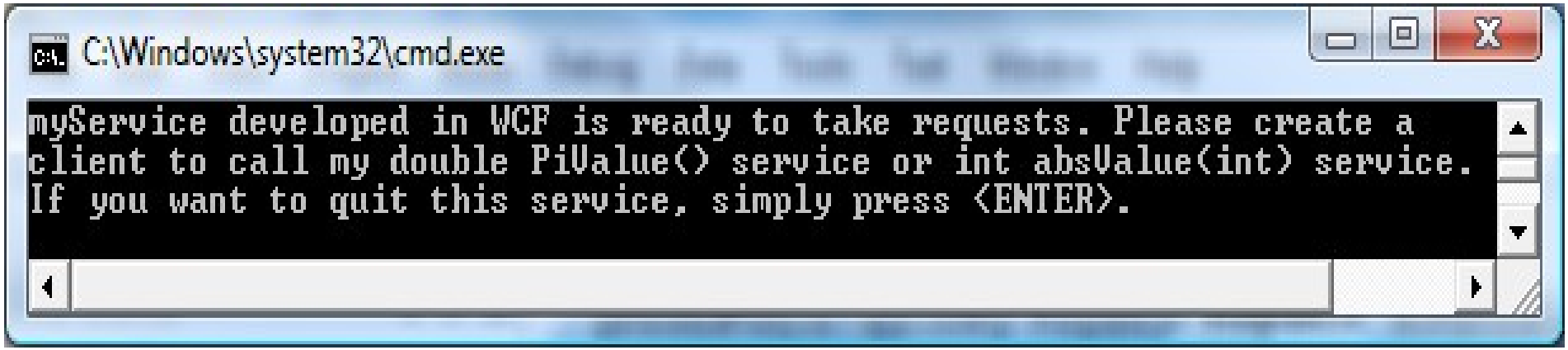
Service Code: Hosting Part

```
// (4) Add metadata for public access.
System.ServiceModel.Description.ServiceMetadataBehavior smb =
      new  System.ServiceModel.Description.ServiceMetadataBehavior( );
       smb.HttpGetEnabled = true;
       selfHost.Description.Behaviors.Add(smb);
// (5) Start the service and waiting for request.
selfHost.Open();
Console.WriteLine("myService developed in WCF is ready to take requests.
Please create a client to call my double PiValue() service or int absValue(int)
service.");
Console.WriteLine("If you want to quit this service, simply press
<ENTER>.\n");
Console.ReadLine();
// (6) Close the ServiceHostBase to shutdown the service.
selfHost.Close();
}}}}
```

Without (4), the service will be .Net - .Net remoting!

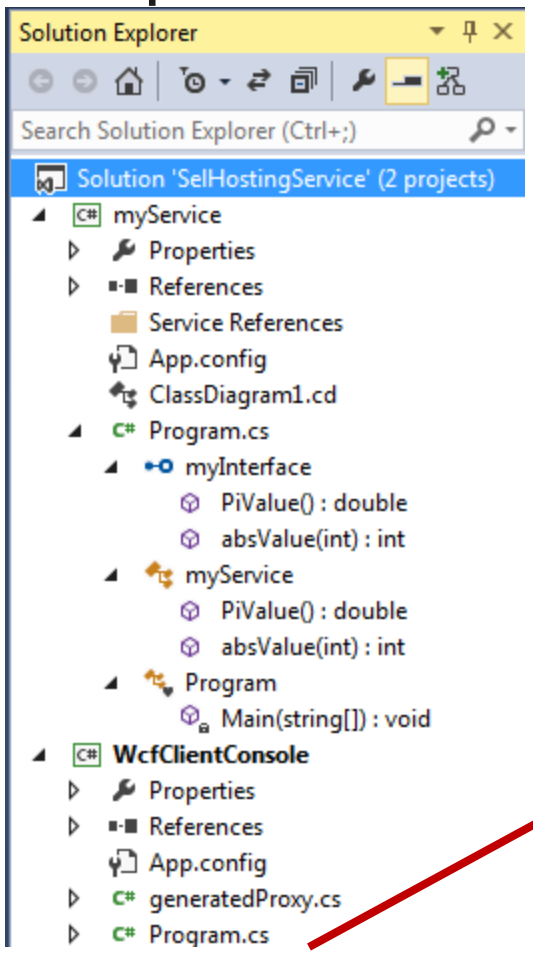
Starting the Service

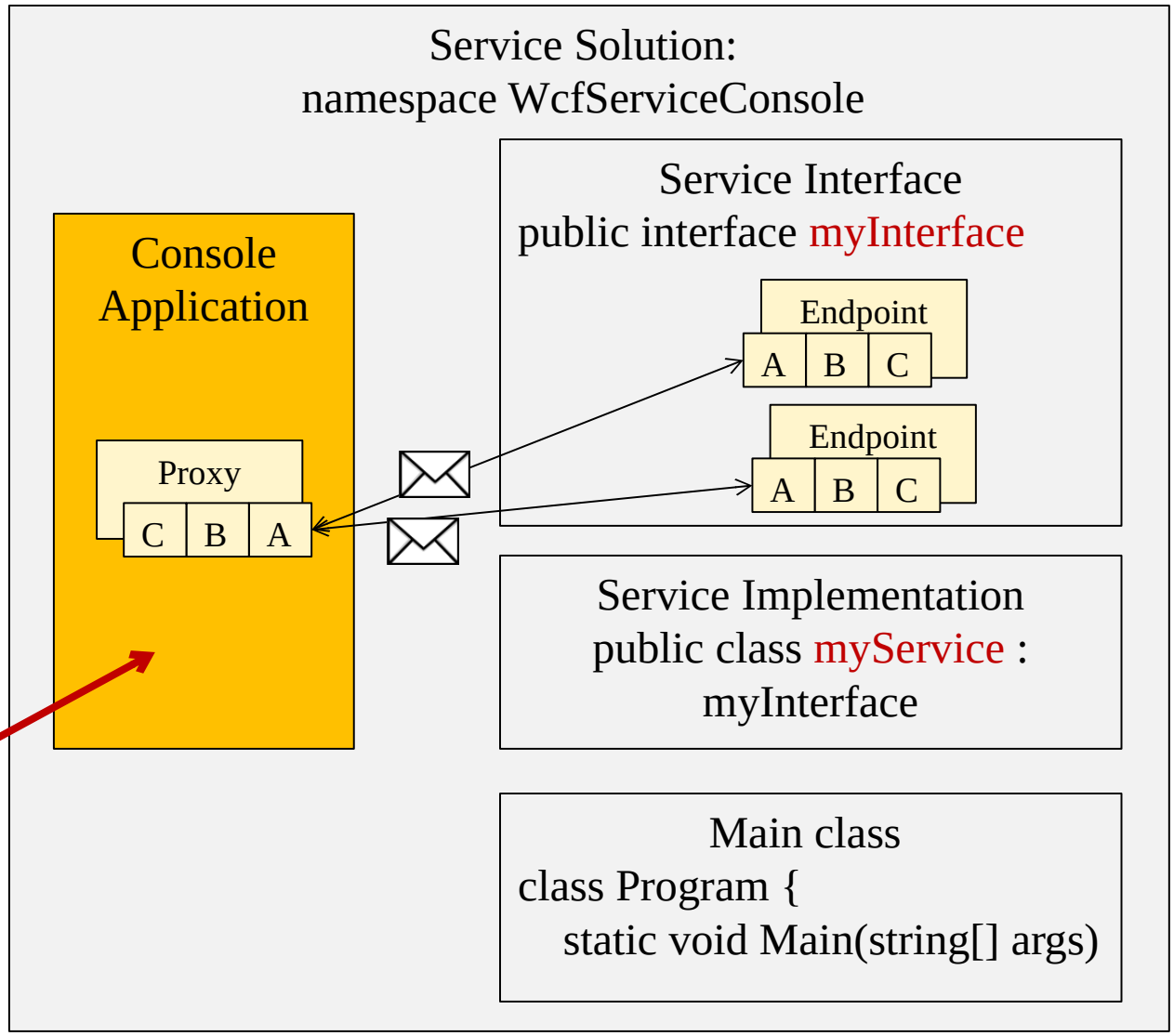
- The service is fully developed and we can start the service in Visual Studio by executing the command "Start without Debugging".



- You could also add and edit the endpoint by editing the Web.config file:
  http://msdn.microsoft.com/en-us/library/ms734786(v=vs.110).aspx

Project Overview: Service and Client



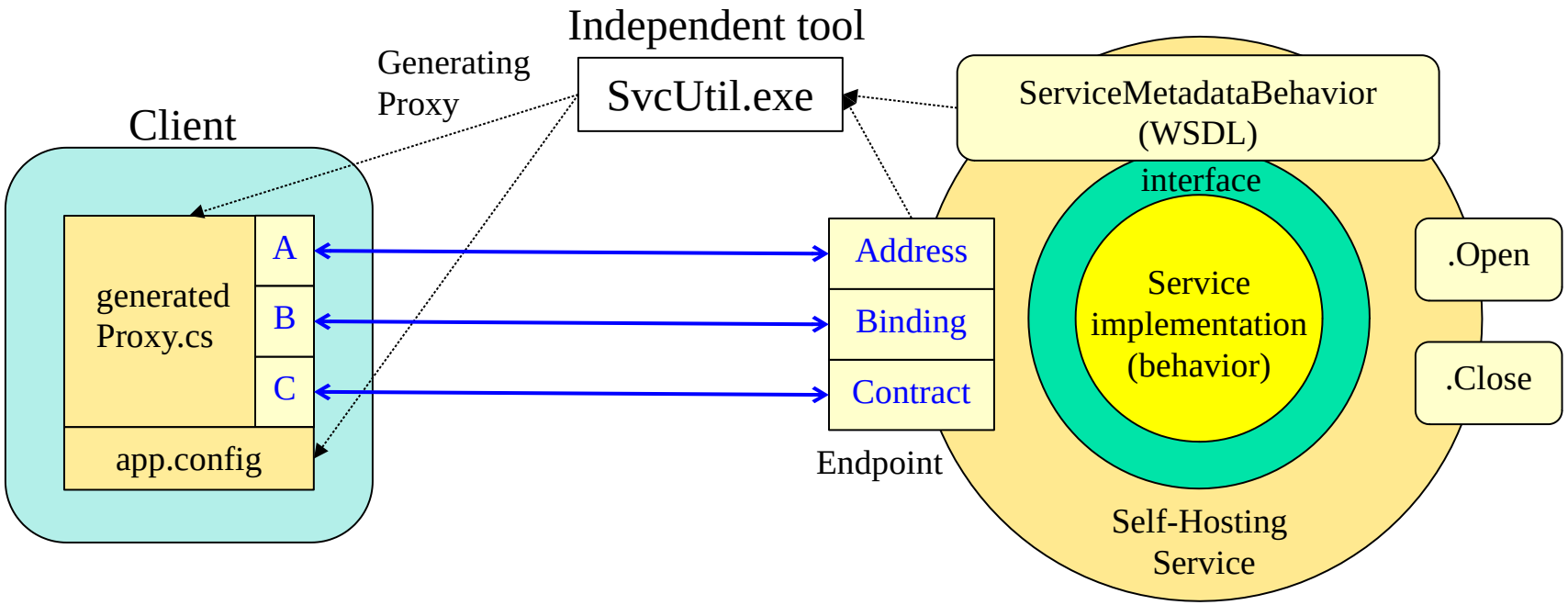


Client to Consume Self-Hosting Service in a Console Application

- Add a new Console Application in the same solution where the self-hosting project resides;
- Right-click to add Reference: System.ServiceModel under Assemblies framework (It may have been added already in the newer version of VS)
- Generate proxy to the service;
  - The service is not hosted in IIS nor in IIS Express server;
  - If the proxy **cannot** be generated by using "Add Service Reference" in your programming environment, follow the next page:

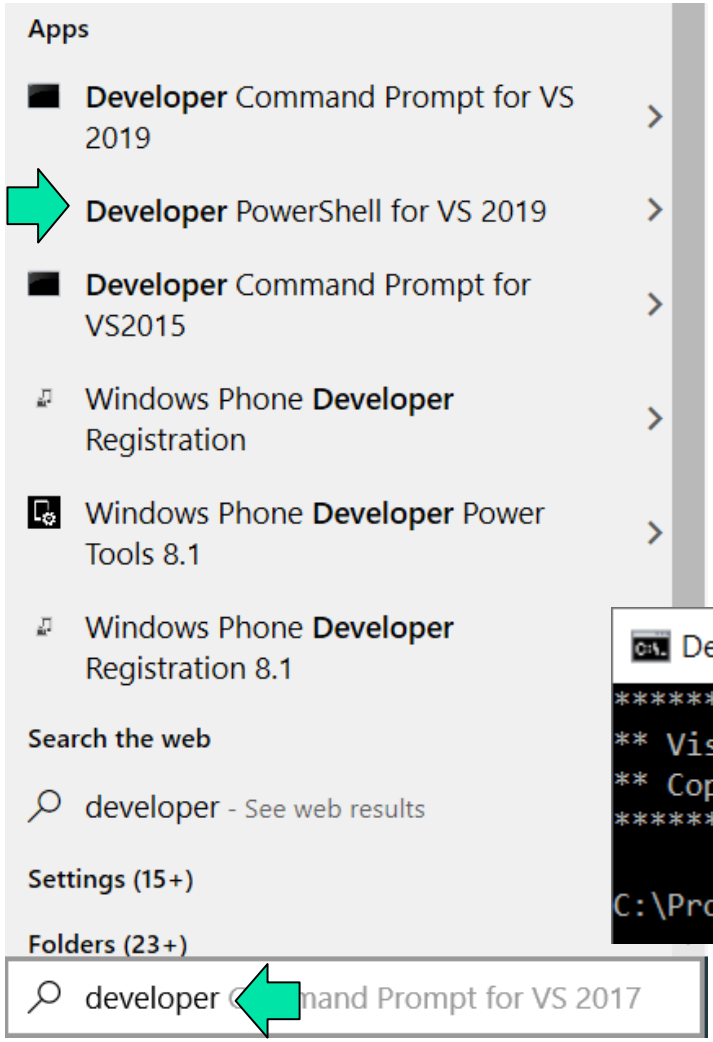
Creating the Proxy to Self-Hosting Service

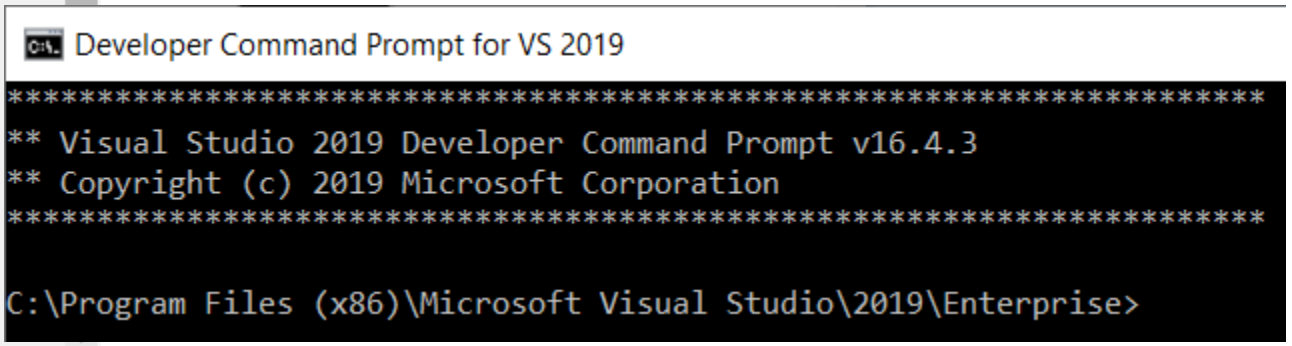
- Generate the proxy based on the metadata published by the service.
- Use the Service Model Metadata Utility Tool (SvcUtil.exe) tool to generate the proxy.



Use SvcUtil.exe Tool to Generate the Proxy

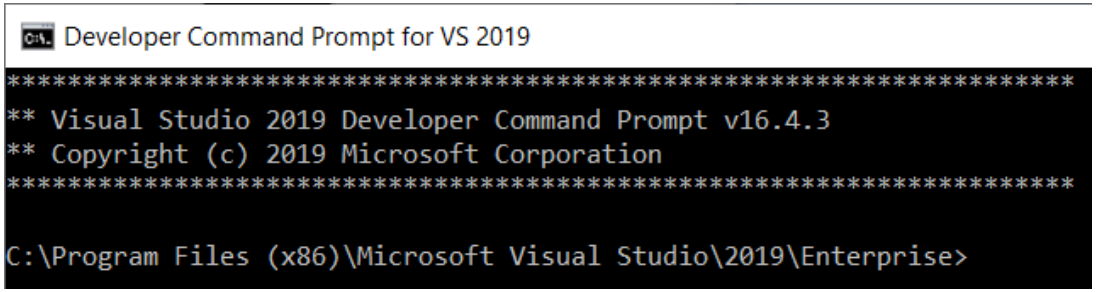
https://docs.microsoft.com/en-us/dotnet/framework/tools/developer-command-prompt-for-vs





Use SvcUtil.exe Tool to Generate the Proxy

https://docs.microsoft.com/en-us/dotnet/framework/tools/developer-command-prompt-for-vs

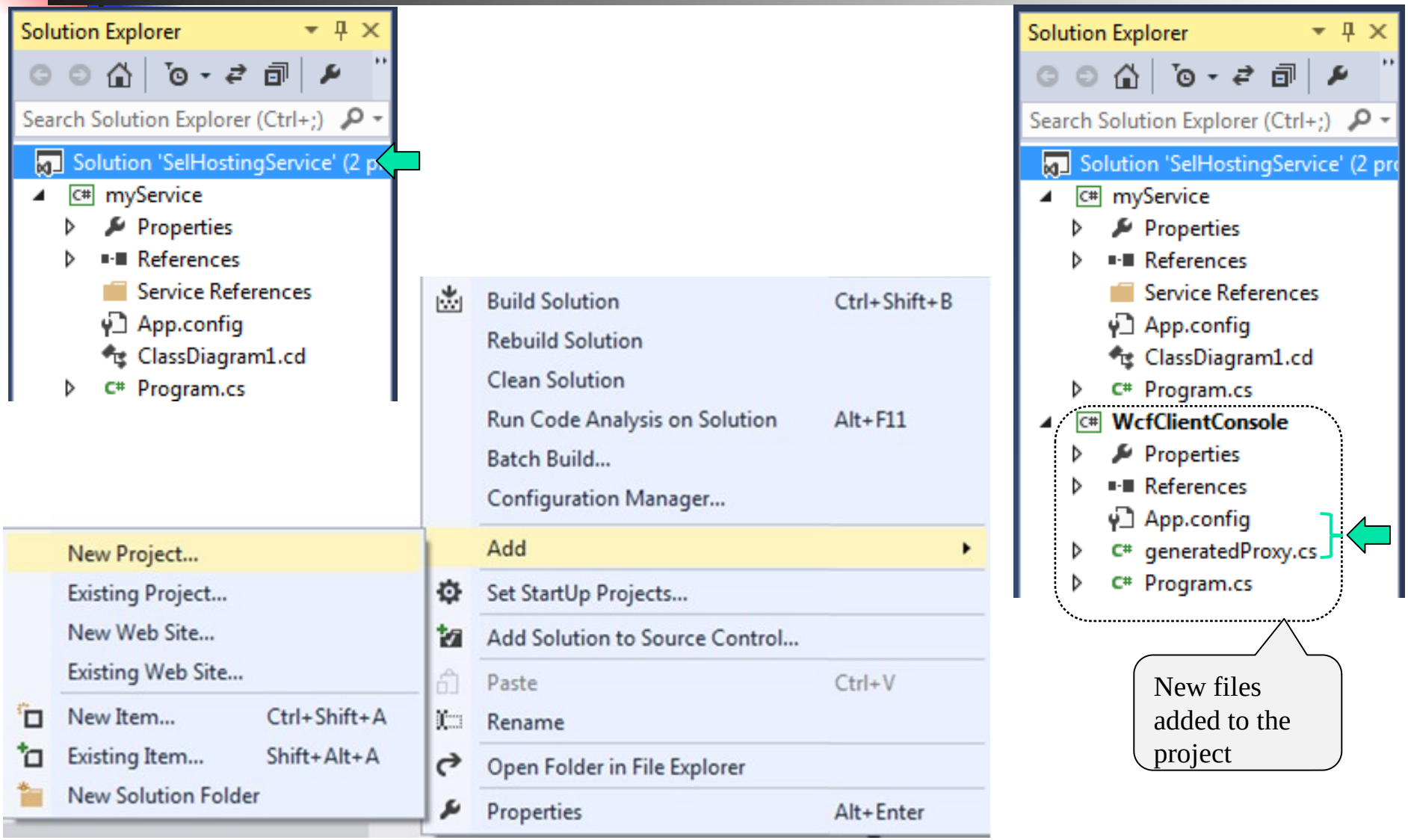


- If necessary, use DOS commands “dir”, “cd ..” and “cd folder” to navigate into the directory where you have your client project is created in the previous step.
- Keep your Service running and type the following command to execute:
- svcutil.exe /language:cs /out:generatedProxy.cs /config:app.config http://localhost:8000/Service
- This command will generate two files: app.config and generatedProxy.cs, and place them into the directory: C:\Windows\System32 or Search for them.

Use SvcUtil.exe Tool to Generate the Proxy

- Copy the two files generatedProxy.cs and app.config into you client project directory, and then add them into your client project: Right-click the client project in Solution Explorer, select Add Existing Item. Browse to the location of generatedProxy.cs to add.
- Repeat the step above to add the app.config configuration file: By default, the Add Existing Item dialog box filters out all files with a .config extension. To see these files select All Files (*.*) in the file type.
- Also read: Developer Command Prompt for Visual Studio
  https://msdn.microsoft.com/en-us/library/ms229859(v=vs.110).aspx

Add the Two Generated Files into Client



appl.config File

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
 <system.serviceModel>
  <bindings>
   <wsHttpBinding>
    <binding name="WSHttpBinding_myInterface"
             closeTimeout="00:01:00" openTimeout="00:01:00"
             receiveTimeout="00:10:00" sendTimeout="00:01:00"
             bypassProxyOnLocal="false" transactionFlow="false"
             hostNameComparisonMode="StrongWildcard"
             maxBufferPoolSize="524288"
             maxReceivedMessageSize="65536"
             messageEncoding="Text" textEncoding="utf-8"
             useDefaultWebProxy="true"
             allowCookies="false">
     <readerQuotas maxDepth="32" maxStringContentLength="8192"
             maxArrayLength="16384" maxBytesPerRead="4096"
             maxNameTableCharCount="16384" />
     <reliableSession ordered="true" inactivityTimeout="00:10:00“ enabled="false" />
```

Editing these default values

appl.config File

```
        <security mode="Message">
          <transport clientCredentialType="Windows"
                proxyCredentialType="None" realm="" />
          <message clientCredentialType="Windows"
                negotiateServiceCredential="true"
            algorithmSuite="Default" establishSecurityContext="true" />
        </security>
      </binding>
    </wsHttpBinding>
  </bindings>
  <client>
    <endpoint address=  "http://localhost:8000/Service/myService"
                binding="wsHttpBinding" bindingConfiguration="WSHttpBinding_myInterface"
                contract="myInterface" name="WSHttpBinding_myInterface">
      <identity> <userPrincipalName value="YinongBYENG\yinong" /> </identity>
    </endpoint>
  </client>
 </system.serviceModel>
</configuration>
```

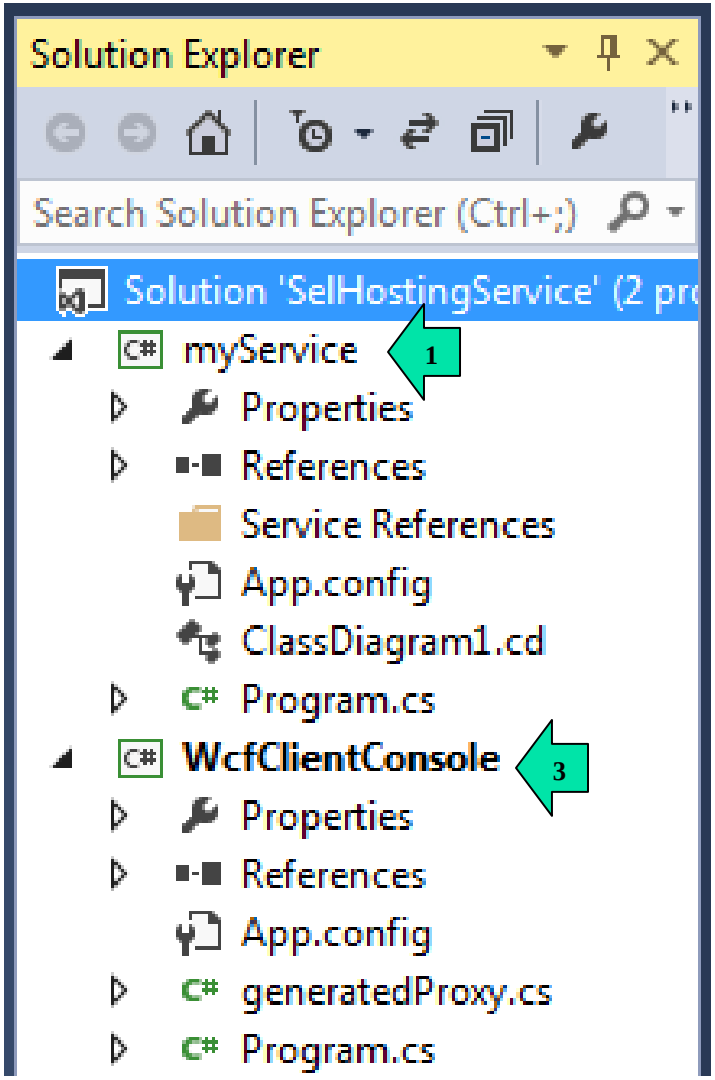
Client Program in Console Application

```
class Program {
    static void Main(string[] args) {
        //(1) Create a proxy to the WCF service.
        myInterfaceClient myProxy = new myInterfaceClient();
        // (2) Call the service operations through the proxy
        double pi = myProxy.PiValue();  // call PiValue operation
        Int32 test1 = 27;   // Create test input 1
        Int32 test2 = -132; // Create test input 2
        Int32 result1 = myProxy.absValue(test1); // call operation
        Int32 result2 = myProxy.absValue(test2); // call operation
        Console.WriteLine("PI value = {0}", pi);
        Console.WriteLine("Absolute values of {0} is {1} and of {2} is {3}",
                                                    test1, test2, result1, result2);
        myProxy.Close(); // (3) Close the proxy & channel to the service
        Console.WriteLine("/nPress <ENTER> to terminate the client./n");
    }
```

Test client and self-hosting service

1. Right click myService project and choose: Set as StartUp Project;
2. Start the project without debugging;
3. Right click the project WCFClientConsole and choose Debug Start new instance.

If the console window closes at the end, you can add Console.ReadLine at the end of the client program to prevent the console to close.

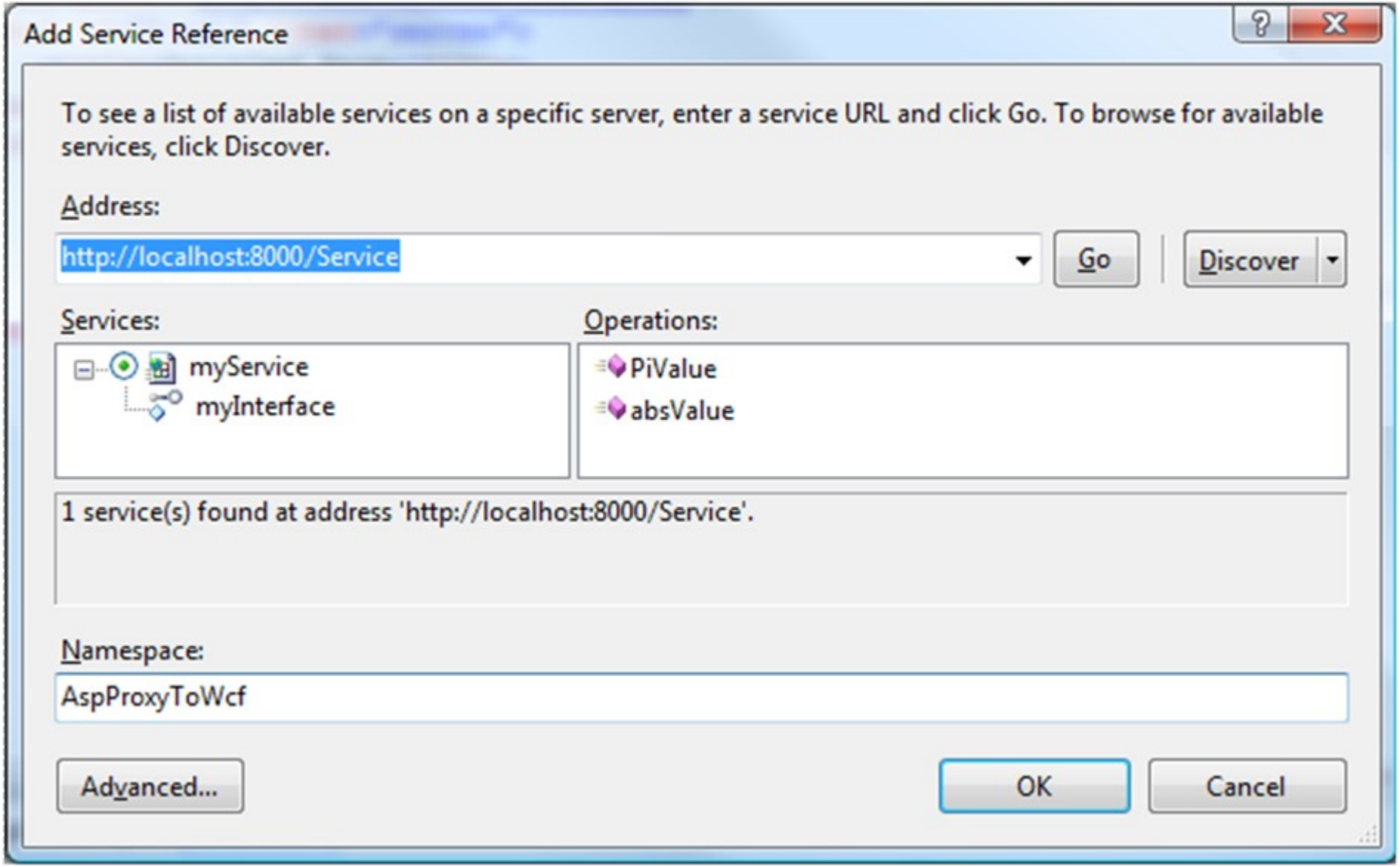


Use Localhost and ASP .Net GUI Client to Consume Self-Hosting Services

The previous example showed to create the proxy manually, so that you have full control of the proxy. You can also create the proxy automatically using Visual Studio.

- Step 1. Add an ASP .Net Web Site into the Self-Hosting Service Solution
  - Right-click the Solution WcfServiceConsole. Select Add New Web Site …
  - Select ASP .Net Web Site and the project "AspSite"
- Start the Service in localhost or in IIS
- Add Service Reference
  - http://localhost:8000/Service/myService

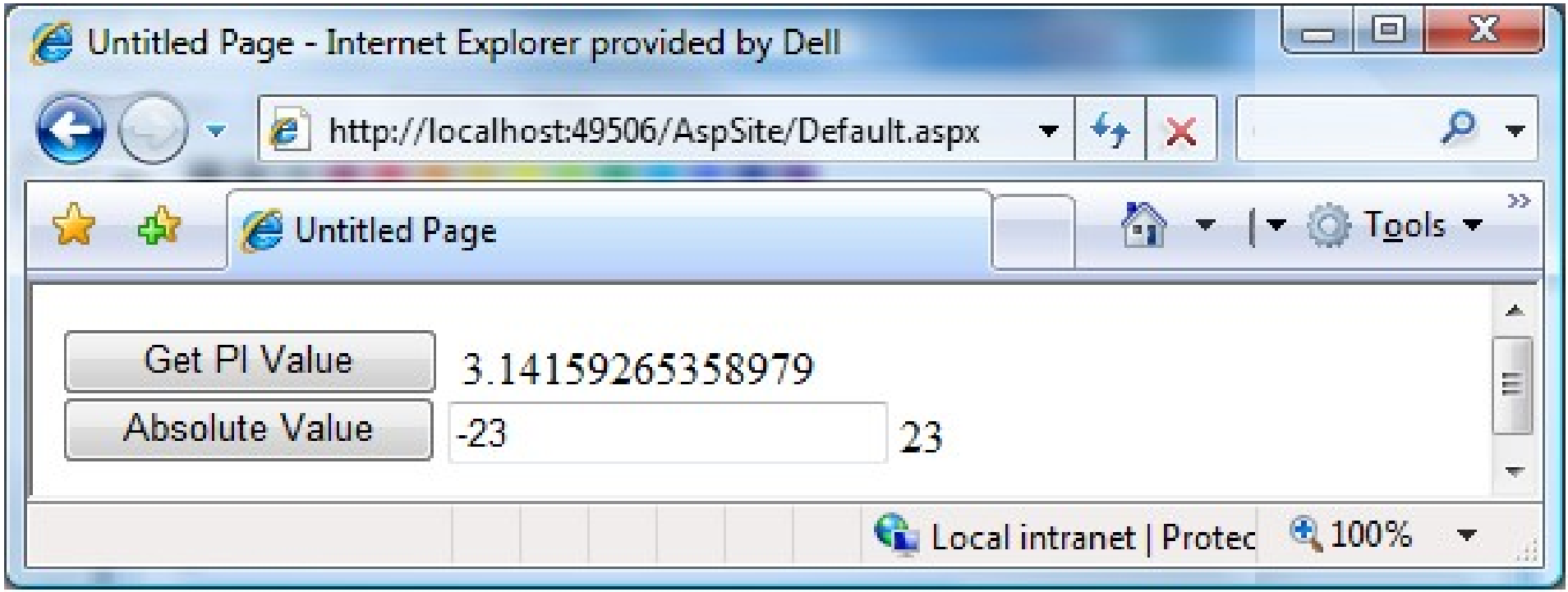
Add Service Reference



C# Code Behind the ASPX Page

```
using System;
public partial class _Default : System.Web.UI.Page {
    protected void Page_Load(object sender, EventArgs e) { }
    protected void btnPiValue_Click(object sender, EventArgs e) {
        AspProxyToWcf.myInterfaceClient refvar = new
    AspProxyToWcf.myInterfaceClient();
        double pi = refvar.PiValue();
        lblPiValue.Text = Convert.ToString(pi);
    }
    protected void Button1_Click(object sender, EventArgs e) {
        AspProxyToWcf.myInterfaceClient refvar = new
    AspProxyToWcf.myInterfaceClient();
        string abstxt = txtAbsoluteValue.Text;
        int absVal = refvar.absValue(Convert.ToInt32(abstxt));
        lblAbsoluteValue.Text = Convert.ToString(absVal);
    }
}
```

Use ASP .Net GUI Client to Consume Self-Hosting Services



Summary

- Developing Basic Service Using WCF Template and Server Hosting
- Developing Raw Service without Server Hosting: Self-Hosting
  - Writing a service without using WCF tools
  - Writing hosting service without using a server
  - Creating Proxy without using “Add Service Reference”
- Consuming WCF services
  - In Console Application
  - In ASP .Net
  - Many more: WF, BPEL, Cloud Computing, etc.