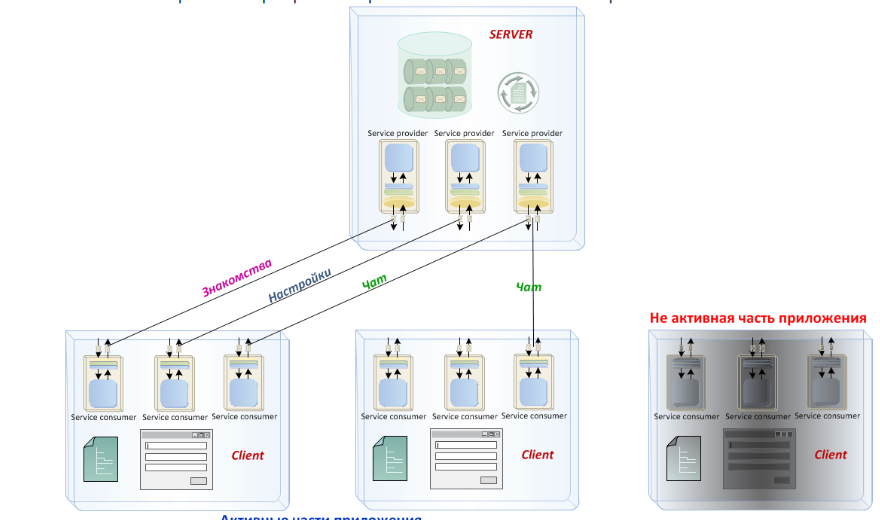
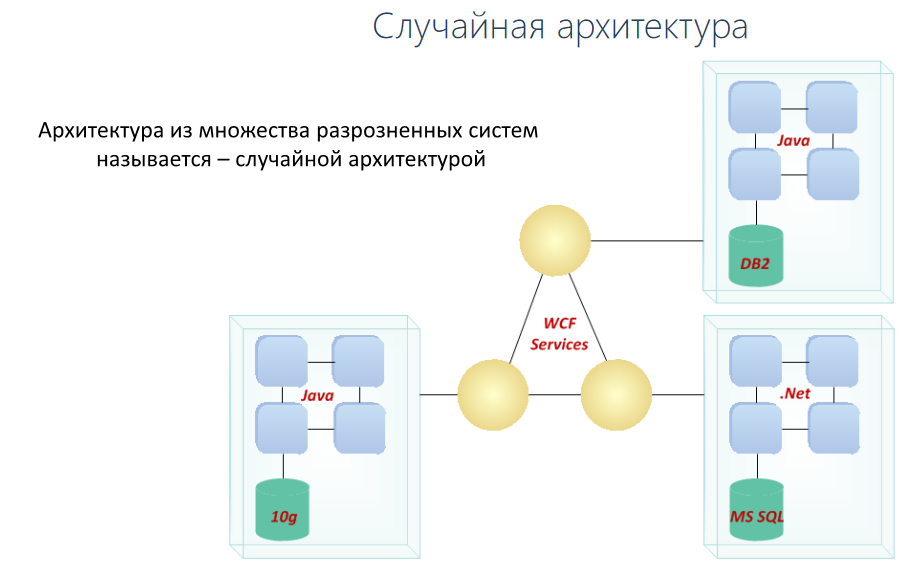
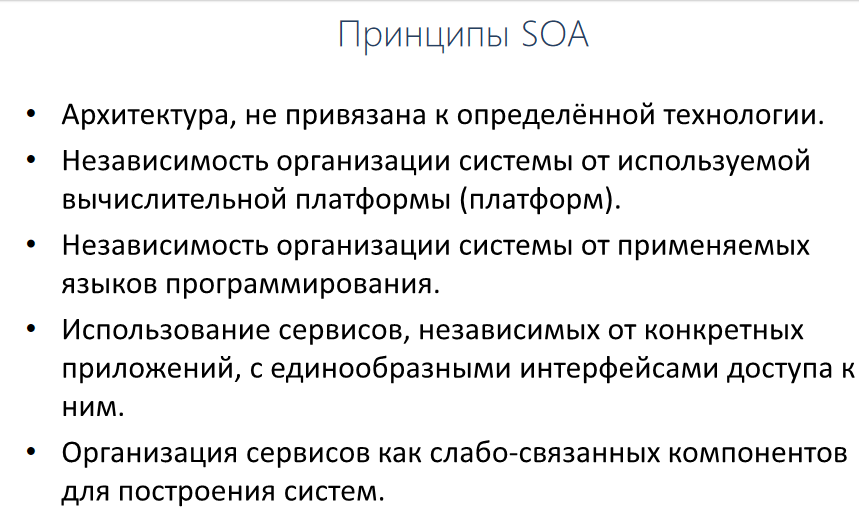
***WCF*** – это технология, предназначенная для проектирования, построения, сопровождения и модификации распределенных приложений.



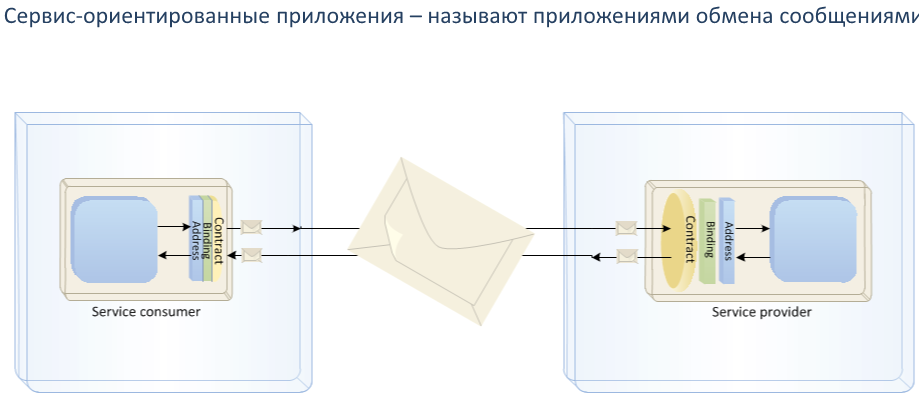


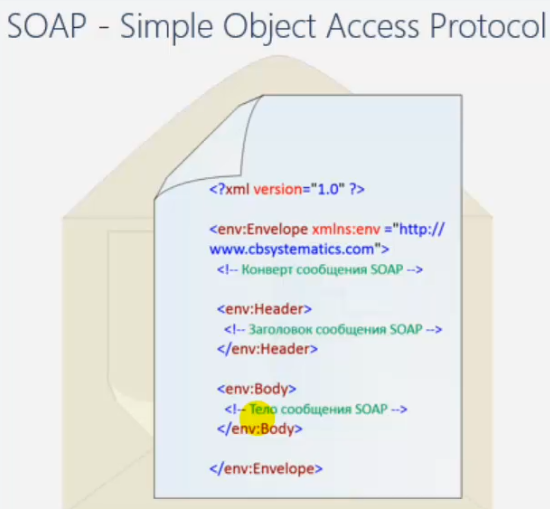




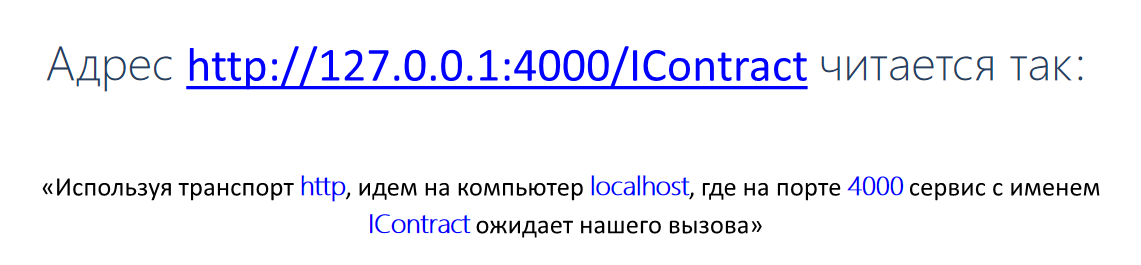


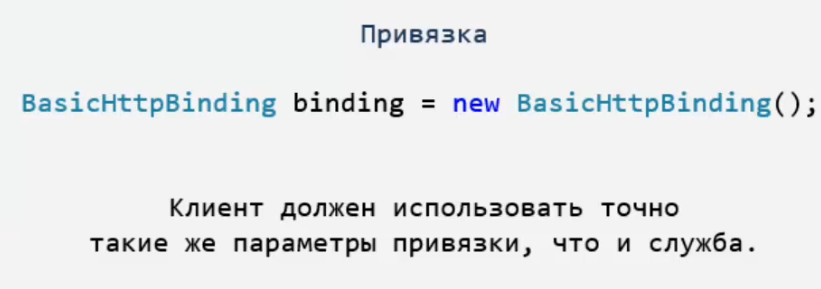




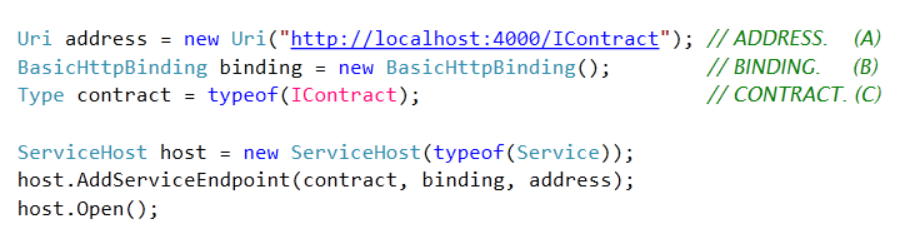


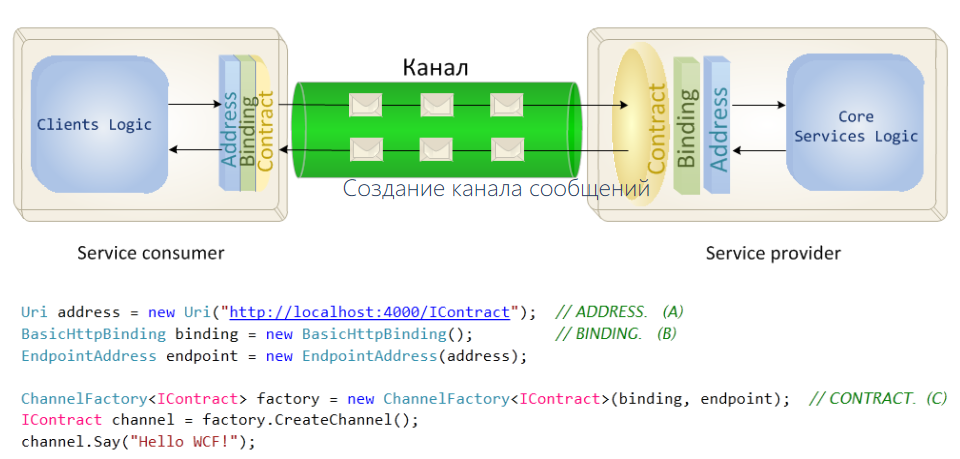


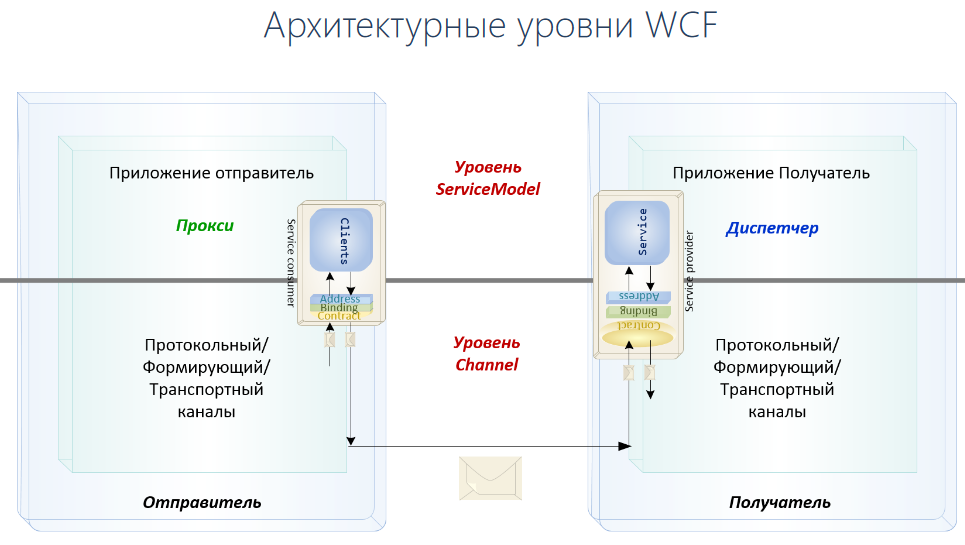












\*\*\*

Клиент

namespace Client

{

[ServiceContract]

interface IContract

{

[OperationContract]

void Say(string name, string input);

}

}

class Client

{

static void Main(string[] args)

{

Console.Title = "CLIENT";

Console.WriteLine("Введите имя пользоветеля");

string user = Console.ReadLine();

// Указание, где ожидать входящие сообщения.

Uri address = new Uri("http://localhost:4000/IContract"); // ADDRESS. (A)

// Указание, как обмениваться сообщениями.

BasicHttpBinding binding = new BasicHttpBinding(); // BINDING. (B)

// Создание Конечной Точки.

EndpointAddress endpoint = new EndpointAddress(address);

// Создание фабрики каналов.

ChannelFactory<IContract> factory = new ChannelFactory<IContract>(binding, endpoint); // CONTRACT. (C)

// Использование factory для создания канала (прокси).

IContract channel = factory.CreateChannel();

// Использование канала для отправки сообщения получателю.

while (true)

{

Console.WriteLine("Введите сообщение");

string message = Console.ReadLine();

channel.Say(user,message);

}

// Задержка.

Console.ReadKey();

}

}

Сервер

[ServiceContract]

interface IContract

{

[OperationContract]

void Say(string name, string input);

}

class Service : IContract

{

// Реализация контракта - IContract.

public void Say(string name,string input)

{

Console.WriteLine("{0}: {1}", name,input);

}

}

class Server

{

static void Main()

{

Console.Title = "SERVER";

// Указание адреса, где ожидать входящие сообщения.

Uri address = new Uri("http://localhost:4000/IContract"); // ADDRESS. (A)

// Указание привязки, как обмениваться сообщениями.

BasicHttpBinding binding = new BasicHttpBinding(); // BINDING. (B)

// Указание контракта.

Type contract = typeof(IContract); // CONTRACT. (C)

// Создание провайдера Хостинга с указанием Сервиса.

ServiceHost host = new ServiceHost(typeof(Service));

// Добавление "Конечной Точки".

host.AddServiceEndpoint(contract, binding, address);

// Начало ожидания прихода сообщений.

host.Open();

Console.WriteLine("Приложение готово к приему сообщений.");

Console.ReadKey();

// Завершение ожидания прихода сообщений.

host.Close();

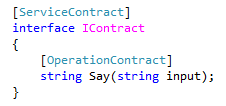
}

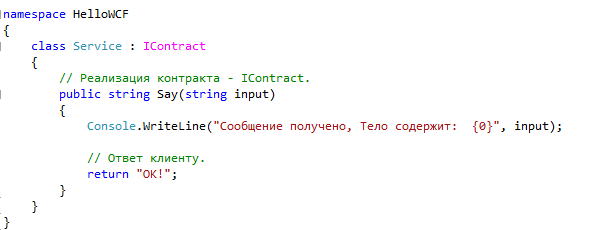
}

\*\*\*

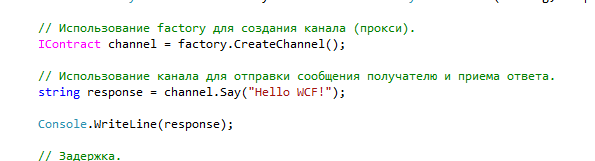
Приложение с возвращаемым значением

Сервер





Клиент



Основная функциональность WCF

разделяется на 10 категорий

1. Независимая модификация
2. Асинхронный однонаправленный обмен сообщениями

**Возможность написания приложений с применением вызовов функций типа «Запрос-Ответ».**

3. Платформенная консолидация

**Объединение возможностей многих предыдущих технологий в единую программную модель.**

4. Безопасность

* •Аутентификация
* •Авторизация
* •Конфиденциальность
* •Целостность
* •Безопасность на уровне транспорта
* •Безопасность на уровне сообщений

5. Надежность

**В распределенных вычислениях, надежная доставка сообщений называется гарантированной.**

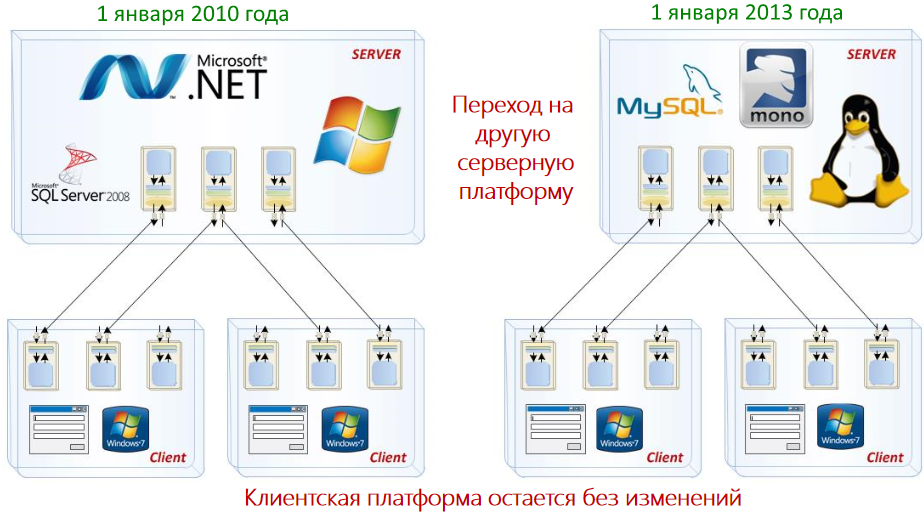
WCF содержит механизмы, обеспечивающие три вида гарантии надежной доставки сообщений:

* 1.Не более одного раза.
* 2.Хотя бы один раз.
* 3.По порядку.

6. Поддержка транзакций

7. Сетевое взаимодействие (Interoperability)

**Способность системы менять программную или аппаратную платформу, не затрагивая других участников сценария распределенных вычислений.**



8. Производительность

**Производительность при взаимодействии между двумя WCF приложениями выше,**

**чем между WCF-приложением и Web-сервисом на базе Java.)**

9. Расширяемость

**WCF - предоставляет возможности**

**для расширения своей архитектуры.**

WCF позволяет создавать свои собственные:

* •Транспорт
* •Каналы
* •Привязки
* •Кодировки
* •Архитектурные парадигмы

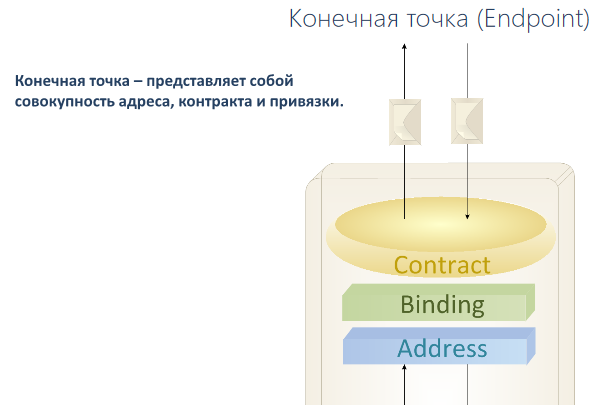
10. Возможности конфигурирования

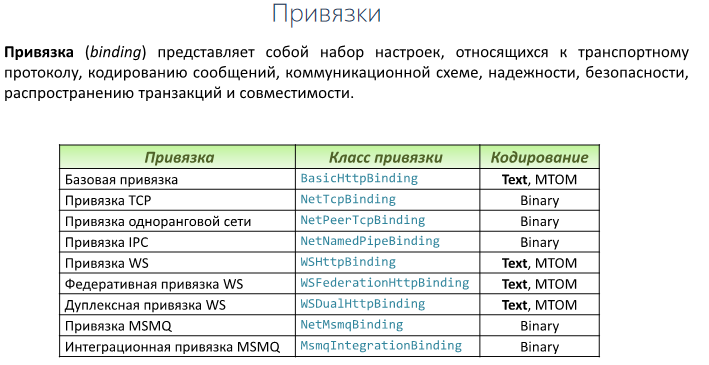
* **WCF - предоставляет возможности конфигурирования посредством**
* **конфигурационных файлов на базе XML.**
* **App.config**
* Возможность конфигурирования позволяет сотрудникам, не являющимся разработчиками, сопровождать приложение и изменять порядок его действий без привлечения команды разработчиков.

Выводы

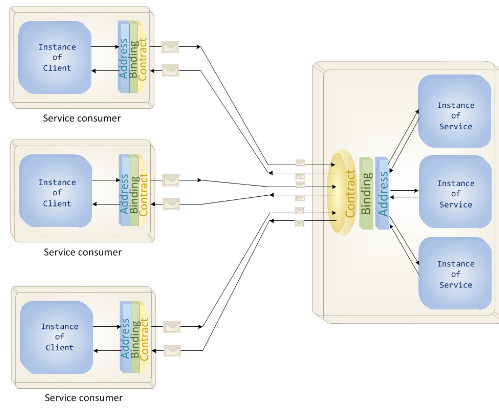
* ***WCF*** позволяет проектировать, строить, отлаживать и сопровождать распределенные системы быстрее и более функционально, чем это было раньше.
* ***WCF*** объединяет множество технологий: RPC, COM+, Remoting, ASMX, WSE и MSMQ.
* ***WCF*** расширяема.

# Конфигурирование и хостинг сервисов WCF









Варианты хостинга

* 1.Автохостинг *(Self-Hosting)*
* 2.Хостинг в службах Windows
* 3.Хостинг WAS *(Windows Activation Services)*
* 4.Хостинг IIS *(Internet Information Services)*

## Client

namespace ClientWCF

{

// КОНТРАКТ.

[ServiceContract]

interface IContractService

{

[OperationContract]

double Method(string s);

}

// КЛИЕНТ.

class Client

{

static void Main()

{

Console.Title = "CLIENT";

ChannelFactory<IContractService> channelFactory =

new ChannelFactory<IContractService>(new BasicHttpBinding(), new EndpointAddress("http://localhost:8000/ServiceWCF"));

IContractService service = channelFactory.CreateChannel();

double digit = service.Method("double");

Console.WriteLine("Double: {0}", digit);

// Задержка.

Console.WriteLine("Для завершения нажмите <Any Key>.");

Console.ReadKey();

}

}

}

## Service

namespace ServiceWCF

{

// КОНТРАКТ.

[ServiceContract]

interface IContractService

{

[OperationContract]

double Method(string s);

}

// СЕРВИС.

class MyService : IContractService

{

public double Method(string s)

{

Console.WriteLine("Обработан запрос. " + s);

if (s == "double")

return 777.77;

else

return 0;

}

}

// ХОСТ.

class Service

{

static void Main(string[] args)

{

Console.Title = "SERVER";

ServiceHost serviceHost = new ServiceHost(typeof(MyService), new Uri("http://localhost:8000/ServiceWCF"));

serviceHost.AddServiceEndpoint(typeof(IContractService), new BasicHttpBinding(), "");

serviceHost.Open();

Console.WriteLine("Для завершения нажмите <Any Key>.");

Console.ReadKey();

serviceHost.Close();

}

}

}

## Config

namespace ClientWCF

{

// КОНТРАКТ.

[ServiceContract]

interface IContractService

{

[OperationContract]

double Method(string s);

}

// КЛИЕНТ.

class Client

{

static void Main()

{

Console.Title = "CLIENT";

ChannelFactory<IContractService> channelFactory =

new ChannelFactory<IContractService>(new BasicHttpBinding(), new EndpointAddress("http://localhost:8000/ServiceWCF"));

IContractService proxy = channelFactory.CreateChannel();

double digit = proxy.Method("double");

Console.WriteLine("Double: {0}", digit);

// Задержка.

Console.WriteLine("Для завершения нажмите <Any Key>.");

Console.ReadKey();

}

}

}

Service

// Использование конфигурационного файла - App.config

namespace ServiceWCF

{

// КОНТРАКТ.

[ServiceContract]

interface IContractService

{

[OperationContract]

double Method(string s);

}

// СЕРВИС.

class MyService : IContractService

{

public double Method(string s)

{

Console.WriteLine("Обработан запрос. " + s);

if (s == "double")

return 777.77;

else

return 0;

}

}

// ХОСТ.

class Service

{

static void Main(string[] args)

{

Console.Title = "SERVER";

ServiceHost serviceHost = new ServiceHost(typeof(MyService));

serviceHost.Open();

Console.WriteLine("Для завершения нажмите <Any Key>.");

Console.ReadKey();

serviceHost.Close();

}

}

}

App config

<?xml version="1.0" encoding="utf-8" ?>

<configuration>

<system.serviceModel>

<services>

<service name ="ServiceWCF.MyService">

<host>

<baseAddresses>

<add baseAddress="http://localhost:8000/ServiceWCF"/>

</baseAddresses>

</host>

<!-- Указание Адреса, Привязки, Контракта -->

<endpoint address=""

binding="basicHttpBinding"

contract="ServiceWCF.IContractService" />

</service>

</services>

</system.serviceModel>

</configuration>

## Wpf

using System.ServiceModel;

// КОНТРАКТ.

namespace Client

{

[ServiceContract]

interface IContract

{

[OperationContract]

string Say(string input);

}

}

namespace Client

{

public partial class Window1 : Window

{

// Указание, где ожидать входящие сообщения.

Uri address = new Uri("http://localhost:4000/IContract");

// Указание, как обмениваться сообщениями.

BasicHttpBinding binding = new BasicHttpBinding();

// Ссылка на экземпляр ChannelFactory<T>, где Т - Контракт.

ChannelFactory<IContract> factory;

// Ссылка на канал (прокси).

IContract channel;

// Конструктор.

public Window1()

{

InitializeComponent();

}

private void button1\_Click(object sender, RoutedEventArgs e)

{

try

{

if (factory == null)

{

// Создание экземпляра ChannelFactory<T>, где Т - Контракт.

factory = new ChannelFactory<IContract>(binding, new EndpointAddress(address));

// Использование factory для создания канала (прокси).

channel = factory.CreateChannel();

}

if (factory != null && channel != null)

{

textBox1.Text += "From Me: " + textBox2.Text + Environment.NewLine;

// Использование channel (прокси) для отправки сообщения получателю.

textBox1.Text += channel.Say(textBox2.Text) + Environment.NewLine;

textBox2.Text = "";

}

}

catch (Exception ex)

{

MessageBox.Show(ex.Message);

}

}

// Реакция на изменение textBox1.

private void textBox1\_TextChanged(object sender, System.Windows.Controls.TextChangedEventArgs e)

{

textBox1.ScrollToEnd();

}

}

}

Server

// КОНТРАКТ.

namespace Server

{

[ServiceContract]

interface IContract

{

[OperationContract]

string Say(string input);

}

}

// СЕРВИС.

namespace Server

{

class Service : IContract

{

public string Say(string input)

{

return "From Server: Вы сказали - " + input;

}

}

}

namespace Server

{

public partial class Window1 : Window

{

// Указание, где ожидать входящие сообщения.

Uri address = new Uri("http://localhost:4000/IContract");

// Указание, как обмениваться сообщениями.

BasicHttpBinding binding = new BasicHttpBinding();

// Ссылка на экзкмпляр ServiceHost.

ServiceHost service;

// Конструктор.

public Window1()

{

InitializeComponent();

}

private void button1\_Click(object sender, RoutedEventArgs e)

{

try

{

if (service == null)

{

// Создание экзкмпляра ServiceHost.

service = new ServiceHost(typeof(Service));

// Добавление "Конечной Точки".

service.AddServiceEndpoint(typeof(IContract), binding, address);

// Начало ожидания прихода сообщений.

service.Open();

textBox1.Text += "Сервер запущен. " + DateTime.Now + Environment.NewLine;

}

}

catch (Exception ex)

{

MessageBox.Show(ex.Message);

}

}

private void button2\_Click(object sender, RoutedEventArgs e)

{

try

{

if (service != null)

{

// Завершение ожидания прихода сообщений.

service.Close();

service = null;

textBox1.Text += "Сервер остановлен. " + DateTime.Now + Environment.NewLine;

}

}

catch (Exception ex)

{

MessageBox.Show(ex.Message);

}

}

}

}

Manifest for privileges

<?xml version="1.0" encoding="utf-8"?>

<asmv1:assembly manifestVersion="1.0" xmlns="urn:schemas-microsoft-com:asm.v1" xmlns:asmv1="urn:schemas-microsoft-com:asm.v1" xmlns:asmv2="urn:schemas-microsoft-com:asm.v2" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

<assemblyIdentity version="1.0.0.0" name="MyApplication.app"/>

<trustInfo xmlns="urn:schemas-microsoft-com:asm.v2">

<security>

<requestedPrivileges xmlns="urn:schemas-microsoft-com:asm.v3">

<!--UAC Manifest Options

If you want to change the Windows User Account Control level replace the

requestedExecutionLevel node with one of the following.-->

<!--<requestedExecutionLevel level="asInvoker" uiAccess="false" />-->

<requestedExecutionLevel level="requireAdministrator" uiAccess="false" />

<!--<requestedExecutionLevel level="highestAvailable" uiAccess="false" />-->

<!--Specifying requestedExecutionLevel node will disable file and registry virtualization.

If you want to utilize File and Registry Virtualization for backward

compatibility then delete the requestedExecutionLevel node.-->

<!--<requestedExecutionLevel level="asInvoker" uiAccess="false" />-->

</requestedPrivileges>

</security>

</trustInfo>

<compatibility xmlns="urn:schemas-microsoft-com:compatibility.v1">

<application>

<!-- A list of all Windows versions that this application is designed to work with.

Windows will automatically select the most compatible environment.-->

<!-- If your application is designed to work with Windows Vista, uncomment the following supportedOS node-->

<!--<supportedOS Id="{e2011457-1546-43c5-a5fe-008deee3d3f0}"></supportedOS>-->

<!-- If your application is designed to work with Windows 7, uncomment the following supportedOS node-->

<!--<supportedOS Id="{35138b9a-5d96-4fbd-8e2d-a2440225f93a}"/>-->

<!-- If your application is designed to work with Windows 8, uncomment the following supportedOS node-->

<!--<supportedOS Id="{4a2f28e3-53b9-4441-ba9c-d69d4a4a6e38}"></supportedOS>-->

</application>

</compatibility>

<!-- Enable themes for Windows common controls and dialogs (Windows XP and later) -->

<!-- <dependency>

<dependentAssembly>

<assemblyIdentity

type="win32"

name="Microsoft.Windows.Common-Controls"

version="6.0.0.0"

processorArchitecture="\*"

publicKeyToken="6595b64144ccf1df"

language="\*"

/>

</dependentAssembly>

</dependency>-->

</asmv1:assembly>

## Wpf host form

using System.ServiceModel;

// КОНТРАКТ.

namespace Server

{

[ServiceContract]

interface IContract

{

[OperationContract]

string Say(string input);

}

}

// КЛИЕНТ.

namespace Client

{

public partial class Window1 : Window

{

// Указание, где ожидать входящие сообщения.

Uri address = new Uri("http://localhost:4000/IContract");

// Указание, как обмениваться сообщениями.

BasicHttpBinding binding = new BasicHttpBinding();

// Ссылка на экземпляр ChannelFactory<T>, где Т - Контракт.

ChannelFactory<IContract> factory;

// Ссылка на канал (прокси).

IContract channel;

// Конструктор.

public Window1()

{

InitializeComponent();

}

private void button1\_Click(object sender, RoutedEventArgs e)

{

try

{

if (factory == null)

{

// Создание экземпляра ChannelFactory<T>, где Т - Контракт.

factory = new ChannelFactory<IContract>(binding, new EndpointAddress(address));

// Использование factory для создания канала (прокси).

channel = factory.CreateChannel();

}

if (factory != null && channel != null)

{

textBox1.Text += "From Me: " + textBox2.Text + Environment.NewLine;

// Использование channel (прокси) для отправки сообщения получателю.

textBox1.Text += "From Server: " + channel.Say(textBox2.Text) + Environment.NewLine;

textBox2.Text = "";

}

}

catch (Exception ex)

{

MessageBox.Show(ex.Message);

}

}

// Реакция на изменение textBox1.

private void textBox1\_TextChanged(object sender, System.Windows.Controls.TextChangedEventArgs e)

{

textBox1.ScrollToEnd();

}

}

}

Server

// КОНТРАКТ.

namespace Server

{

[ServiceContract]

interface IContract

{

[OperationContract]

string Say(string input);

}

}

// ХОСТ.

namespace Server

{

[ServiceBehavior(InstanceContextMode = InstanceContextMode.Single)]

public partial class Window1 : Window, IContract

{

// Указание, где ожидать входящие сообщения.

Uri address = new Uri("http://localhost:4000/IContract");

// Указание, как обмениваться сообщениями.

BasicHttpBinding binding = new BasicHttpBinding();

// Ссылка на экзкмпляр ServiceHost.

ServiceHost service;

// Конструктор.

public Window1()

{

InitializeComponent();

}

private void button1\_Click(object sender, RoutedEventArgs e)

{

try

{

if (service == null)

{

// Создание экзкмпляра ServiceHost.

service = new ServiceHost(this);

// Добавление "Конечной Точки".

service.AddServiceEndpoint(typeof(IContract), binding, address);

// Начало ожидания прихода сообщений.

service.Open();

textBox1.Text += "Сервер запущен. " + DateTime.Now + Environment.NewLine;

}

}

catch (Exception ex)

{

MessageBox.Show(ex.Message);

}

}

private void button2\_Click(object sender, RoutedEventArgs e)

{

try

{

if (service != null)

{

// Завершение ожидания прихода сообщений.

service.Close();

service = null;

textBox1.Text += "Сервер остановлен. " + DateTime.Now + Environment.NewLine;

}

}

catch (Exception ex)

{

MessageBox.Show(ex.Message);

}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// СЕРВИС.

public string Say(string input)

{

textBox1.Text += "From Client: " + input + Environment.NewLine;

return "Вы сказали - " + input;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

}

}

## UIConnection

// ХОСТ.

namespace Server

{

public partial class Window1

{

// Указание, где ожидать входящие сообщения.

Uri address = new Uri("http://localhost:4000/IContract");

// Указание, как обмениваться сообщениями.

BasicHttpBinding binding = new BasicHttpBinding();

// Ссылка на экзкмпляр ServiceHost.

ServiceHost service;

// Конструктор.

public Window1()

{

InitializeComponent();

}

private void button1\_Click(object sender, RoutedEventArgs e)

{

try

{

if (service == null)

{

// Создание экзкмпляра ServiceHost.

service = new ServiceHost(typeof(Service));

// Добавление "Конечной Точки".

service.AddServiceEndpoint(typeof(IContract), binding, address);

// Начало ожидания прихода сообщений.

service.Open();

textBox1.Text += "Сервер запущен. " + DateTime.Now + Environment.NewLine;

}

}

catch (Exception ex)

{

MessageBox.Show(ex.Message);

}

}

private void button2\_Click(object sender, RoutedEventArgs e)

{

try

{

if (service != null)

{

// Завершение ожидания прихода сообщений.

service.Close();

service = null;

textBox1.Text += "Сервер остановлен. " + DateTime.Now + Environment.NewLine;

}

}

catch (Exception ex)

{

MessageBox.Show(ex.Message);

}

}

}

// Сервис

class Service : IContract

{

public string Say(string input)

{

UiConnection.UpdateText("From Client: " + input + Environment.NewLine);

return "Вы сказали - " + input;

}

}

internal static class UiConnection

{

// Получаем ссылку на текущую форму

static readonly Window1 Window1 =

Application.Current.Windows

.Cast<Window>()

.FirstOrDefault(window => window is Window1) as Window1;

// Изменяем состояние контрола

internal static void UpdateText(string @string)

{

Window1.textBox1.Text += Environment.NewLine + @string;

}

}

}

NT Services

namespace WindowsServiceNT

{

[RunInstaller(true)]

public class ProjectInstaller : Installer

{

private ServiceProcessInstaller process;

private ServiceInstaller service;

public ProjectInstaller()

{

process = new ServiceProcessInstaller();

process.Account = ServiceAccount.LocalSystem;

service = new ServiceInstaller();

service.ServiceName = "lllll ServiceNT lllll";

service.Description = "My Service !!!!!!!!!!";

service.StartType = ServiceStartMode.Automatic;

Installers.Add(process);

Installers.Add(service);

}

}

public partial class Service1 : ServiceBase

{

// Ссылка на экземпляр ServiceHost.

ServiceHost service = null;

public Service1()

{

InitializeComponent();

}

protected override void OnStart(string[] args)

{

if (service == null)

{

// Создание экземпляра ServiceHost.

service = new ServiceHost(typeof(Service));

// Начало ожидания прихода сообщений.

service.Open();

}

}

protected override void OnStop()

{

if (service != null)

{

// Завершение ожидания прихода сообщений.

service.Close();

}

}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

[ServiceContract]

interface IContract // КОНТРАКТ.

{

[OperationContract]

string Say(string input);

}

class Service : IContract // СЕРВИС.

{

public string Say(string input)

{

return "From Server: Вы сказали - " + input;

}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

}