Broadcasting Events with a Duplex WCF Service

Mike_Liu, 11 Jun 2015 CPOL Rate:

★★★★★ 4.87 (49 votes)

Write a duplex WCF service to broadcast events to all connected clients, so all connected clients can get real time notifications whenever an event occurs on any of the clients.

Download WCFBroadcastorSolution.zip - 195.5 KB

Introduction

In my previous articles, I have explained WCF, LINQ, LINQ to SQL, Entity Framework, LINQ to Entities, and how to apply LINQ to Entities to WCF services. You can find the updated articles for .NET 4.5 here:

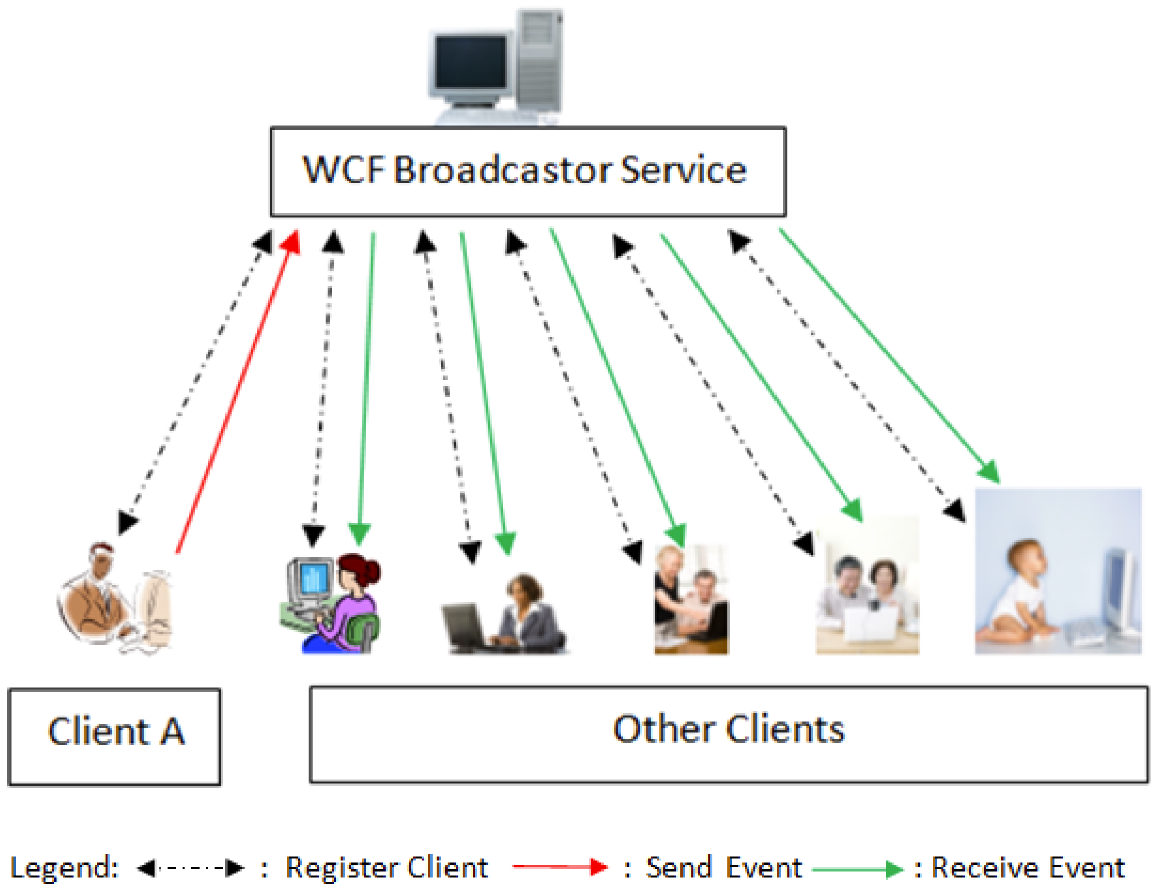
- Implementing a Basic Hello World WCF Service (v4.5)
- Implementing a WCF Service with Entity Framework (v4.5)

Now in this article, I will explain how to write a WCF service as the central point to broadcast events to all other connected clients. All other clients will get a real time notification whenever an event occurs on any of the clients.

You will need Visual Studio 2012/2013 to try the downloaded sulutions, but it will work if you have only Visual Studio 2018 or 2010; just replace .NET 4.5 with 3.5 or 4.0.

Overview

The architecture of the applications in this article is like this:


WCF Broadcastor Service
Client A
Other Clients
Legend: ◄-·-·-► : Register Client ——► : Send Event ——► : Receive Event

As you can see from this diagram, the WCF service will be running on a dedicated server. When a client starts up, it first registers itself with the service (the dotted line in the diagram). The broadcastor service will always know which client is active at any given time. The registration process also keeps a callback reference to the client, so the service can always reaches to the client if it is needed. When an event occurs on Client A, for example some data has been changed on Client A, an event notification will be sent from that client to the broadcastor service (the red line in the diagram). The service then loops through all other clients and broadcasts the event to them (the green lines in the diagram). After receiving the event notification, all other clients can process the event as they like, for example refresh the screen, or pop up a message to the end user.

These are the steps we will follow to create the WCF broadcastor service and the client application:

- Creating the WCF broadcastor service
    - Creating the WCF Service Library project
    - Defining the service interface
    - Implementing the service
    - Testing the service with WCF test client
- Hosting the duplex WCF service in IIS
    - Creating the config file
    - Adding a svc file
    - Installing IIS and ASP.NET modules
    - Enabling WCF

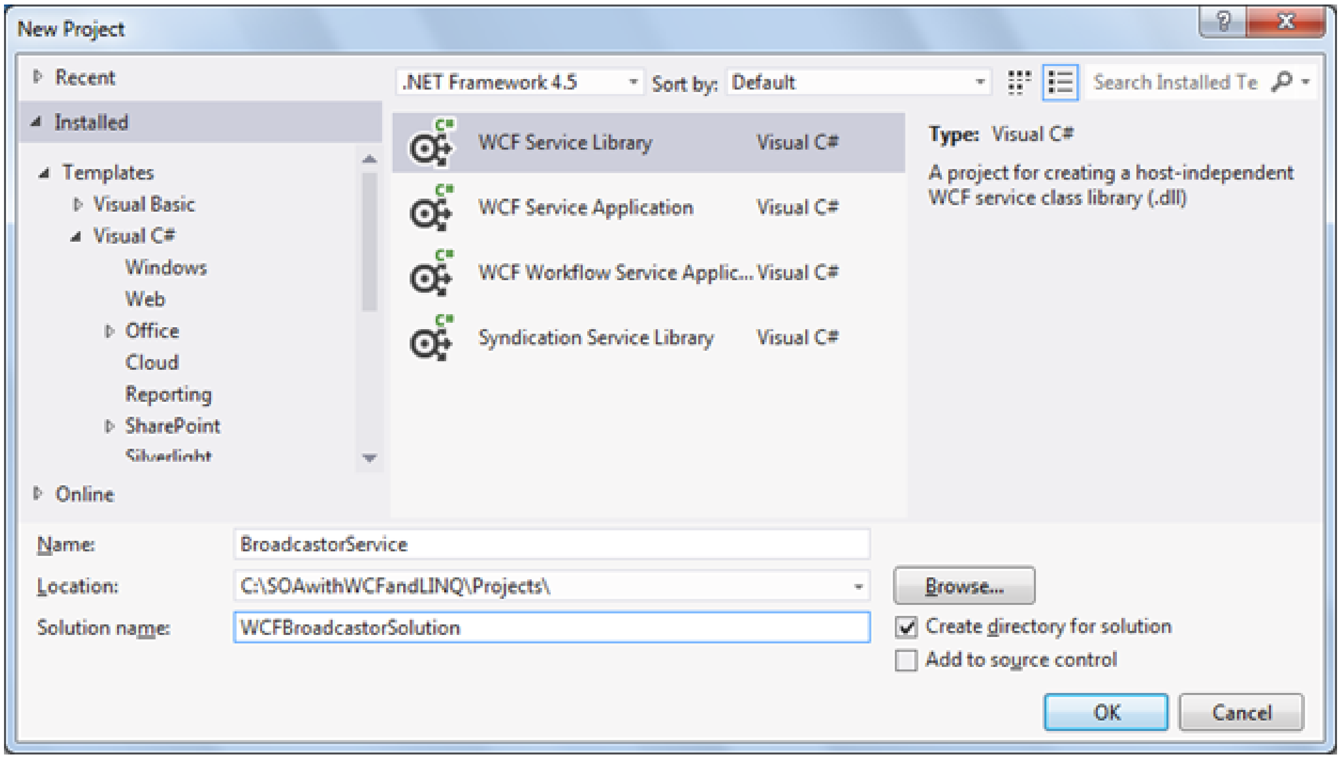
Creating the WCF broadcastor service

First of all, we will create the WCF service. We will define the service interfaces, implement the service, then modify the configuration file and define a service endpoint in the configuration file.

Creating the WCF Service Library project

To start with, we need to create a new solution with a WCF service project inside it. We will use the built-in WCF service template to create the service.

You can follow these steps to create the solution and project:



Type: Visual C#
A project for creating a host-independent WCF service class library (.dll)

Name: BroadcastorService
Location: C:\SOAwithWCFandLINQ\Projects\
Solution name: WCFBroadcastorSolution
[✓] Create directory for solution
[ ] Add to source control

1. Start Visual Studio, select the menu options File | New | Project..., and you will see the New Project dialog box.

2. In the New Project window, specify Visual C# | WCF | WCF Service Library as the project template, BroadcastorService as the (project) name, and change the solution name from the defaulted BroadcastorService to WCFBroadcastorSolution. Make sure that the checkbox Create directory for solution is selected.
3. Click on the OK button, and the solution is created with a WCF project inside it. The project already has an IService1.cs file to define a service interface and Service1.cs to implement the service. It also has an app.config file, which we will cover shortly.
4. Now we have created the WCF service project. This project is actually an application containing a WCF service, a hosting application (WcfSvcHost), and a WCF Test Client. This means that we don't need to write any other code to host it, and as soon as we have implemented a service, we can use the built-in WCF Test Client to invoke it. This makes it very convenient for WCF development.

Defining the service interface

In the previous section, we have created a WCF project by using the WCF Service Library template. In this section, we will create the service interface layer contracts.

As two sample files have already been created for us, we will try to re-use them as much as possible. Then we will start customizing these two files to create the service contracts.

Follow these steps to create the service interfaces:

1. Right-click on the file IService1.cs and rename it to IBroadcastorService.cs. Click Yes on the popup dialog window to rename all references of IService1 in the project.

2. Open file IBroadcastorService.cs

3. Delete all existing operations/data contracts within this file. Now the interface file should be like this, besides the namespace and a few using statements:

Hide Copy Code

```
[ServiceContract]
public interface IBroadcastorService
{
}
```

4. Add following data contract class to this file:

Hide Copy Code

```
[DataContract()]
public class EventDataType
{
    [DataMember]
    public string ClientName { get; set; }

    [DataMember]
    public string EventMessage { get; set; }
}
```

In next sections all clients will exchange event information using this data contract. In your real work you can add as many attributes as you like to this data contract, like event id, event type, and any other useful event data you need. You can also embed another object within this data contract, for example you may want to create a class type for the client, so the event data will contain only a client id, and from the client id you will get the client name, client type, user id, client computer name, etc.

5. Define service operations: RegisterClient, NotifyServer

Just as any WCF service, we need to define the service operations in the service interface. In this example we will define only two operations, RegisterClient and BroadcastToServer.

The first operation, RegisterClient, will be used to register a new client. When a new client application starts up, it will connect to the WCF service, and register itself with the service by invoking this operation. In this way, when an event occurs on other clients later, the service will know this client is active and waiting for an event notification, so the service will broadcast the event to this client.

The operation RegisterClient is like this:

Hide Copy Code

```
[OperationContract(IsOneWay = true)]
void RegisterClient(string clientName);
```

The operation contract attribute IsOneWay is set to true, which means this operation is going only from the client to the service. When a new client starts up, it will invoke this operation then forget about it. The client won’t wait for a response from the service for this operation.

Setting the operation contract attribute IsOneWay to true is typical for all operations in a duplex service, as there is a great chance of deadlock for a duplex service, since the service may invoke an operation on the client, while the client is invoking an operation on the service. Cares must be taken to design the operations to avoid such a deadlock and you will see more explanations later for this matter.

However for this operation it is OK to set this attribute to false if you design so. For example you may want to save the new client information into a database, create a client id, then get back that client id to the client.

The second operation, NotifyServer, will be used to notify the service that an event has occurred on a client. When an event occurs, the client will notify the service about the event by invoking this operation. the service then loops through all other clients and broadcasts the event to them.

The operation NotifyServer is like this:

Hide Copy Code

```
[OperationContract(IsOneWay = true)]
void NotifyServer(EventDataType eventData);
```

This operation has one parameter, eventData, which is of type EventDataType that we have just defined earlier. It will contain all necessary information for the service to broadcast, as you will see soon in next sections.

The service interface should be like this now (remember the data contract class EventDataType is right behind this interface within the same file):

Hide Copy Code

```
[ServiceContract]
public interface IBroadcastorService
{
[OperationContract(IsOneWay = true)]
void RegisterClient(string clientName);

[OperationContract(IsOneWay = true)]
void NotifyServer(EventDataType eventData);
}
```

6. Add a call back interface to this file:

Hide Copy Code

```
public interface IBroadcastorCallBack
{
    [OperationContract(IsOneWay = true)]
    void BroadcastToClient(EventDataType eventData);
}
```

While the service interface defines the operations that a client will invoke to notify the service about an event, the callback interface defines the operations that the service will callback the clients to broadcast the event. This callback interface is the contract between the WCF service and all clients. When the service receives an event notification, it will broadcast the event to all clients, and the operation BroadcastToClient in this interface, which will be implemented on all clients, will be the method that the service is going to invoke to broadcast the event. We will see more details when we implement this method in the client application.

The service interface should be decorated by this call back interface, in order to be a duplex service. However, once you add this call back interface to the service interface, the built-in WCF Test Client can no longer be used to test this service. So we will decorate the service interface with this call back interface after we have tested our WCF service with the built-in WCF test client.

Now we have finished defining the service interface for the broadcastor service. The full content of the file IBroadcastorService.cs should be like this now:

Hide Shrink ▲ Copy Code

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

namespace BroadcastorService
{
    [ServiceContract]
    public interface IBroadcastorService
    {
        [OperationContract(IsOneWay = true)]
        void RegisterClient(string clientName);

        [OperationContract(IsOneWay = true)]
        void NotifyServer(EventDataType eventData);
    }

    [DataContract()]
    public class EventDataType
    {
        [DataMember]
        public string ClientName { get; set; }
        [DataMember]
        public string EventMessage { get; set; }
    }

    public interface IBroadcastorCallBack
    {
        [OperationContract(IsOneWay = true)]
        void BroadcastToClient(EventDataType eventData);
    }
}
```

Implementing the service

In previous section we have defined the service interface. Next we will implement all of the operations in this interface, except the callback operation which will be implemented by the clients.

Follow these steps to implement the service interfaces:

1. Right-click on the file Service1.cs and rename it to BroadcastorService.cs. Click Yes on the popup dialog window to rename all references of Service1 in the project.

2. Open file BroadcastorService.cs

3. Delete all existing methods within this file. Now the file should be like this, besides the namespace and a few using statements:

Hide Copy Code

```
public class BroadcastorService : IBroadcastorService
{
}
```

4. Add service behavior: InstanceContextMode

First we need to decorate the service with the service behavior attribute. You can add following line of code right before the BroadcastorService class definition line:

Hide Copy Code

```
[ServiceBehavior(InstanceContextMode = InstanceContextMode.Single, ConcurrencyMode =
ConcurrencyMode.Multiple)]
```

Here we have specified the InstanceContextMode to be InstanceContextMode.Single, which means this service will be a singleton service. All clients will connect to the same instance of the service, so the client list, which will be defined soon, will be shared by all clients. This makes it possible for the service to broadcast to all clients when an event occurs.

We have also specified ConcurrencyMode to be ConcurrencyMode.Multiple, which means the service will be multi-threaded. This will increase the scalability of the service, but at same time we have to take care of the data sharing among multiple threads.

5. Add member variables: clients and locker

Now we need to add following class member variables to this class:

Hide Copy Code

```
private static Dictionary<string, IBroadcastorCallBack> clients =
new Dictionary<string, IBroadcastorCallBack>();
private static object locker = new object();
```

The first variable, clients, will be used to hold a list of all currently connected active clients. The second one, locker, will be used to synchronize the access to the variable clients among multiple threads.

6. Implement operation: RegisterClient

Now we have the variables ready, we can implement the first operation, RegisterClient. Just add following method the class:

Hide Copy Code

```
public void RegisterClient(string clientName)
{
    if(clientName != null && clientName != "")
    {
        try
        {
            IBroadcastorCallBack callback =
OperationContext.Current.GetCallbackChannel<IBroadcastorCallBack>();
            lock (locker)
            {
                //remove the old client
                if (clients.Keys.Contains(clientName))
                    clients.Remove(clientName);
                clients.Add(clientName, callback);
            }
        }
        catch (Exception ex)
        {
        }
    }
}
```

This method will be the first one that a client calls when it starts up. It will pass in its name, so the service can add it to the client list.

The callback channel we get from the current operation context is the key for the service. There will be a unique callback channel for each client, and all of the callback channels will be saved along with the corresponding client details in the client list. After a client has notified the service about an event, the service will loop through all clients, connect to each client, and broadcast the event to each client though this callback channel.

Note we need to remove the client from the list before we add a new client to the list just in case the same client re-registers itself. In your real situation, only a client name probably is not good enough to distinguish between clients. You may need to find another way on this purpose, like a GUID, or a database generated client ID each time when a client is registered.

We have also wrapped all the code that handles the client list within a lock statement, so this piece of code can be executed by multiple threads simultaneously.

7. Implement operation: NotifyServer

After a client registers itself, it needs to notify the service when an event occurs. This will be handled in the operation NotifyServer. You can add following method to the class:

Hide Shrink ▲ Copy Code

```
public void NotifyServer(EventDataType eventData)
{
    lock (locker)
    {
        var inactiveClients = new List<string>();
        foreach (var client in clients)
        {
            if (client.Key != eventData.ClientName)
            {
                try
                {
                    client.Value.BroadcastToClient(eventData);
                }
                catch (Exception ex)
                {
                    inactiveClients.Add(client.Key);
                }
            }
        }

        if (inactiveClients.Count > 0)
        {
            foreach (var client in inactiveClients)
            {
                clients.Remove(client);
            }
        }
    }
}
```

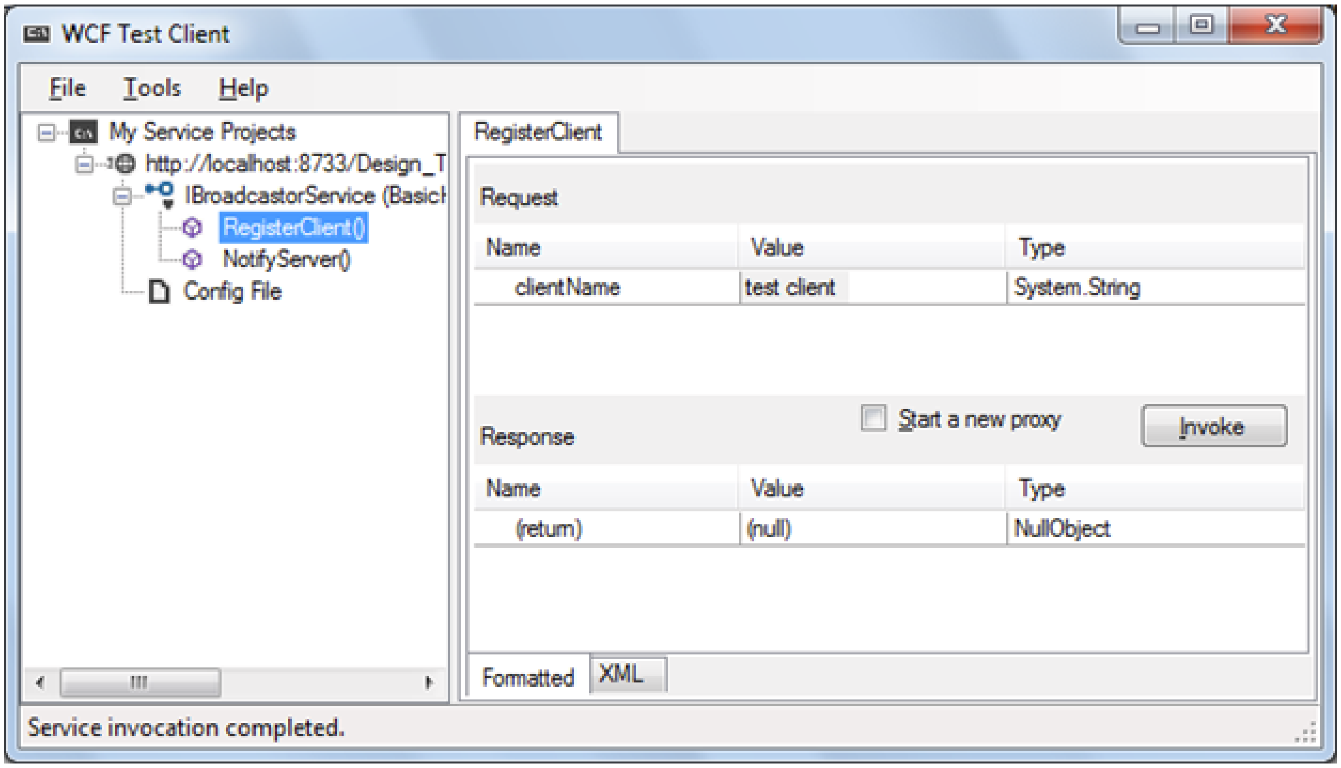
As we have said earlier, this method will handle the actual event broadcasting. When an event occurs, a client will call this method to notify the service about the event. After receiving the event notification, the service will loop though all current (presumably) active clients, connect to each client through the callback channel, which is saved earlier when the client registered itself, and broadcast the event to the client.

Remember that the method BroadcastToClient is defined in interface IBroadcastorCallBack, and will be implemented by the client.

When the service broadcasts an event to a specific client, the client may no longer be active. The client may have stopped normally, may have crashed due to a client application exception (we know this happens), or the computer may have shut down. In any of these cases, the callback channel for the client will be invalid, thus the callback will fail with an exception. To avoid future overhead with these dead clients, here we just remove them from the client list.

Testing the service with WCF test client

Now we have defined the service interface and implemented it, we can test it with the built-in WCF test client. You can build the solution, start it, and you will see the WCF test client pop up on your screen:



Request

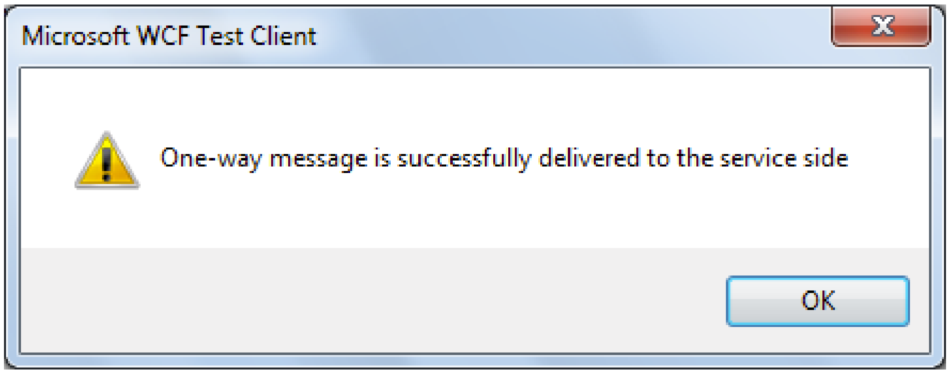
Name	Value	Type
clientName	test client	System.String

Response

Name	Value	Type
(return)	(null)	NullObject

Service invocation completed.

Open the operation RegisterClient, enter a name for the client, click on button Invoke, and you will get a message like this:



One-way message is successfully delivered to the service side

However don't be fooled by this message. It says the message is successfully delivered to the server side. But is the message processed successfully on the server side?

Let's find out following these steps:

1. Set a breakpoint on this line of code:

Hide Copy Code

```
IBroadcastorCallBack callback = OperationContext.Current.GetCallbackChannel<IBroadcastorCallBack>();
```

2. Start the service in debugging mode

3. Open the operation RegisterClient on the WCF test client

4. Enter a client name, then click on button Invoke

5. The service will stop on the breakpoint within Visual Studio.

6. Press F10.

7. An exception is thrown when the callback is casted. The error message is:
Unable to cast object of type 'System.ServiceModel.Channels.ServiceChannel' to type 'BroadcastorService.IBroadcastorCallBack

This means the type of client callback channel object is not IBroadcastorCallBack. This is expected as the WCF test client doesn't implement this interface. However because we have a try / catch block to ignore all exceptions on the service side, we won't see this exception on the test client.

It is even trickier here as this operation (RegisterClient) is a one-way operation. As a one-way operation, the client actually doesn't care or even know the result of the operation. It's a fire then forget operation. This means even if we don't have the try / catch block to ignore the exception, the client will still report "the One-way message is successfully delivered to the server side". If you read it carefully, you may have noticed the words one-way which hopefully have reminded you that the message is only delivered to the server side. You can comment out the try / catch block and re-run it to test this.

Now with the behavior of One-way operation in mind, in a real situation, you may want to change the operation RegisterClient to be two-ways by setting the operation contract attribute IsOneWay to be false. With this change, the client will know immediately if the registration is successful, and can start to post event messages to or receive event messages from other clients. Also with this change, when a client registers, you can even save the client information to a database, generate a unique id for the client, return the generated client id to the client and from there on, the client will be identified only by the client id in all communications between the client and the service. However in this article we won't go that far. We will keep it simple, and use the client name to identify all the clients.

Hosting the duplex WCF service in IIS

As you have seen we cannot test this service with the built-in WCF test client. Actually currently the service is hosted with basicHttp binding, which does not support duplex functionalities. To make it a real duplex service, we need to host it outside of Visual Studio. In this section we will host it in IIS.

Creating the config file

To host the service in IIS, we first need to create a *web.config* file. Since we have had a *App.config* file, we will create the *Web.config* file based on the *App.config* file. To do so, just copy the file *App.config* to *Web.config* file, then change it like this:

1. Remove the address attribute from the service endpoint.
2. Change the service endpoint binding from basicHttpBinding to netTcpBinding.
3. Remove the identity child element from the service endpoint.
4. Change the mex endpoint binding from mexHttpBinding to mexTcpBinding.
5. Remove the host child element from the Service element.

The *Web.config* file should be like this now:

Hide Copy Code

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

```
<configuration>
  <appSettings>
    <add key="aspnet:UseTaskFriendlySynchronizationContext" value="true" />
  </appSettings>
  <system.web>
    <compilation debug="true" />
  </system.web>
  <system.serviceModel>
    <services>
      <service name="BroadcastorService.BroadcastorService">
        <endpoint binding="netTcpBinding" contract="BroadcastorService.IBroadcastorService">
        </endpoint>
        <endpoint address="mex" binding="mexTcpBinding" contract="IMetadataExchange" />
      </service>
    </services>
    <behaviors>
      <serviceBehaviors>
        <behavior>
          <serviceMetadata httpGetEnabled="True" httpsGetEnabled="True"/>
          <serviceDebug includeExceptionDetailInFaults="False" />
        </behavior>
      </serviceBehaviors>
    </behaviors>
  </system.serviceModel>
</configuration>
```

Adding a svc file

As we have learned in previous articles, every WCF service should have a svc file to define the endpoint address. The svc file could be a physical file on the host server, or configured as file-less inside the Web.config file. In this article we will create a physical file. To see how to create a file-less svc file, you can refer to my other articles.

You can follow these steps to create the svc file:

1. Right click on the project file.
2. Add a new text file to the project with name Service.svc
3. Add following line of code to the new text file:

Hide Copy Code

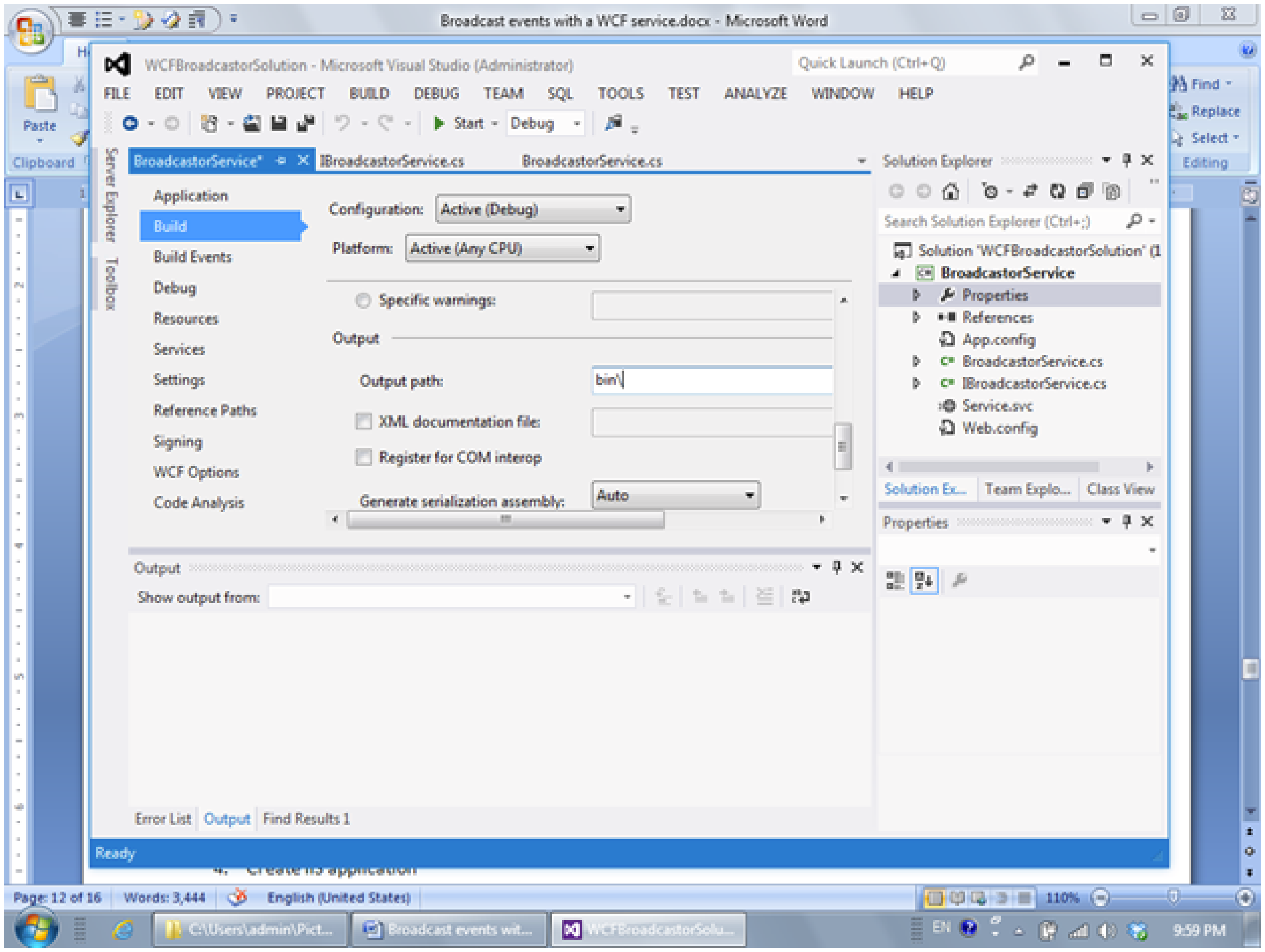
```
<%@ ServiceHost Service="BroadcastorService.BroadcastorService" %>
```

4. Save the file

5. Rebuilding the service

Now that we have finished creating the files to host the service, we need to rebuild the service. But before rebuilding the service, we need to change the output directory of the build, so IIS can find the binary files for the service.

To change the output directory, open the properties of the service project, change the build output path from bin\debug\ to bin\, as shown in following diagram:


Broadcast events with a WCF service.docx - Microsoft Word
WCFBroadcasterSolution - Microsoft Visual Studio (Administrator)
FILE EDIT VIEW PROJECT BUILD DEBUG TEAM SQL TOOLS TEST ANALYZE WINDOW HELP
BroadcastorService* IBroadcastorService.cs BroadcastorService.cs
Application
Build
Build Events
Debug
Resources
Services
Settings
Reference Paths
Signing
WCF Options
Code Analysis
Solution Explorer
Solution 'WCFBroadcasterSolution'
BroadcastorService
Properties
References
App.config
BroadcastorService.cs
IBroadcastorService.cs
Service.svc
Web.config
Output
Error List Output Find Results 1
Ready

Configuration: Active (Debug)

Platform: Active (Any CPU)

Specific warnings:

Output

Output path: bin\

XML documentation file:

Register for COM interop

Generate serialization assembly: Auto

Now rebuild the service. You can open Windows Explorer to verify that all binary files are under bin directory now.

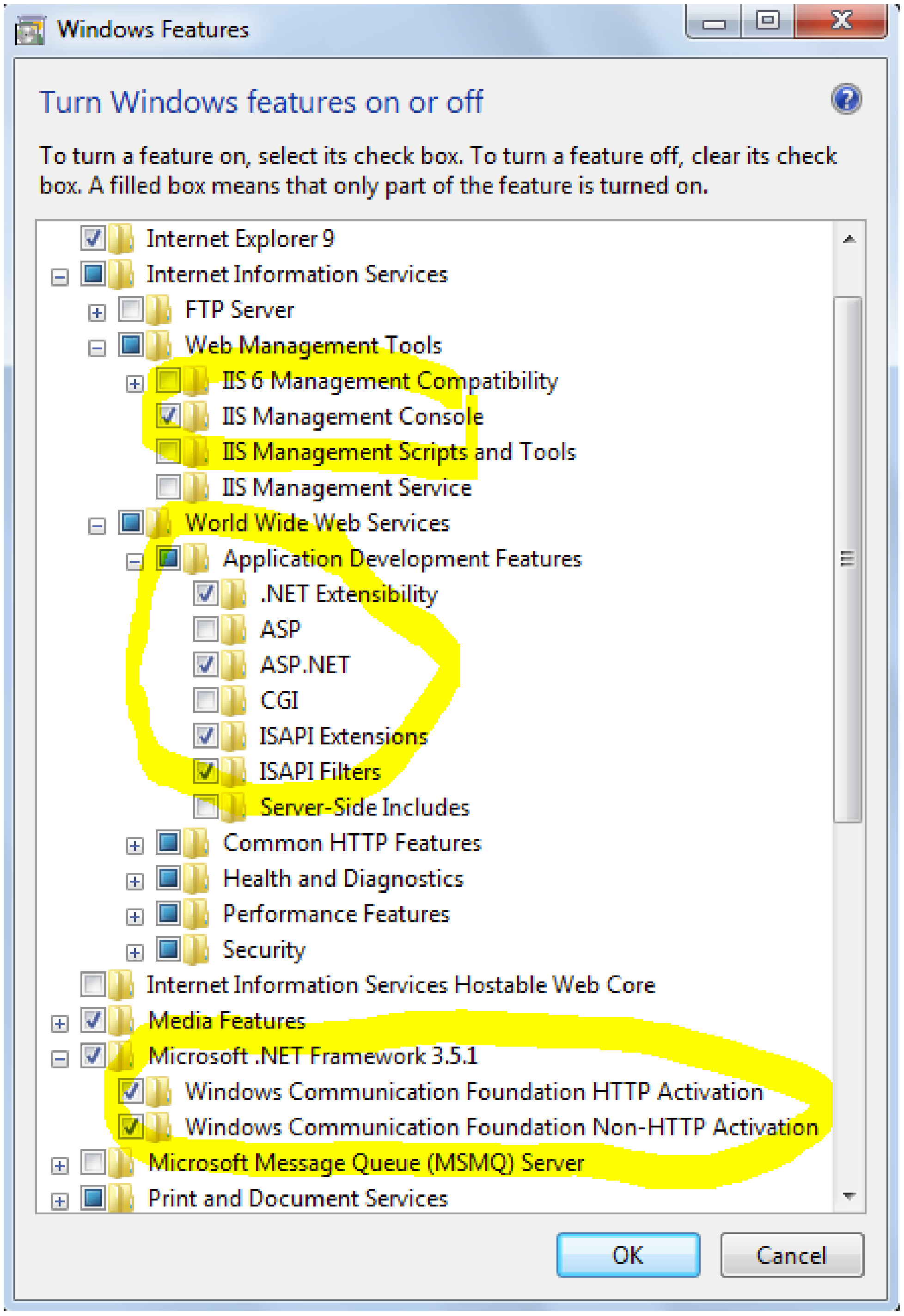
Installing IIS and ASP.NET modules

To host a WCF service in IIS, you first need to make sure Internet Information Services (IIS) is up and running. For Windows 7 and 8, IIS is not turned on by default.

To turn on IIS, go to Controls Panels\Programs\Turn Windows features on or off, check World Wide Web Services under Internet Information Services. This will install default features of IIS. Also make sure IIS Management Console is checked under Web Management Tools, which will enable you to configure IIS using IIS Manager.

As ASP.NET modules are add-ons to IIS, you also need to install them in order to host a WCF service in IIS. To enable ASP.NET modules, just expand the node Application Development Features, then check ASP.NET. This will also enable .NET Extensibility, ISAPI Extensions and ISAPI Filters, which are required for ASP.NET modules to work promptly.

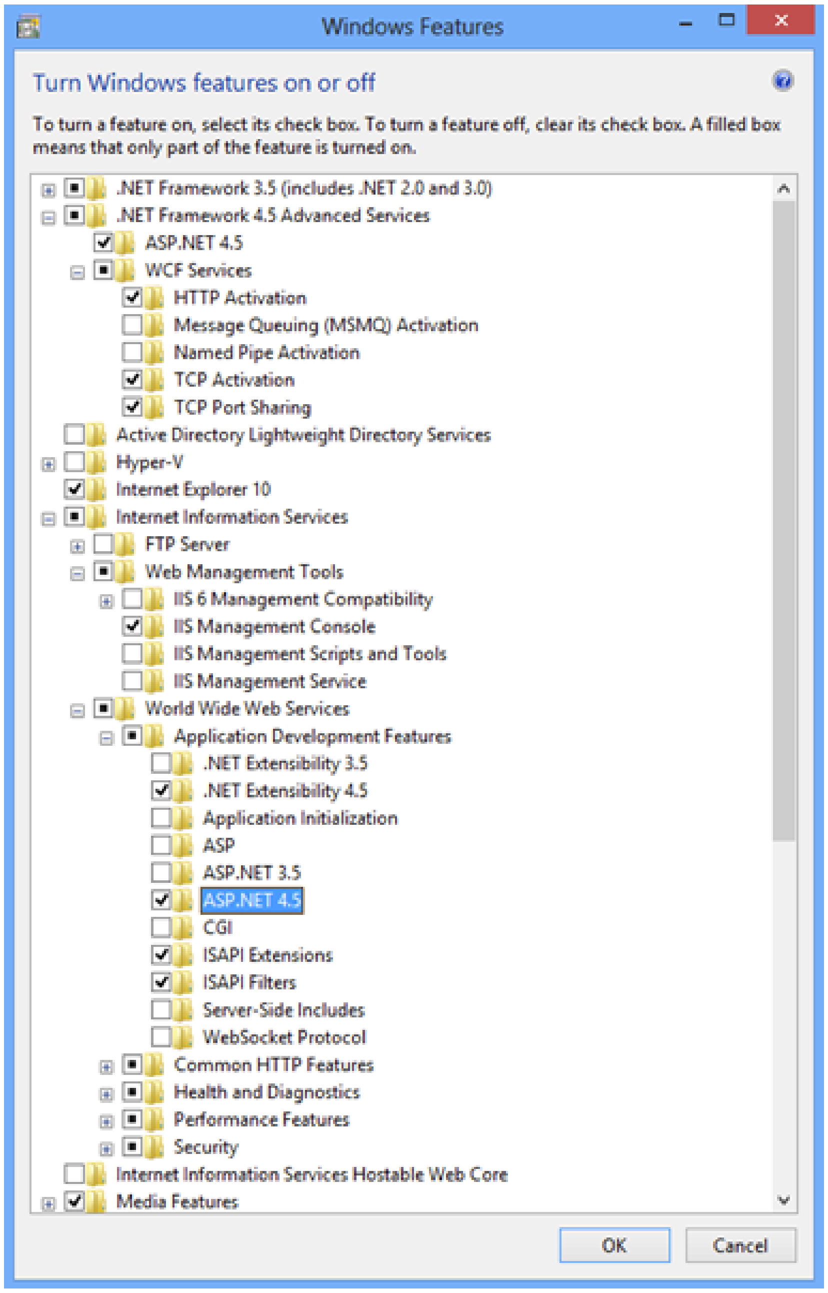
The Internet Information Services node for Windows 7 is shown in this diagram:



To turn a feature on, select its check box. To turn a feature off, clear its check box. A filled box means that only part of the feature is turned on.

- [✓] Internet Explorer 9
- [■] Internet Information Services
  - [ ] FTP Server
  - [■] Web Management Tools
    - [ ] IIS 6 Management Compatibility
    - [✓] IIS Management Console
    - [ ] IIS Management Scripts and Tools
    - [ ] IIS Management Service
  - [■] World Wide Web Services
    - [■] Application Development Features
      - [✓] .NET Extensibility
      - [ ] ASP
      - [✓] ASP.NET
      - [ ] CGI
      - [✓] ISAPI Extensions
      - [✓] ISAPI Filters
      - [ ] Server-Side Includes
    - [■] Common HTTP Features
    - [■] Health and Diagnostics
    - [■] Performance Features
    - [■] Security
- [ ] Internet Information Services Hostable Web Core
- [✓] Media Features
- [✓] Microsoft .NET Framework 3.5.1
  - [✓] Windows Communication Foundation HTTP Activation
  - [✓] Windows Communication Foundation Non-HTTP Activation
- [ ] Microsoft Message Queue (MSMQ) Server
- [■] Print and Document Services

For Windows 8, the ASP.NET Modules node is called ASP.NET 4.5, instead of just ASP.NET. Following is the corresponding diagram for Windows 8.



Turn Windows features on or off

To turn a feature on, select its check box. To turn a feature off, clear its check box. A filled box means that only part of the feature is turned on.

- ■ .NET Framework 3.5 (includes .NET 2.0 and 3.0)
- ■ .NET Framework 4.5 Advanced Services
  - ✓ ASP.NET 4.5
  - ■ WCF Services
    - ✓ HTTP Activation
    - ☐ Message Queuing (MSMQ) Activation
    - ☐ Named Pipe Activation
    - ✓ TCP Activation
    - ✓ TCP Port Sharing
- ☐ Active Directory Lightweight Directory Services
- ☐ Hyper-V
- ✓ Internet Explorer 10
- ■ Internet Information Services
  - ☐ FTP Server
  - ■ Web Management Tools
    - ☐ IIS 6 Management Compatibility
    - ✓ IIS Management Console
    - ☐ IIS Management Scripts and Tools
    - ☐ IIS Management Service
  - ■ World Wide Web Services
    - ■ Application Development Features
      - ☐ .NET Extensibility 3.5
      - ✓ .NET Extensibility 4.5
      - ☐ Application Initialization
      - ☐ ASP
      - ☐ ASP.NET 3.5
      - ✓ ASP.NET 4.5
      - ☐ CGI
      - ✓ ISAPI Extensions
      - ✓ ISAPI Filters
      - ☐ Server-Side Includes
      - ☐ WebSocket Protocol
    - ■ Common HTTP Features
    - ■ Health and Diagnostics
    - ■ Performance Features
    - ■ Security
- ☐ Internet Information Services Hostable Web Core
- ✓ Media Features

After you click on button OK, you can open a browser with this URL to test if you have IIS installed correctly:

http://localhost

With this URL you should see the default IIS Internet Information Services homepage. If you get an error, you need to fix it before going forward.

Enabling WCF

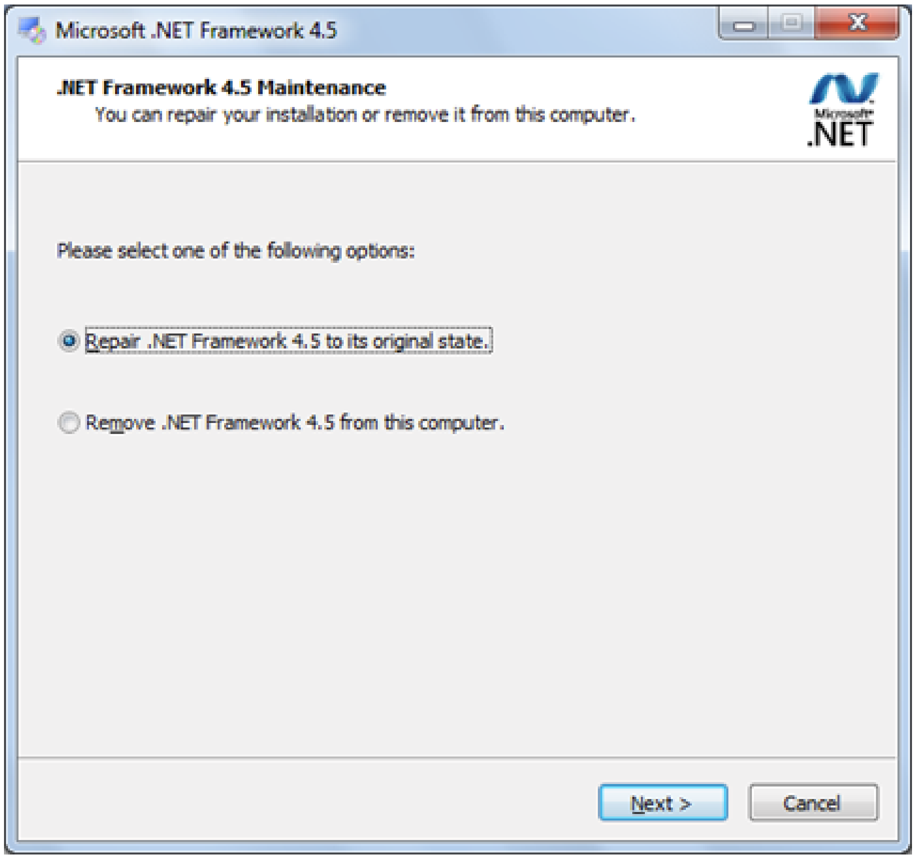
Now IIS is turned on and it can process ASP.NET requests, but it still doesn't understand WCF requests. We need to enable WCF on the machine.

To do so, you need to open Controls Panels\Programs\Turn Windows features on or off again, expand the node Microsoft .NET Framework 3.5.1, and check Windows Communication Foundation Non-HTTP Activation. Also make sure the Windows Communication Foundation HTTP Activation is checked while you are here. You can refer to previous Windows 7 Features diagram in this section to see what it looks like (and for Windows 8, you need to check TCP Activation and HTTP Activation under the node .NET Framework 4.5 Advanced Services | WCF Services, as shown in previous Windows 8 Features diagram).

Repairing .NET Framework

No matter how precisely you have followed the steps in previous sections, your IIS may still doesn't work promptly for WCF services. It may be because you have installed Visual Studio before you install IIS, you haven't registered ASP.NET correctly, you haven't configured WCF on your machine, or any other explainable / unexplainable reason. To fix all of them or just to make sure you have done everything correctly, I recommend you repairing your .NET Framework at this point. The repair won't change anything if you have done everything correctly, but if you have missed anything, it will fix it for you.

To repair .NET Framework, you can go to Control Panel\Uninstall a program, choose Microsoft .NET Framework 4.5, Click on Uninstall/Change, then Repair .NET Framework 4.5 to its original state, as shown in following diagram. You may need to restart your machine to finish this step.


Microsoft .NET Framework 4.5

.NET Framework 4.5 Maintenance
You can repair your installation or remove it from this computer.

Please select one of the following options:

[✓ Repair .NET Framework 4.5 to its original state.]

Remove .NET Framework 4.5 from this computer.

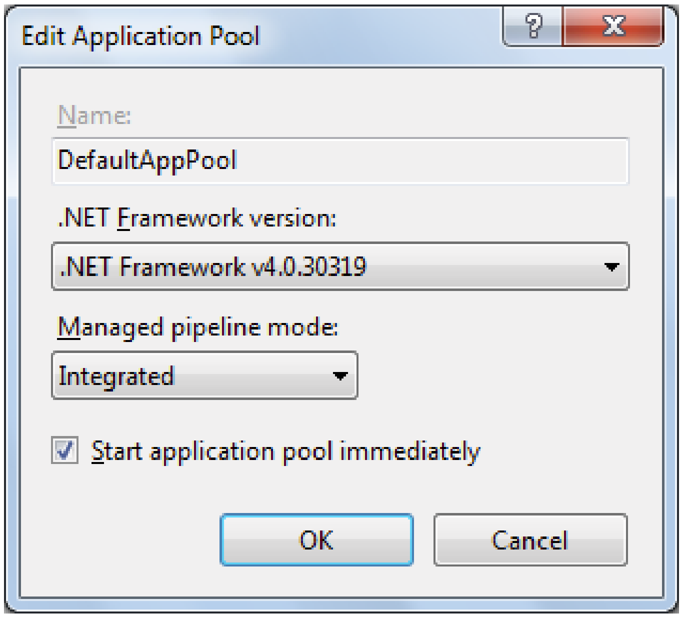
Creating a IIS application

Now IIS is installed and WCF is enabled, we can create an IIS application for the WCF service. The IIS application will host the WCF service with TCP enabled, so the client applications can connect to it, send events to it, and receive events from it.

You can follow these steps to create this application:

1. Open IIS manager

2. Select DefaultAppPool under Application Pools node
3. Open its Basic Settings to make sure its .NET Framework version is 4.0, like in this diagram:


Edit Application Pool

Name: DefaultAppPool

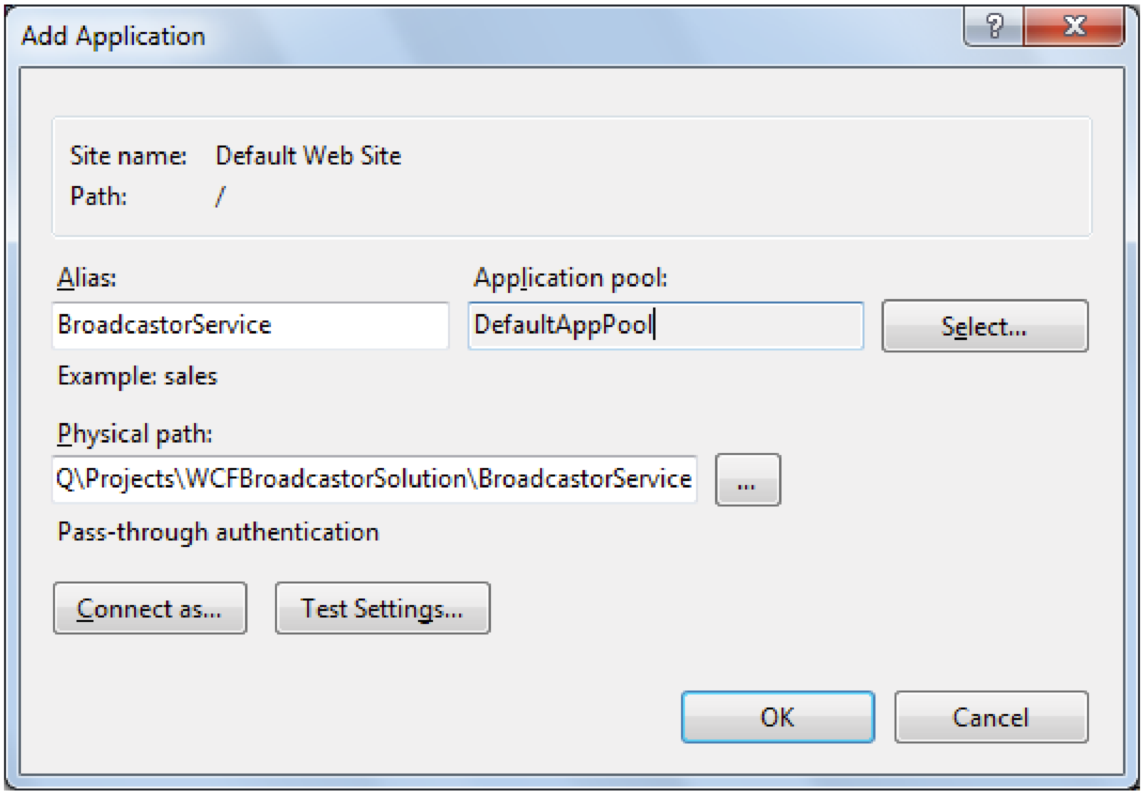
.NET Framework version: .NET Framework v4.0.30319

Managed pipeline mode: Integrated

☑ Start application pool immediately

If its version is not 4.0, it means you haven't configured your environment correctly. You need to go back to previous sections to re-configure your IIS/.NET environments.

4. Now right click on Default Web Site and Select Add Application ... from the context menu
5. Enter BroadcastorService as the Alias, leave DefaultAppPool as the Application pool, and select the service project folder C:\SOAwithWCFandLINQ\Projects\WCFBroadcastorSolution\BroadcastorService as the application's Physical path, as shown in this diagram:


Add Application

Site name: Default Web Site
Path: /

Alias: BroadcastorService

Application pool: DefaultAppPool

Example: sales

Physical path: Q\Projects\WCFBroadcastorSolution\BroadcastorService

Pass-through authentication

6. Click button OK to create the IIS application.

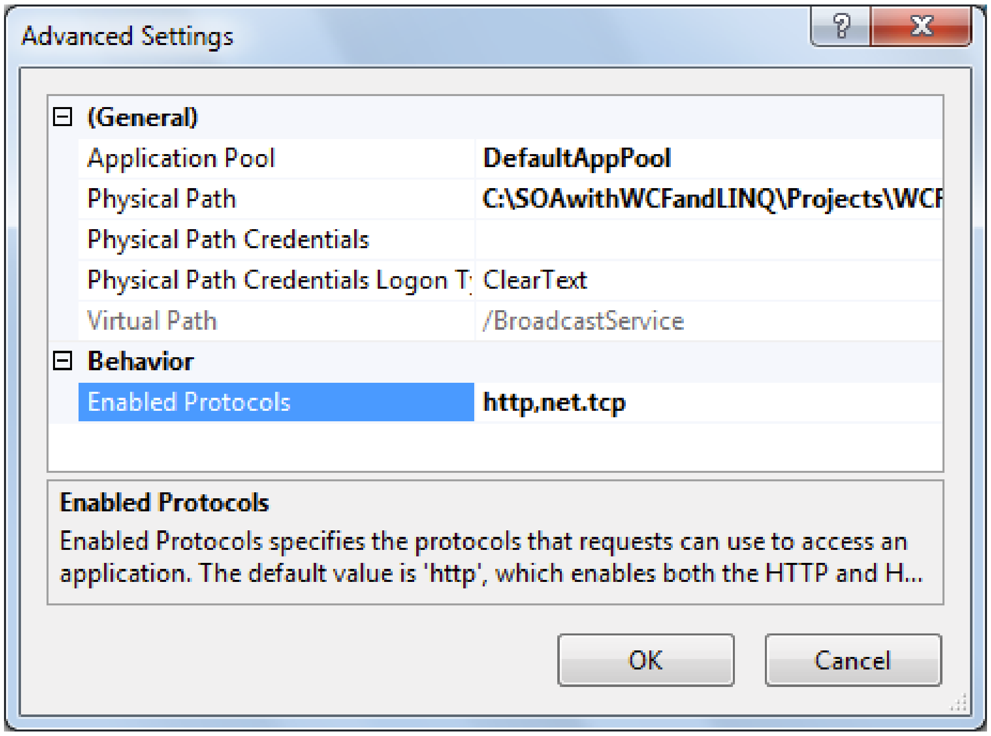
As you can see, the IIS application for this service is just a plain regular IIS application. The difference is the svc file extension which tells the IIS runtime that this is a WCF service.

Enabling protocol net.tcp

The last step to make the WCF service in this article to work is to enable TCP protocol for the service. By default only HTTP protocol is supported for an IIS application, but the WCF service we have just created needs TCP support for the duplex functionalities. So let's enable it.

You can follow these steps to enable TCP for the WCF service:

1. Open IIS manager
2. Select the service application BroadcastorService
3. Click on Advanced Settings link on the right hand panel
4. Change Enabled Protocols from http to http,net.tcp



(General)	
Application Pool	**DefaultAppPool**
Physical Path	**C:\SOAwithWCFandLINQ\Projects\WCF**
Physical Path Credentials	
Physical Path Credentials Logon T	ClearText
Virtual Path	/BroadcastService
Behavior	
Enabled Protocols	**http,net.tcp**

Enabled Protocols
Enabled Protocols specifies the protocols that requests can use to access an application. The default value is 'http', which enables both the HTTP and H...

Testing the service with a browser

Now the service is hosted properly, we can test it in a browser. Just enter following URL in your internet browser:

http://localhost/BroadcastorService/Service.svc

and you should see the service description. Click on the service link on this page should give you the service wsdl page, which proves that the service is up and running.

Adding Callback Contract Attribute

As this WCF service will be a duplex service, we need to specify the callback contract of the service, i.e. we need to specify the CallbackContract attribute in the service interface decoration. You may recall in a previous section we have defined the callback interface, but didn't decorate the service. Now we have hosted the service in IIS, we can add this attribute, so in next section, we can implement this callback interface in the client program.

To do so, just open the service interface file IBroadcastorService.cs, change the empty service interface decoration [ServiceContract] to be like this:

Hide Copy Code

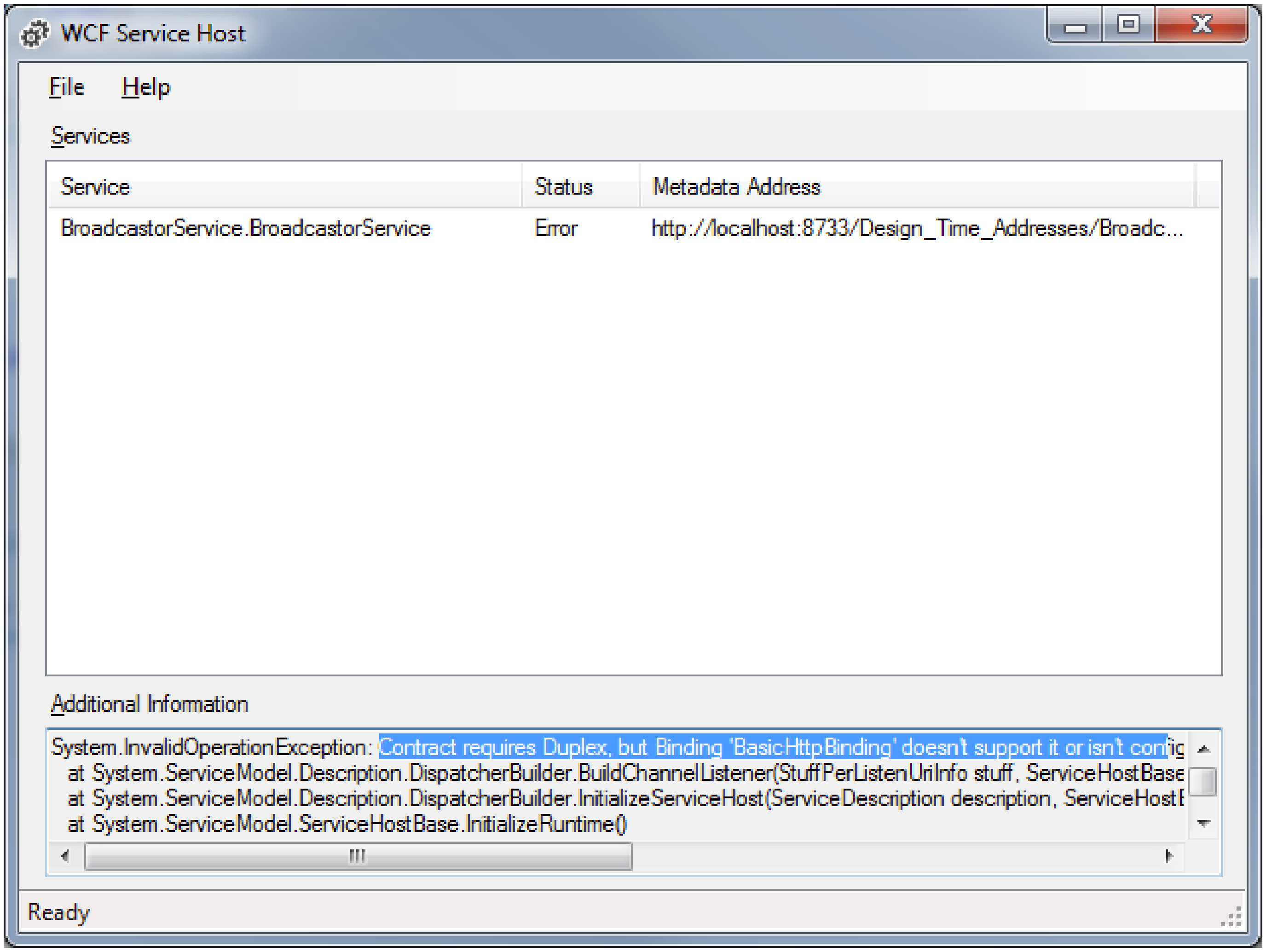
```
[ServiceContract(CallbackContract = typeof(IBroadcastorCallBack))]
```

As you can see, the attribute CallbackContract is added to the service interface decoration. With this attribute, the service will be able to invoke the operations that are specified in the callback contract. In our example there is only one operation IBroadcastorCallBack for the service to call back, which will be implemented by the clients.

And you need to rebuild the service to make this change take effect.

Disabling WCF Service Host

As soon as the callback contract attribute is added to the service, the built-in WCF Service Host will not work with the service. To try it, you can press Ctrl+F5 to start the service/WCF Test client and you will get the error "Contract requires Duplex, but Binding 'BasicHttpBinding' doesn't support it or isn't configured properly to support it":



```
BroadcastorService.BroadcastorService    Error    http://localhost:8733/Design_Time_Addresses/Broadc...
```

```
System.InvalidOperationException: Contract requires Duplex, but Binding 'BasicHttpBinding' doesn't support it or isn't config
   at System.ServiceModel.Description.DispatcherBuilder.BuildChannelListener(StuffPerListenUriInfo stuff, ServiceHostBase
   at System.ServiceModel.Description.DispatcherBuilder.InitializeServiceHost(ServiceDescription description, ServiceHostB
   at System.ServiceModel.ServiceHostBase.InitializeRuntime()
```

This will be annoying later if you run the clients in debug mode, as the built-in WCF Service Host will try to start every time when you are debugging another project in the same solution. Since from this point on, we only need to use IIS to host the WCF service, we can disable the built-in WCF Service Host.

To do so, you can open property of the service project, go to WCF Options tab, then uncheck "Start WCF Service Host when debugging another project in the same solution".

Creating the client application

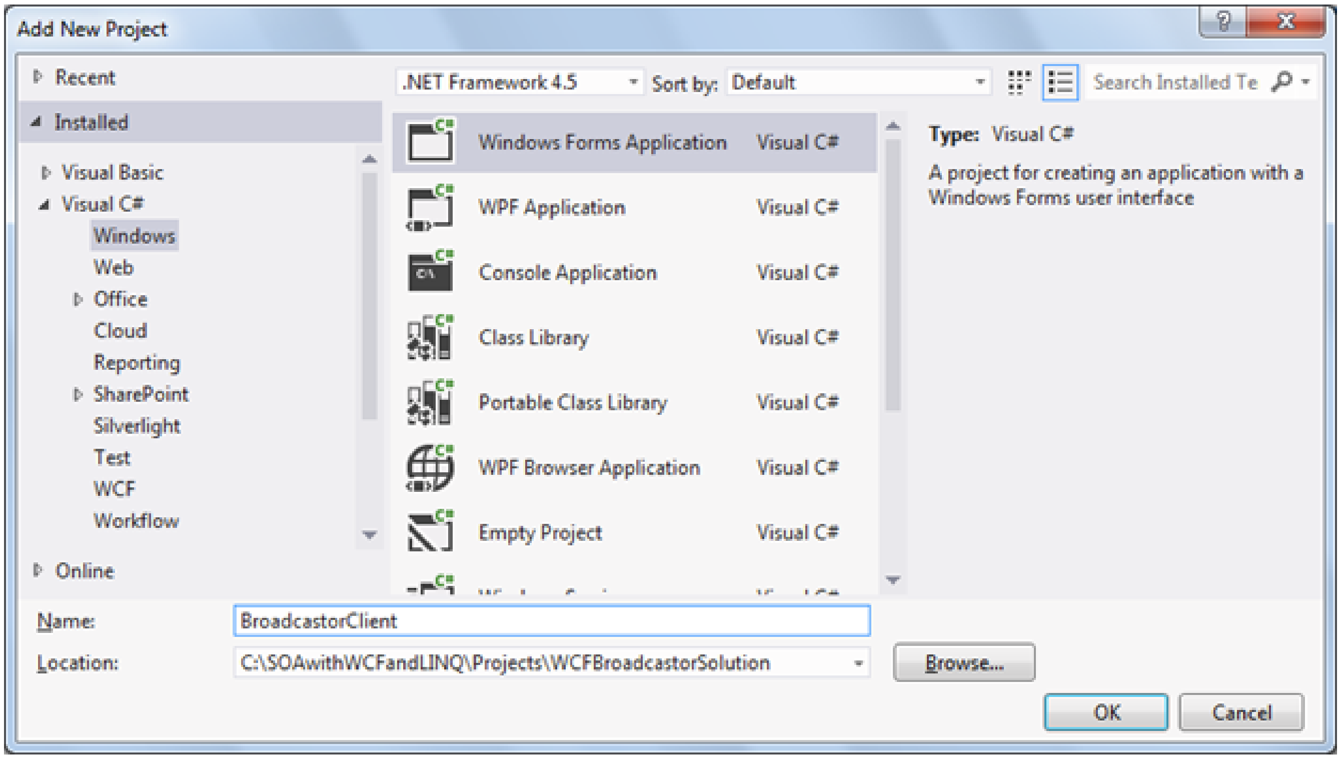
In previous sections we have created the duplex WCF service and hosted it in IIS. Now in this section we will create a client application to connect to it. As we have said earlier, the client will register itself, then send an event notification to the service when an event occurs. The service then broadcasts the event to all other clients and all other clients will process the event accordingly.

Creating the client application projects

First of all we need to add a client application to the solution. In this article we will create a Winforms application and a WPF application for demo purpose, but in reality, you can create any kind of application, as long as it can consume a WCF web service that is hosted with TCP protocol (thanks to reader sweet pain, I now have added a WPF client).

You can follow these steps to create these two client applications:

1. Right click on the solution item in Solution Explorer
2. Select Add | New Project ... from the context menu
3. Choose Windows Forms Application/WPF application as the type, and enter BroadcastorClient/BroadcastorWPFClient as the name (we will only show the screenshot for the Winforms client, but the WPF one is very similar to it):

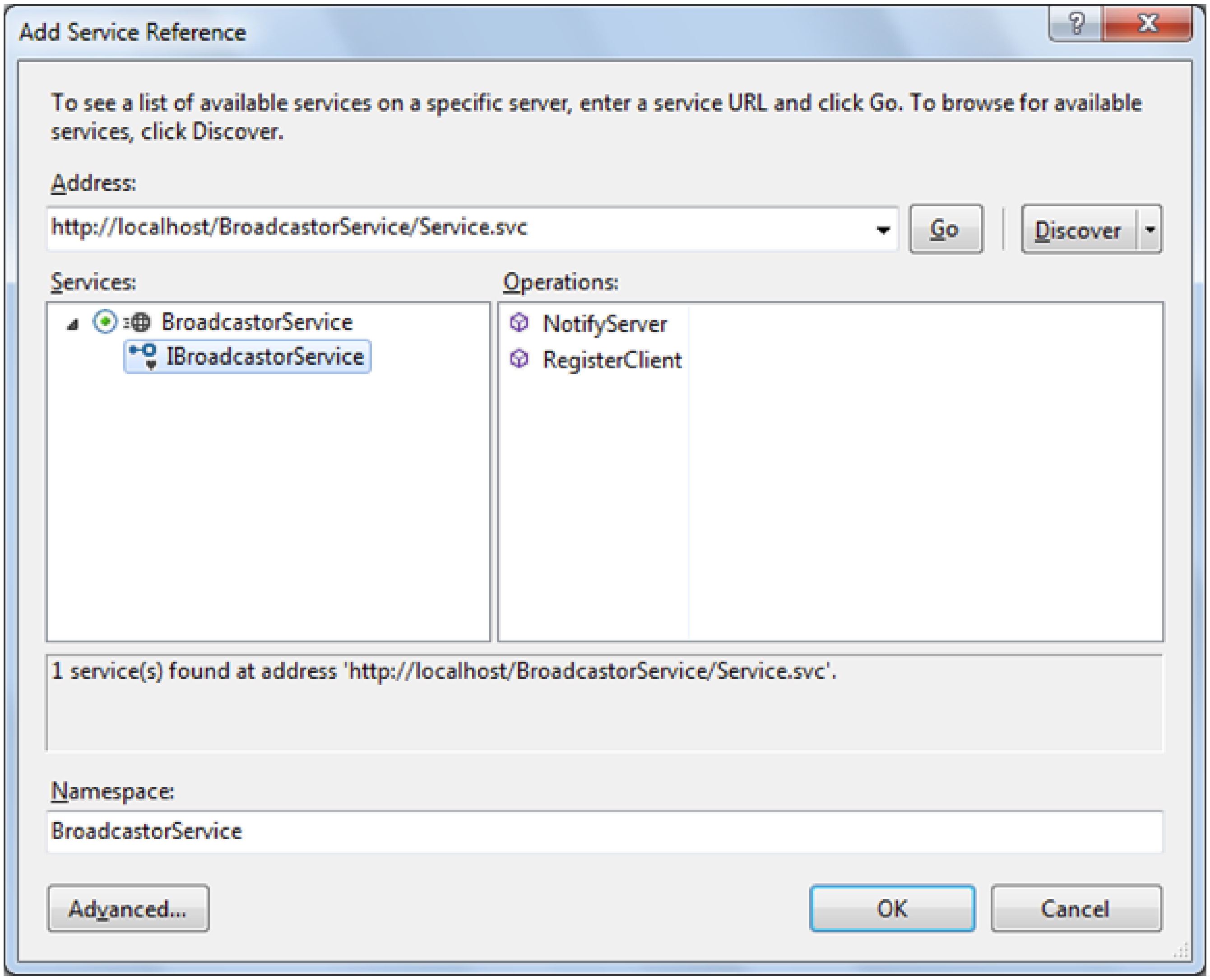


Name: BroadcastorClient
Location: C:\SOAwithWCFandLINQ\Projects\WCFBroadcastorSolution

Adding a service reference to the project

After the project has been added to the solution, we need to add a service reference to the service we have just created in previous sections.

1. Right click on the project item in solution explorer
2. Select Add Service Reference ... from the context menu
3. Enter http://localhost/BroadcastorService/Service.svc to the address box, and change the Namespace to be BroadcastorService
4. Click on button Go t download the wsdl of the service
5. Click on button OK to add the service reference


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/BroadcastorService/Service.svc

Services: BroadcastorService, IBroadcastorService

Operations: NotifyServer, RegisterClient

1 service(s) found at address 'http://localhost/BroadcastorService/Service.svc'.

Namespace: BroadcastorService

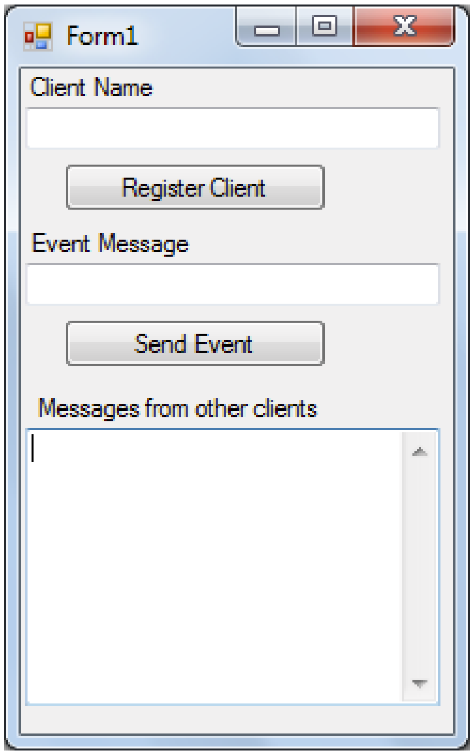
Adding controls to the form

Let's first add a few controls to the form, include:

1. A label with text/content Client Name
2. A text box named txtClientName
3. A button with text/content Register Client and named btnRegisterClient
4. A label with text/content Event Message
5. A text box named txtEventMessage
6. A button with text/content Send Event and named btnSendEvent
7. A label with text/content Messages from other clients
8. A text box named txtEventMessages. For Winform client, set its Multiline property to true, and Scrollbars to Vertical; for WPF client, set its VerticalScrollBarVisibility to Auto.

Note that the textbox txtEventMessage is the message that will be sent out from this clinet to other clients, while the textbox txtEventMessages will be the messages that are received by this client from all other clients.

The finished form should be like this:



Implementing the client

Next we will implement the client application. We need to define a class back class, and add a few event handlers for those buttons. We also need to add code to process events that are broadcasted from other clients.

1. Define the callback class: BroadcastorCallback

First we need to add a call back class. This class will implement the IBroadcastorServiceCallback interface on the WCF service, so the service can callback to the client when an event occurs on another client.

The class should be like this:

Hide Copy Code

```
public class BroadcastorCallback : BroadcastorService.IBroadcastorServiceCallback
{
    private System.Threading.SynchronizationContext syncContext =
        AsyncOperationManager.SynchronizationContext;

    private EventHandler _broadcastorCallBackHandler;
    public void SetHandler(EventHandler handler)
    {
        this._broadcastorCallBackHandler = handler;
    }

    public void BroadcastToClient(BroadcastorService.EventDataType eventData)
    {
        syncContext.Post(new System.Threading.SendOrPostCallback(OnBroadcast),
               eventData);
    }

    private void OnBroadcast(object eventData)
    {
        this._broadcastorCallBackHandler.Invoke(eventData, null);
    }
}
```

You can add this class to be an inner class of the client Form1/MainWindow class, or add it as a standalone class within the same namespace as the class Form1/MainWindow. In this article we will make it an inner class of Form1/MainWindow.

There are 3 methods in this class. The first one, SetHandler, is used to set the callback handler for the client. We will set this handler soon later when we register the client.

The second method, BroadcastToClient, is used to allow the service to call the client. When other clients send an event notification to the service, the service will connect to this client through the callback channel, then call this method to notify this client the event.

The third method, OnBroadcast, is the connection between the client callback handler, which is set in the first method, and the actual service call, which will be invoked by the service through the second method. When the service calls the second method, BroadcastToClient, to notify a event, the call will be delegated to this method, OnBroadcast, and then the same call will be delegated to the client callback handler.

2. Add method: HandleBroadcast

The method SetHandler in the callback class that we have defined in previous section will take an event handler object as its parameter. In this section we will define such a handler.

For the WinForms client, you can add following code to the Form1 class:

Hide Copy Code

```
private delegate void HandleBroadcastCallback(object sender, EventArgs e);
public void HandleBroadcast(object sender, EventArgs e)
{
    if (InvokeRequired)
    {
        BeginInvoke(new HandleBroadcastCallback(HandleBroadcast), sender, e);
    }
    else
    {
        try
        {
            var eventData = (BroadcastorService.EventDataType)sender;
            if (this.txtEventMessages.Text != "")
                this.txtEventMessages.Text += "\r\n";
            this.txtEventMessages.Text += string.Format("{0} (from {1})",
                eventData.EventMessage, eventData.ClientName);
        }
        catch (Exception ex)
        {
        }
    }
}
```

For the WPF client, you can add following code to the MainWindow class:

Hide Copy Code

```
private delegate void HandleBroadcastCallback(object sender, EventArgs e);
public void HandleBroadcast(object sender, EventArgs e)
{
    try
    {
        var eventData = (BroadcastorService.EventDataType)sender;
        if (this.txtEventMessages.Text != "")
            this.txtEventMessages.Text += "\r\n";
        this.txtEventMessages.Text += string.Format("{0} (from {1})",
            eventData.EventMessage, eventData.ClientName);
    }
    catch (Exception ex)
    {
    }
}
```

When the service broadcasts an event, it will invoke this handler on the client side to notify the client. Once notified, the client will get the event data from the sender parameter, and display the event message on the screen.

As you can see we are passing the event data as the sender to simplify the example here. In your real situation you should really define your own EventArgs type, and pass the event data with the parameter e, instead of the sender.

3. Add member variable : _client

Now we need to add a member variable to the main form/main window class, like this:

Hide Copy Code

```
private BroadcastorService.BroadcastorServiceClient _client;
```

As you will see in next section, this member variable will hold a reference to the broadcastor service proxy object, so we the client application can broadcast/receive events from the central service.

4. Add method : RegisterClient

Next let's add this method to the main form/main window class:

Hide Copy Code

```
private void RegisterClient()
{
    if ((this._client != null))
    {
        this._client.Abort();
        this._client = null;
    }

    BroadcastorCallback cb = new BroadcastorCallback();
    cb.SetHandler(this.HandleBroadcast);

    System.ServiceModel.InstanceContext context =
        new System.ServiceModel.InstanceContext(cb);
    this._client =
        new BroadcastorService.BroadcastorServiceClient(context);

    this._client.RegisterClient(this.txtClientName.Text);
}
```

This method will be the entry point of the whole broadcasting process. When a new client application starts up, it should call this method to establish a connection between this new client instance and the broadcastor service. As you can see, within this method we first create a callback client of the broadcastor service, set the callback handler to our customized method HandleBroadcast, then create an instance context using this callback client object, and finally initialize our _client member variable. The _client member variable now holds a reference to the service, and after we call the RegisterClient method, the service is notified of this new client instance. If the service receives a new event, the service can now invoke the HandleBroadcast method on this new client to notify the new event. We will test this soon.

5. Add event handler for button Register Client

In a real application, the method RegisterClient is normally called whenever a new client application is started, so the service can keep tracking all new clients. And ideally another method (which is not implemented in this article), UnregisterClient, is called whenever the client application is stopped, so the service doesn't need to broadcast to inactive clients.

In this article we will call the RegisterClient method when the Register Client button is clicked, so you will have a visual impression as when the client is connected to the service.

The clicked event handler for button Register Client should be like this:

Hide Copy Code

```
private void btnRegisterClient_Click(object sender, EventArgs e)
{
    if (this.txtClientName.Text == "")
    {
        MessageBox.Show(this, "Client Name cannot be empty");
        return;
    }
    this.RegisterClient();
}
```

When you start up a new client, you need to specify a name for it, so later on when broadcasting messages, you will know where that message is from. The client name could also be automatically generated using your own algorithm/formula, even a GUID will work here.

6. Add event handler for button Send Event

Now all the clients are connected with the service (assuming you have clicked the button Register Client on all clients). Next we need to implement the client to send an event to the service so the service can broadcast the event to all connected clients.

Again in a real application, an event could occur on various scenarios, like a piece of data has been changed by one client, a control is added/removed on the UI, or a button is clicked.

In this article we will trigger an event when the button Send Event is clicked, so you can send as many events as you like.

The clicked event handler for button Send Event should be like this:

Hide Copy Code

```
private void btnSendEvent_Click(object sender, EventArgs e)
{
    if (this._client == null)
    {
        MessageBox.Show(this, "Client is not registered");
        return;
    }

    if (this.txtEventMessage.Text == "")
    {
        MessageBox.Show(this, "Cannot broadcast an empty message");
        return;
    }

    this._client.NotifyServer(
        new BroadcastorService.EventDataType()
        {
            ClientName = this.txtClientName.Text,
            EventMessage = this.txtEventMessage.Text
        });
}
```

Within this method we first make sure the member variable _client is not null, meaning this new client has been registered with the service (a call back channel has been established between this client and the service). We then make sure there is something we want to send to the service (the event message), and finally we notify the service about the event.

Testing the service with our own clients

Now we have finished implementing the client and let's test it.

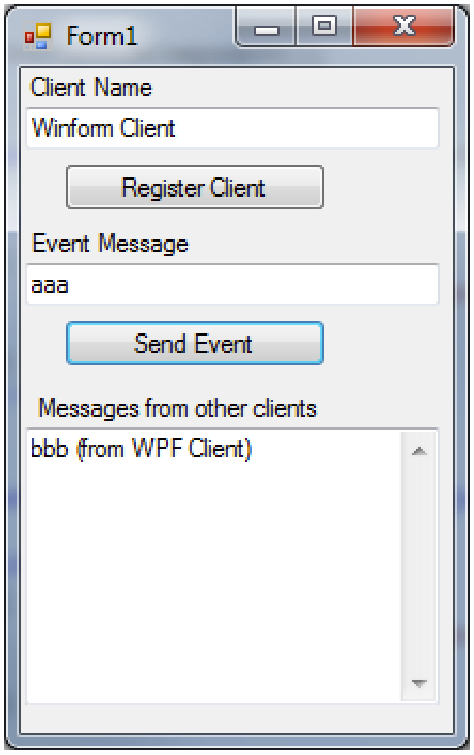
First make sure the Winforms client project is the startup project, then run it. Let's call this as Client A. Enter a name for it, like Winform Client, then click button Register Client.

Now go back Visual Studio and set the WPF client project as the startup project and run it. Let's call this as Client B. Again enter a name for it, like WPF Client, and click button Register Client.

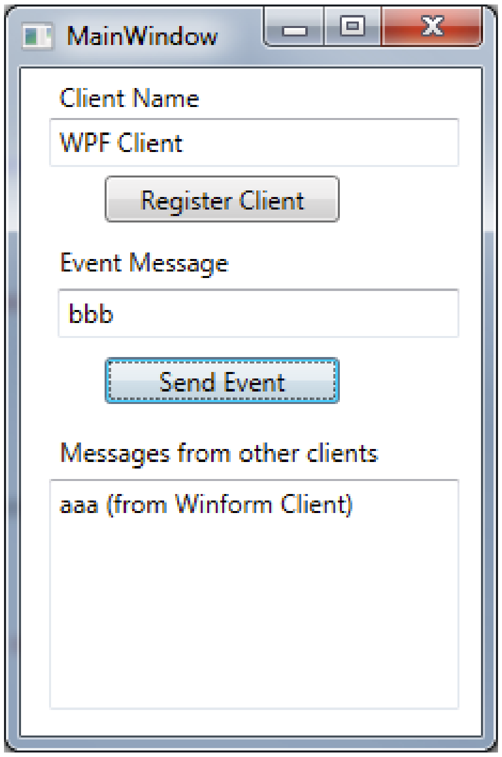
Next enter something on the textbox Event Message on Client A, like aaa, and click button Send Event. Guess what? This event is broadcasted to other clients, which is only Client B at this time.

Now enter something on the textbox Event Message on Client B, like bbb, and click button Send Event. Guess what? This event is broadcasted to other clients, which is only Client A at this time.

Following is Client A which just received a message from Client B:



And following is Client B which just received a message from Client A:



You can start another client application, and try to send a message from it, which will be broadcasted to both Client A and Client B.

Summary

In this article, we have implemented a duplex WCF service to broadcast events. A client will register itself when it starts up, and send a message to the service when an event occurs. The service will broadcast the event to all connected clients, and all clients will process the received event message accordingly.

Open issues

This article only gives you a simple idea as how to implement a duplex WCF service to broadcast events to all connected clients. There are a lot open issues not answered in this article, like:

- A client re-registers itself without un-registering first
- Client application is terminated abnormally (crashed or killed),
- Computer goes to sleep/standby
- Network is down between the client and the service
- Maximum concurrent connection limit of the service is reached
- Service is crashed/offline (queue up all outgoing messages and process them whenever the service is back?)
- Pin the service to re-connect automatically if the service has been restarted
- Event message buffer is too big to broadcast

Hosting the duplex WCF service in IIS on the Internet

Now tou have the service and the clients up and running on your local network, you may want to move the service to the Internet, so the clients can connect to the service through Internet. Thanks to reader BugRaptor, this has been tested and one such service is actually running on the Internet. You can even download a client to connect to that service, You can find out more about this service (and how to set up this kind of service) at this address:

http://stackoverflow.com/questions/30344977/hosting-a-wcf-service-using-net-tcp-binding-in-iis-7-unreachable-from-outside

Thanks BugRaptor for implementating such a service!

Author's Note

Part of this article is based on Chapter 2 and Chapter 3 of my book "WCF 4.5 Multi-Layer Services Development with Entity Framework (*Third Edition)*" (ISBN: 9781849687669). This book is a hands-on guide to learn how to build SOA applications on the Microsoft platform using WCF and Entity Framework. It is updated for VS2012 from my previous book: WCF 4.0 Multi-tier Services Development with LINQ to Entities.

With this book, you can learn how to master WCF, LINQ, Entity Framework and LINQ to Entities concepts by completing practical examples and applying them to your real-world assignments. It is ideal for beginners who want to learn how to build scalable, powerful, easy-to-maintain WCF Services. This book is rich with example code, clear explanations, interesting examples, and practical advice. It is a truly hands-on book for

C++ and C# developers.

You don't need to have any experience in WCF, LINQ, Entity Framework or LINQ to Entities to read this book. Detailed instructions and precise screenshots will guide you through the whole process of exploring the new worlds of WCF, LINQ, Entity Framework and LINQ to Entities. This book is distinguished from other WCF, LINQ, Entity Framework and LINQ to Entities books by that, this book focuses on how to do it, not why to do it in such a way, so you won't be overwhelmed by tons of information about WCF, LINQ, Entity Framework and LINQ to Entities. Once you have finished this book, you will be proud that you have been working with WCF, LINQ, Entity Framework and LINQ to Entities in the most straightforward way.

You can buy this book from Amazon at WCF 4.5 Multi-Layer Services Development with Entity Framework , or from the publisher's website athttp://www.packtpub.com/windows-communication-foundation-4-5-multi-layer-services-development-framework/book.

Update

The latest version of my book WCF Multi-layer Services Development with Entity Framework - Fourth Edition (for Visual Studio 2013 / Windows 7 and Windows 8.1) has published. You can get it directly from the publisher's website at this address

https://www.packtpub.com/application-development/wcf-multi-layer-services-development-entity-framework-4th-edition

or from Amazon at this address:

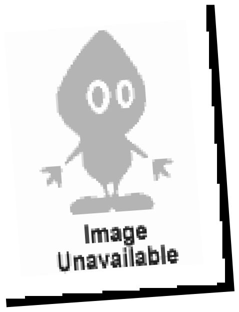
http://www.amazon.com/Multi-Layer-Services-Development-Entity-Framework/dp/1784391042

License

Share

EMAIL TWITTER

About the Author



Mike_Liu

Software Developer (Senior)

United States

Mike is a senior software engineer at an investment management firm, and a faculty member at Brandeis University, teaching Windows Communication Foundation programming with C#. He is a Microsoft Certified Solution Developer and a Sun Certified Java Programmer. He has authored a few books for Unix C/C++, C#, WCF and LINQ. Mike has deep expertise in software development with WCF, WPF, ASP.NET, MVC, BPMS, JavaScript, and SQL Server. He has been a key developer, architect and project lead for many software products in various industries, such as statistical data reporting, telecommunication, resource control, call center, student loan processing, and investment management.