module 27

wcf basics

INTRODUCTION

This module defines what WCF is and why you might use it in your own solutions. It shows you how to define, host and consume a basic WCF service.

OUTCOMES

By the end of this module, you should be able to do the following:

- ***Construct a simple end to end system of a WCF client consuming a WCF service.***
- ***Define an interface that can be exposed by a WCF host***

module 27: wcf basics

27.1 – Service Oriented Architecture (SOA) Comparison

In the OOP world code would be written to be reusable. Source code could be written for one project, and then be copied and pasted into another application when it requires the same functionality. The problems with this is there is now code duplication going on, and changing code (bug fixes, new business logic) in one place will not update the code in other areas (DLLs can be used, but these still have similar problems with deployment and versioning). Further problems arise when trying to share functionality with other departments/businesses.

SOA unifies business processes by structuring large applications as an ad-hoc collection of smaller components called 'services'. These services can be used by people both inside and outside the company, and new applications can be built by combining these services into new ways.

Instead of having every application having to handle the creation or retrieval of customer records, this can be created as a standard service which all applications can consume.

In summary:

- **OOP**
  - "Code Reuse"
  - Deployed code would have to be recompiled and redeployed
- **SOA**
  - Progression from 'code reuse' to 'Service Reuse'
  - Standard functionality can be written once, applications can consume this service
  - Logic only needs to be updated in a single place

Building all applications from the same common set of services reduces development time and when exposed to external consumers, can make the goal of transactions bridging companies easier.

A good example of a service on the web is Amazon.com. They have services exposed which allow you to query and retrieve book details in a standardised fashion.

27.1.1 – The Four Tenets of SOA

1. Explicitly expose functionality

- Location is irrelevant
- Technology is irrelevant

2. Services are autonomous
    - they are Operated independently
3. Services share Schema and Contract, not Class
    - Only deal with what is allowed to pass in and come out.
4. Compatibility is based upon Policy
    - Every service advertises its capabilities and requirements in the form of a machine-readable policy expression

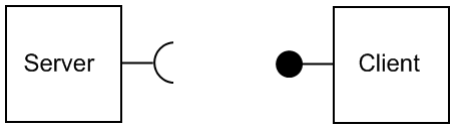
27.2 – Windows Communication Foundation

WCF is an SDK for building SOA on Windows and was introduced in .NET 3.0. WCF combines many previous Microsoft messaging technologies together into a standard coding framework. A service is written as a logical piece of work and is then exposed using whichever method is most appropriate. This keeps the service logic and the technical transportation pieces of the service separate and reduces the complexity.

- Services send and receive messages
- All messages are SOAP messages
- WCF services may interoperate with non-WCF services
- Brings together all of the disparate Microsoft messaging technologies
    - ASMX web services
    - .NET remoting
    - Microsoft Message Queuing

27.2.1 – Overview

- Server exposes a service
- Client creates a 'proxy class' of the service using metadata
- Client connects and consumes the service by making method calls as if the class was local
- .NET handles the plumbing to get from A to B



27.3 – WCF Endpoints - the ABC of WCF

An easy way to remember how a WCF service is defined is to remember its ABCs:

- **A**ddress - where the service is
- **B**inding - how to talk to the service

- **C**ontract - what the service can do

Consider the following real world analogy. The address of the service is where it is located; this is unique to it just like the address of a house. Just as there can be many houses with unique addresses the same goes with WCF, each address can only be used for one service. Addresses must be unique.

Now that you know where you are going, the next question is how are you going to get there? You could go by truck or train or walking, each has its own advantage/disadvantage. The same question needs to be asked of the messages sent with WCF, how will they travel to where they are going? This could be via HTTP or TCP, or could be guarded with extra security as it travels.

Contracts are the menu of available functions the service provides.

27.4 – Exposing endpoints

- Every service must expose at least one endpoint
- Each endpoint has exactly one contract
- All endpoints should have unique addresses
- Single service can expose multiple endpoints
    - Can use same or different bindings
    - Can expose same or different contracts
- Are set up in a config file separated from the code

27.4.1 – Address

The address is a combination of the **server name**, **port number** and **path**. The transport at the beginning will be determined by the binding e.g. HTTP.

27.4.2 – Bindings

There are numerous ways that a message can be formatted/sent/secured, this allows you to tailor your service for the compatibility/performance you require for your solution.

- Transport
    - HTTP
    - TCP
    - MSMQ
- Message formats and encoding
    - Plain text
    - Binary
    - MTOM
- Communication security

  - No security
  - Transport security
  - Message security
  - Authenticating and authorizing callers

Because there is such a large number of choices that can be made these have been packaged into 'standard binding' sets for easy use.

A service can support multiple sets of bindings, however each must be on a separate address. The Client must use exactly the same binding as service otherwise they will not be able to communicate properly.

27.4.3 – Standard Bindings

These standard binding sets mostly relate back to each of the previous Microsoft message technologies that were combined under the WCF banner.

- **BasicHTTP** – for integration with legacy WebService clients
- **TCP** – messages are sent over TCP
- **Peer TCP** – Peer to Peer (P2P) networks
- **Named Pipe** – memory streams for inter-process communication
- **WS** – Web services with support for security, reliability, etc.
- **MSMQ** – MicroSoft Message Queuing

The standard bindings can be customised and new custom bindings can be created.

27.4.4 – Contracts

Each service has three main sets of contracts

- **Service contracts**
  - List of operations the service is exposing
- **Data contracts**
  - Allows you to create 'data objects' to be sent between client/server. Specifies the schema.
- **Fault contracts**
  - Abstraction away from "Exceptions"
  - Which errors can raised by the service. To allow handling of them

To expose a class and methods as a service you need to decorate them with metadata, this allows .NET to map the CLR to WCF. A single Class can implement multiple **ServiceContracts**; it just needs to implement multiple Interfaces. The

methods exposed with `[OperationContract]` can only use primitive or data contracts as parameters, i.e.:

- **[ServiceContract]** – Defines a 'set' of operations
- **[OperationContract]** – Defines a single method

```
[ServiceContract]
public interface IService1
{
    [OperationContract]
    string GetData(int value);
}

public class Service1 : IService1
{
    public string GetData(int value)
    {
        return string.Format("You entered: {0}", value);
    }

    public string OtherMethod()
    {
        return "This method can't be called by WCF";
    }
}
```

27.5 – MetaData

A service can expose multiple endpoints with different bindings and addresses. It can also expose a MetaData endpoint which allows clients to discover how to interact with the service. The bindings used, operation contracts and data contracts are all defined allowing the client to consume the service without having to contact the developers. MetaData endpoints allow WCF to be self describing.

27.6 – Hosting

The available hosting options are

- **IIS**
  - HTTP only
- **WAS** (Windows Activation Service)
  - Can use any transport
  - Vista and Server2008 only
- **Self hosting**
  - Can use any transport
  - Useful for communicating between applications
  - Self hosting can be done in any .NET application: WPF, Winforms, WF, console, windows service, etc.

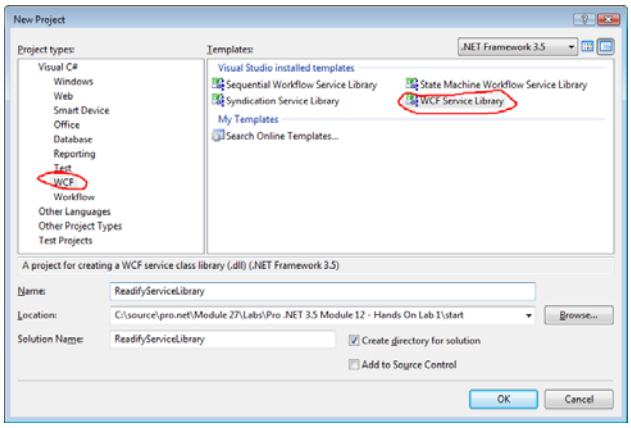
An advantage of hosting on IIS or WAS over self hosting is that the process needs to be running constantly with a self hosted solution, whereas one hosted on IIS or WAS isn't launched until there is a client request.

module 27: hands on lab

Exercise 1

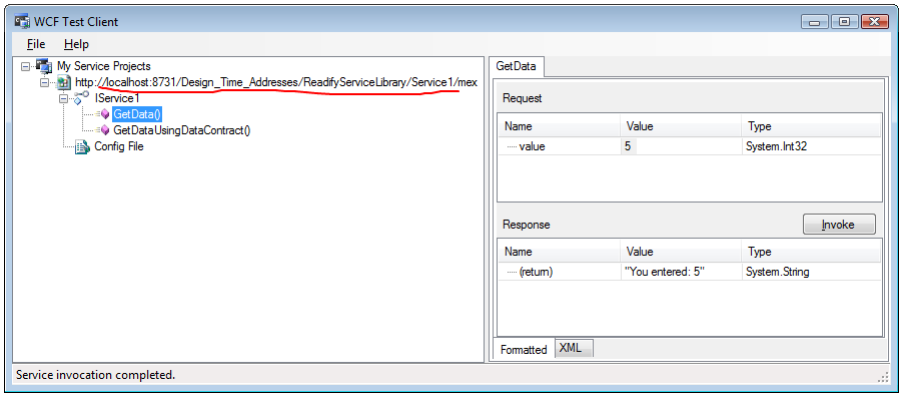
This exercise is designed to show you how to create a WCF Service Library that can be reused in multiple hosting environments in later exercises. For this exercise Visual Studio will be used to host the service to show how you can create/debug a WCF service without worrying about the hosting context as you develop.

1. Open a new instance of Visual Studio 2008 and create a new **WCF Service Library**. Save it anywhere, but the Lab1/start folder would be a good place.



   This will create a new WCF Service Library which can be hosted in any .NET application. The template creates a basic service for you that will take an integer and return a string. It also accepts a custom data object through the DataContract.

2. Press F5 to compile and run the project.



   Visual Studio will host the WCF Service Library in a custom host. A frontend is then displayed which allows you to invoke methods on the service to test them. In the screenshot you can see highlighted the MetaData address of the Service as it is being hosted at the moment. You can use this to test the methods without needing to create a client application to test it with.

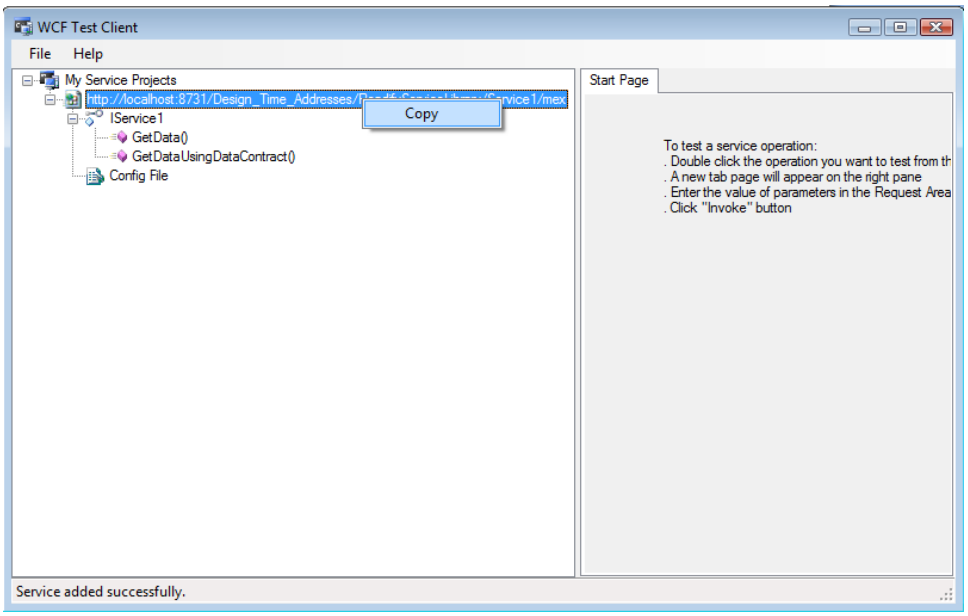
3. In the **WCF Test Client** frontend, double click the **GetData()** method and try invoking it with different values. This method just takes an integer primitive as a parameter.
4. Double click the method **GetDataUsingDataContract**. This method takes a DataContract object called `composite`. The testing interface is smart enough to expose the data fields allowing you to send through a DataContract object to test.

Exercise 2

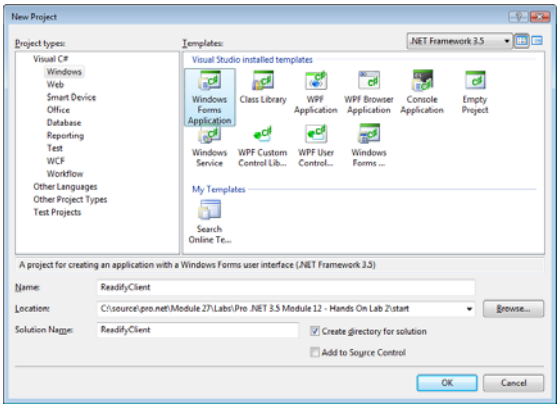
In this exercise we will use the ServiceLibrary produced in Exercise 1 and create a client to consume it. We will modify the bindings to demonstrate the configuration tool.

Creating the client

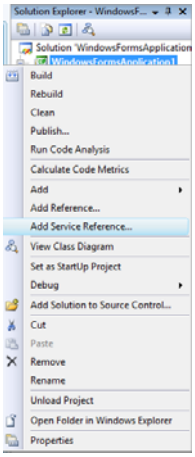
1. Open the Start solution for Exercise 2 and run it by pressing F5. The WCF Service is now being hosted by Visual Studio so that we can develop with it.
2. Get the URI for the service from the testing interface - right click and select **Copy**.



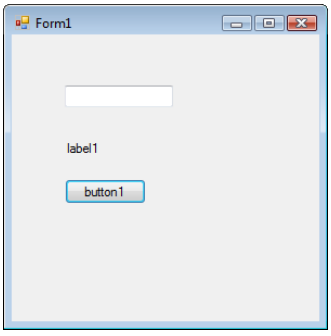
3. Open a new instance of Visual Studio 2008 and create a new Windows Forms application and save it in the same location as the Start solution.



4. Right click on the project in the solution explorer and create a new **Service Reference**



5. Paste the URI of the service into the address bar and click **Go**. It will hit the metadata endpoint of the service and extract information on how to communicate with the service. If you expand the tree you can see all the contract interfaces that the service implements and are available for your consumption.
6. Change the namespace to `ReadifyServiceReference` and click OK to have Visual Studio automatically create the proxy class. This class handles all of the complicated plumbing for you, saving you from having to look at it.
7. Drag a TextBox, Label and Button onto the form.



8. Double click on the button and enter this code

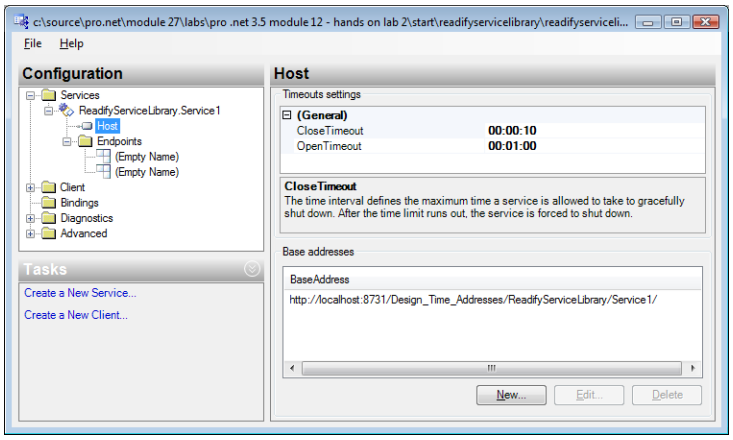
```
int number = int.Parse(textBox1.Text);

using
   (ReadifyServiceReference.Service1Client serviceproxy =
    new ReadifyServiceReference.Service1Client() )
    {
        label1.Text = serviceproxy.GetData(number);
    }
    textBox1.Text = (number + 1).ToString();
```

The using statement will automatically close the WCF proxy when the process is finished, saving you from having to remember to close the connection manually.

9. Press F5 to run the application.
10. Enter a number into the textbox and click the button. Keep clicking the button for fun, it should automatically increase for to let you know the requests are going through successfully

Bindings

11. Stop running both the client and the WCF service. Go to the instance of Visual Studio with the **WCFServiceLibrary** project. Right click the **app.config** file and select **Edit WCF configuration.**
    Note: If this is your first time opening the configuration utility it may not be present as an option. Select the app.config file then use the **Tools->WCF Service Configuration** menu command.



    If you expand the tree you will be able to see two endpoints defined: one for the service and the other as the metadata endpoint.
12. Add a new base address for TCP services. Do this by clicking on the host node in the tree, copy the current base address and then clicking New. Now paste the current base address in, but increment the port number by one and change the transport to Net.TCP. For example,

Original:
http://localhost:8731/Design_Time_Addresses/ReadifyServiceLibrary/Service1/
New:
Net.TCP://localhost:8732/Design_Time_Addresses/ReadifyServiceLibrary/Service1/

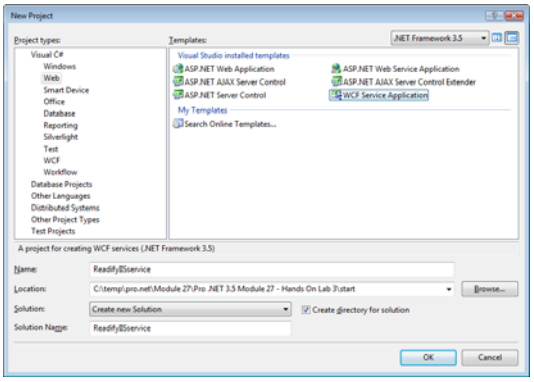
13. Click on the current endpoint and change it from wsHttpBinding to **netTcpBinding**
14. Press F5 run the WCFServiceLibrary project again. The metadata is now still being shared at the Http address **http://localhost:8731/Design_Time_Addresses/ReadifyServiceLibrary/Service1/mex**
    But the actual binding for the client and server is happening over TCP at address **net.tcp://localhost:8732/Design_Time_Addresses/ReadifyServiceLibrary/Service1/**
15. Open the **app.config** file in the client Visual Studio project to notice that the endpoint will still point to an http address.
16. While the WCF Service is running, go to the client Visual Studio instance, right click on the service reference in the Solution Explorer and select the **Update Service Reference** menu item. It will detect the change in bindings and update it for you.
17. Inspect the app.config to verify that the endpoint address has been changed to use **net.tcp**.
18. Press F5 to run the client and verify it all still works
19. Repeat this process trying different binding types. Remember that if you change the binding or address of the metadata, you will have to tell the client where the new address is. Right click on the service reference and configure, then change the new address of the metadata.

Exercise 3

This exercise we will discover how to host the WCFServiceLibrary in IIS. We will use the Library that we created in Exercise 1

Creating an ASP.Net endpoint

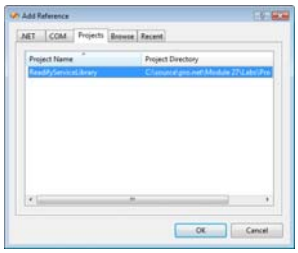
1. Create a new ASP.NET WCF web service in the Start solution for Exercise 3. Name it **ReadifyIISservice**.



2. Delete the files **IService1.cs** and **Service1.svc.cs**. These files are where the service code was written, but since we will be using a service we have already created we no longer need these.
   Be sure **not** to delete **Service1.svc** as this will be the page hosting the service.

Adding the existing Service Library

3. Right click the solution and add an existing project - select the project in Lab3\start\ReadifyServiceLibrary. This is the WCF implementation we created in a previous exercise.
4. Right click the website project and add a reference to the existing service library. Select **projects** to see all the projects in the solution and add a reference to the ReadifyServiceLibrary



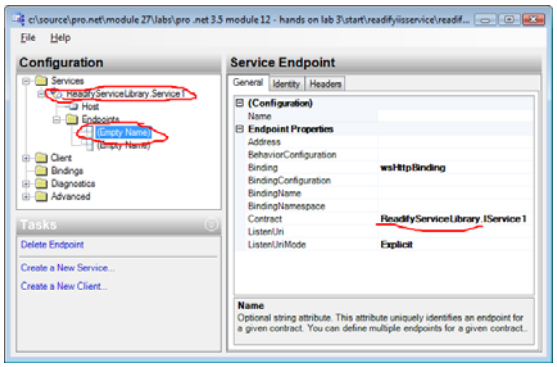
Configuring to host the service library

5. Open Service1.svc in the website project.
6. Edit the line to load the service library we have previously created. Remove the code behind definition and change the fully qualified name to point to the ReadifyServiceLibrary

```
<%@ ServiceHost Language="C#" Debug="true"
Service="ReadifyServiceLibrary.Service1"
CodeBehind="Service1.svc.cs" %>
```

Now that we have the page loading the correct service class, we just need to correct the endpoint definition.

7. Right click the web.config file and edit the **WCF configuration**. Change the service name and the endpoint contract from ReadifyIISService to our ReadifyServiceLibrary



8. Right click **service.svc** and set this to be the startup page.
9. Press F5 and our service should be loaded and telling us how to interact with it.

This service is now hosted on IIS and can be deployed to any IIS server and will accept client requests. Developing WCF services this way (library first rather than just creating an ASP.NET WCF service and writing code) allows more flexibility in how you decide to finally host the WCF service, if later you decide to host it under WAS, then you will have to do some work to extract the service out of this solution.