Services and *Windows Communication Foundation* - WCF

Definition

Endpoints

ABC – Address, Binding, Contract

Reuse from Libraries to Services

C++ and Java

Classes and objects in OO

Same process
How:
Include files, code files

White - box

COM

Components

Several processes
Same machine
Interfaces (IDL)

DCOM

Components

Several processes
Different machines
Interfaces (IDL)

COM+

Components

Several machines
Interfaces
Containers
Configurable aspects

ASMX WS, .Net Remoting

Components

Several machines
Interfaces
Hosts (IIS, other procs)

Black - box

WCF

Services

Unifying (several protocols)
Interoperable
Interfaces (contracts)
Hosts
Configurable aspects

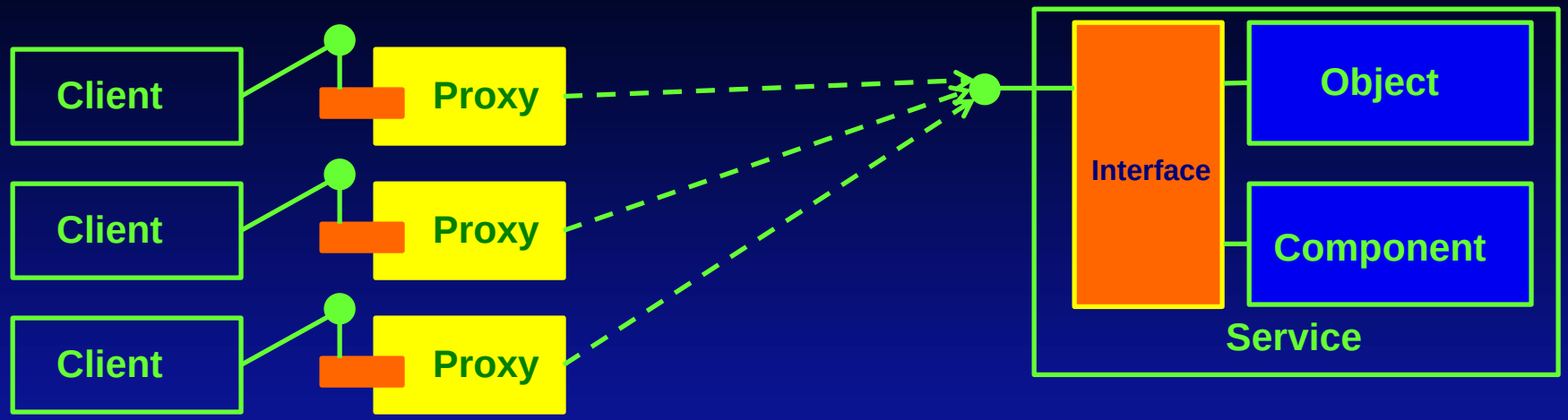
White-box versus Black-box

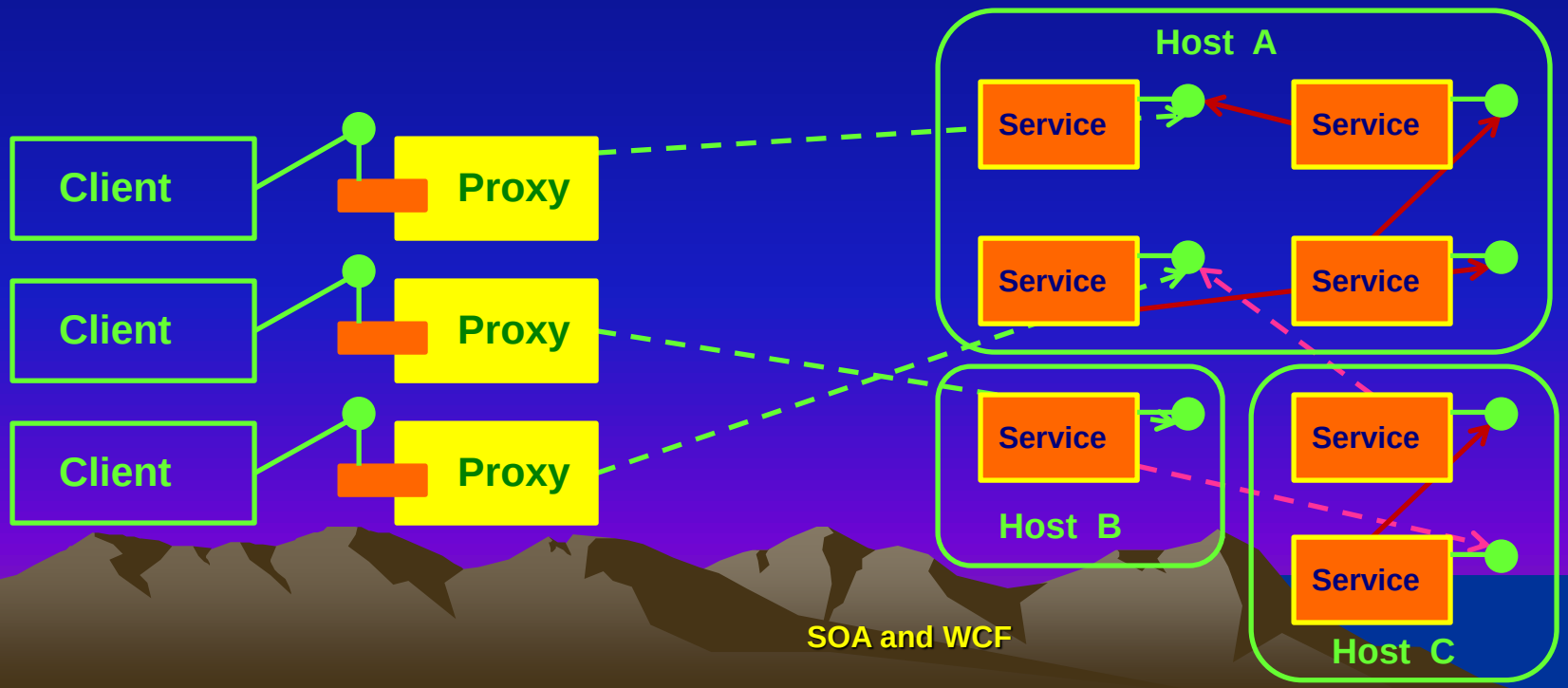
- Need for code reuse
  - Has motivated the adoption of OO languages over procedimental
- White-box reuse needs and characteristics
  - Detailed knowledge of class objects at every development level
  - Clients linked and dependent of class libraries
  - Changes in classes, belonging to the libraries, can cause significant impacts in clients
- Black-box reuse needs and characteristics
  - Only knowledge of an interface (contract)
  - Separation between definition and implementation
  - Clients don’t need to know anything about implementation

Services and SOA

- Services
  - Promote increasing degrees of decoupling (separation)
  - Next stage of component evolution, the same way as components (interface based) were an evolution from objects
- Services are today, in many situations, the best way to build complex distributed applications
  - Promote productivity
  - Promote a good evolution and maintenance
  - Promote extensibility
  - Promote an easy and real reuse
- The functionality (business process activities) implementation are based on
  - Standards and protocols
  - General policies (transactions, reliable messaging, …)
  - Contracts (WSDL)
  - Messaging

Service Oriented Architecture & Apps



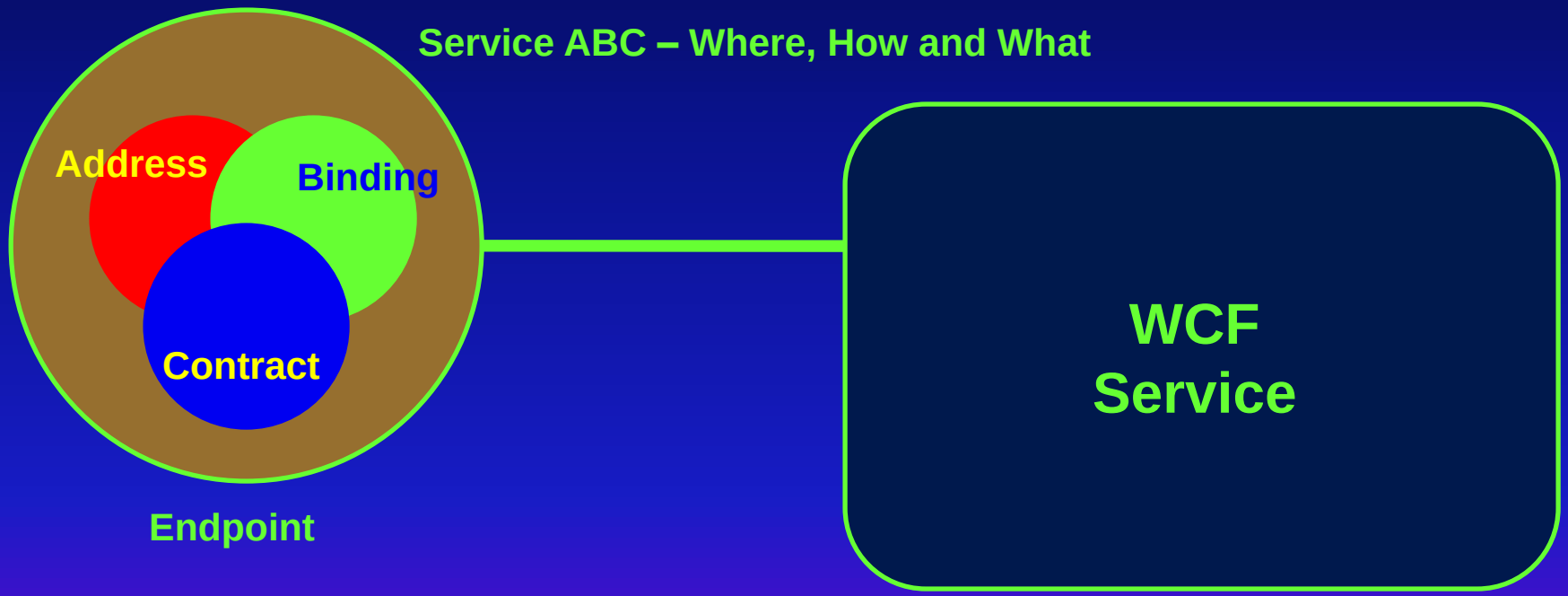


WCF

- Technology unification
  - Unification through common principles and APIs
  - Integrates functionalities from:
    - COM+ (transactions, queued components, events, …)
    - .NET Remoting
    - Message Queues
    - XML and REST Web Services
    - WS-* web services
    - Websockets
- Promotes best practices through SOA principles
- Appropriate for intra-machine, intranet and internet applications
- Interoperable with other technologies
  - Other Microsoft distribution technologies
  - Other distribution *stacks* based on WS, WS-*, REST and Websockets

A WCF Service

- WCF services are defined, installed and consumed through ***endpoints***

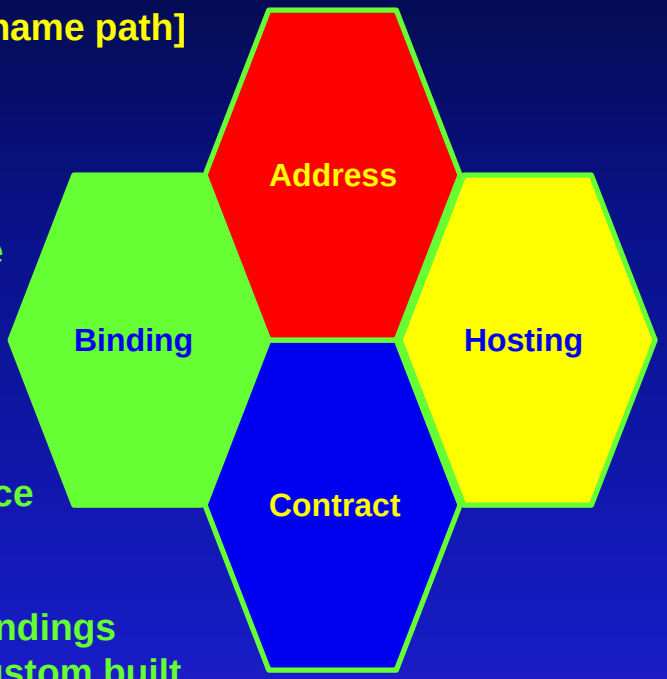


WCF ABC

Address – Identifies a service unambiguously and provides its location:

[transport]://[machine][[:port]][name path]

Examples:
http://localhost
http://localhost:8081/Service
net.tcp://mach/Services/Service
net.pipe://localhost/Pipe



Contract – Define available operations, data types and errors.

Some contract types that we can specify:

Service Contract
Operation Contract
Data Contract
Fault Contract

Binding – Specify several service protocols (info exchange) and other characteristics.
There are several predefined bindings (ready to use) or they can be custom built

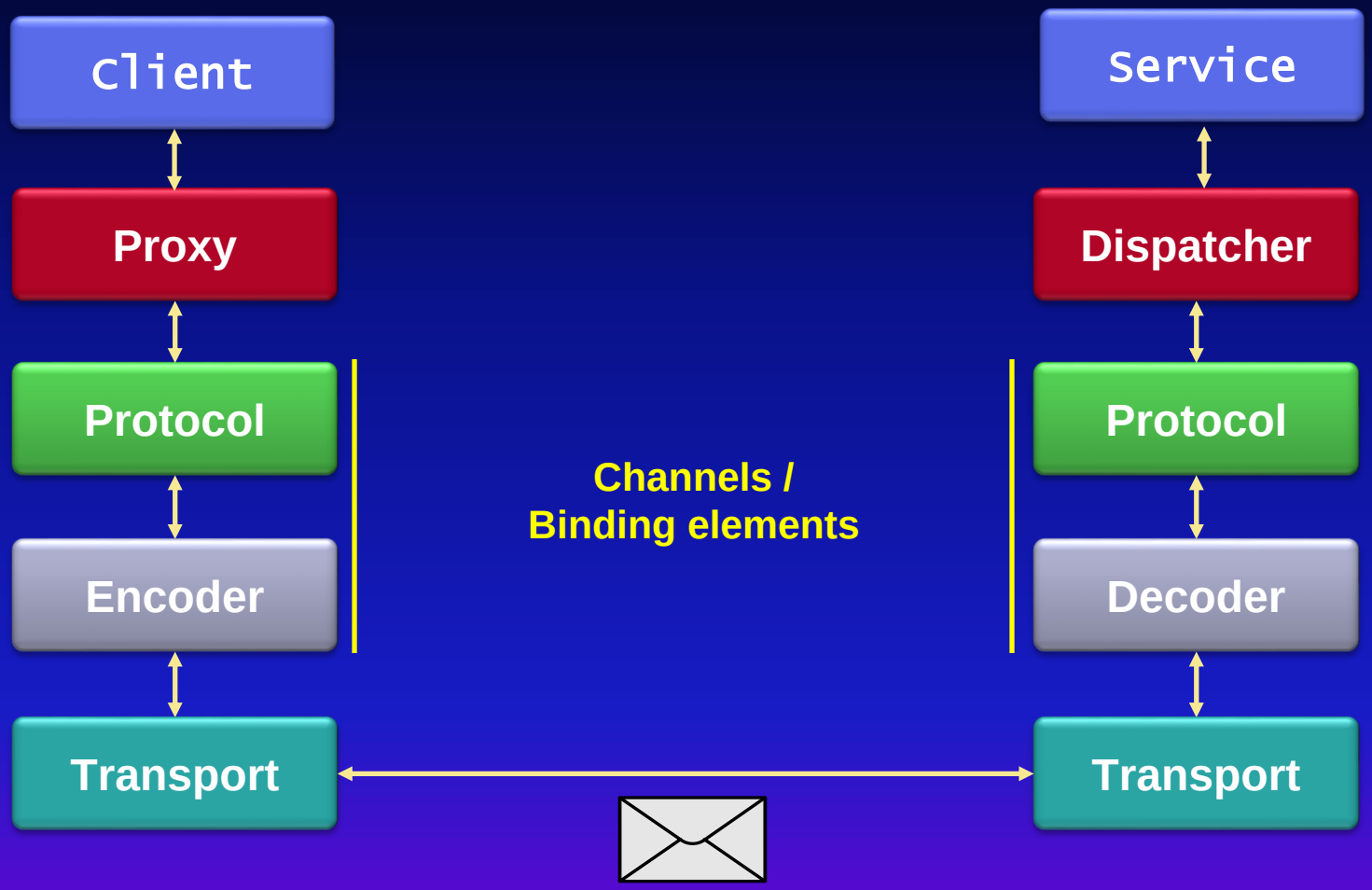
Examples:
BasicHttpBinding
NetTcpBinding
WSHttpBinding
NetMsmqBinding

Hosting – Define in what server we run the service and how we make it available

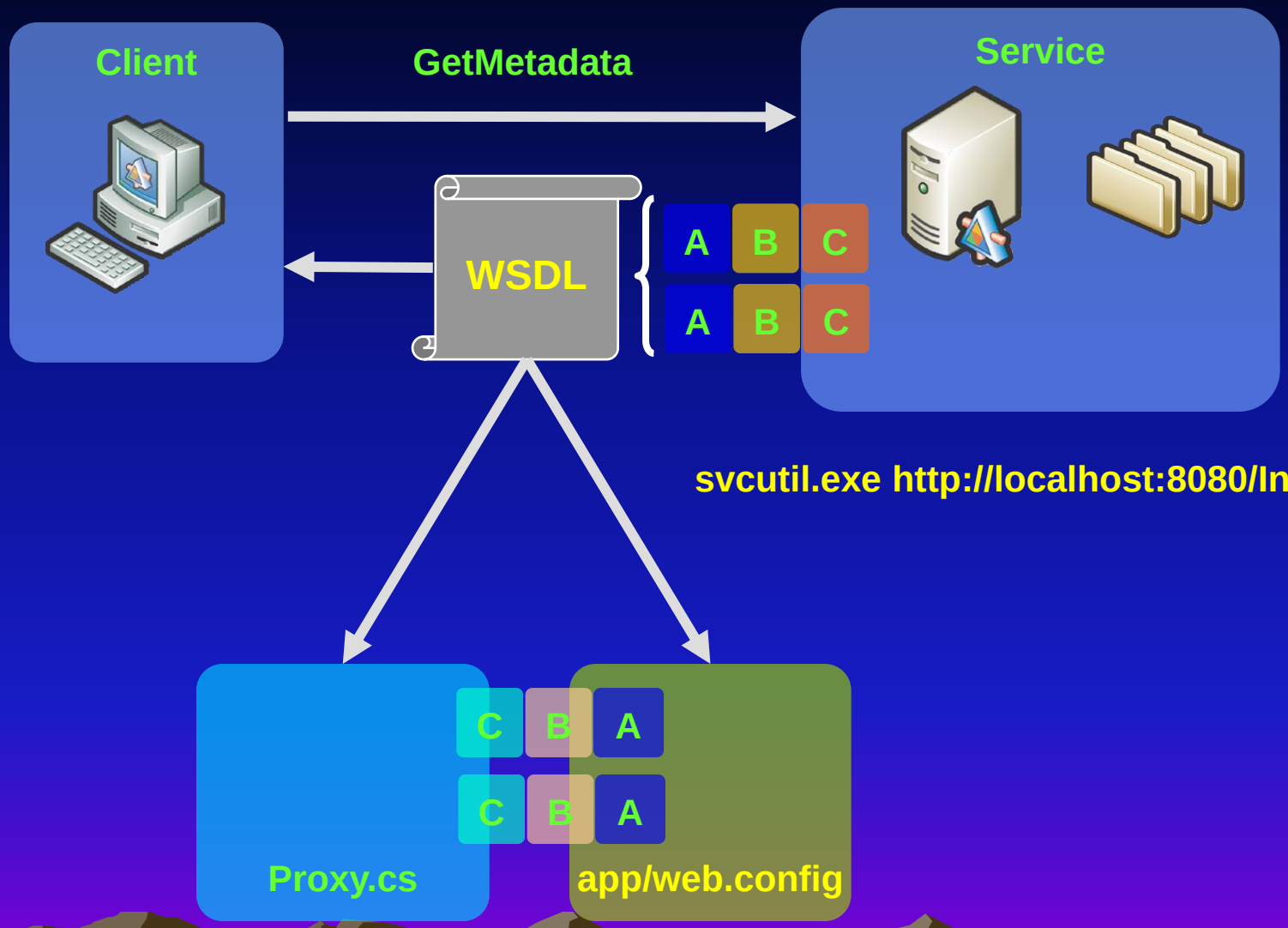
Some options:

IIS
Windows Activation Service
Windows Application
Console Application
Custom Windows Service (OS)

Architecture



Client Configuration



svcutil.exe http://localhost:8080/Interop/

Developing a New WCF Service

- Define your contract writing an interface
  - Write an interface and decorate it with attributes
    - [ServiceContract] and [OperationContract]
  - It may include data type definitions decorated with attributes
    - [DataContract] and [DataMember]
- Implement the service
  - A class implementing the contract
- Define the service Host and write it if you need to
- Define the configuration in a config file containing the endpoints specification through their ABC parameters
- The three first items can be put in the same *assembly*

Contracts

```
using System.ServiceModel;

namespace MyWCFSvc {

    [ServiceContract]
    public interface IStockService {

        [OperationContract]
        Info GetStockPrice(string ticker);
    }

    [DataContract]
    public class Info {

        [DataMember]
        public string name;

        [DataMember]
        public double value;
    }
}
```

Operations

Immutable aspects
in the service definition

Data

Service Implementation

```
using System.ServiceModel;

namespace MyWCFSvc {

    public class StockService : IStockService {

        public Info GetStockPrice(string ticker) {
            Info stock = new Info();
            stock.name = ticker;
            stock.value = 9.38;
            return stock;
        }

    }

}
```

Must derive from the interface defined as a [ServiceContract]

All the methods defined as [OperationContract] must be also present

The implementation can include other methods

Data classes defined as [DataContract] can be used as parameters or return values

Host Implementation (as a console app)

```
using System.ServiceModel;

class Program {

    static void Main(string[ ] args) {
        private ServiceHost SHost = null;

        SHost = new ServiceHost(typeof(MyWCFSvc.StockService));
        SHost.Open();

        Console.WriteLine("Service open.  Press <Enter> to terminate.");
        Console.ReadLine();

        SHost.Close();
    }

}
```

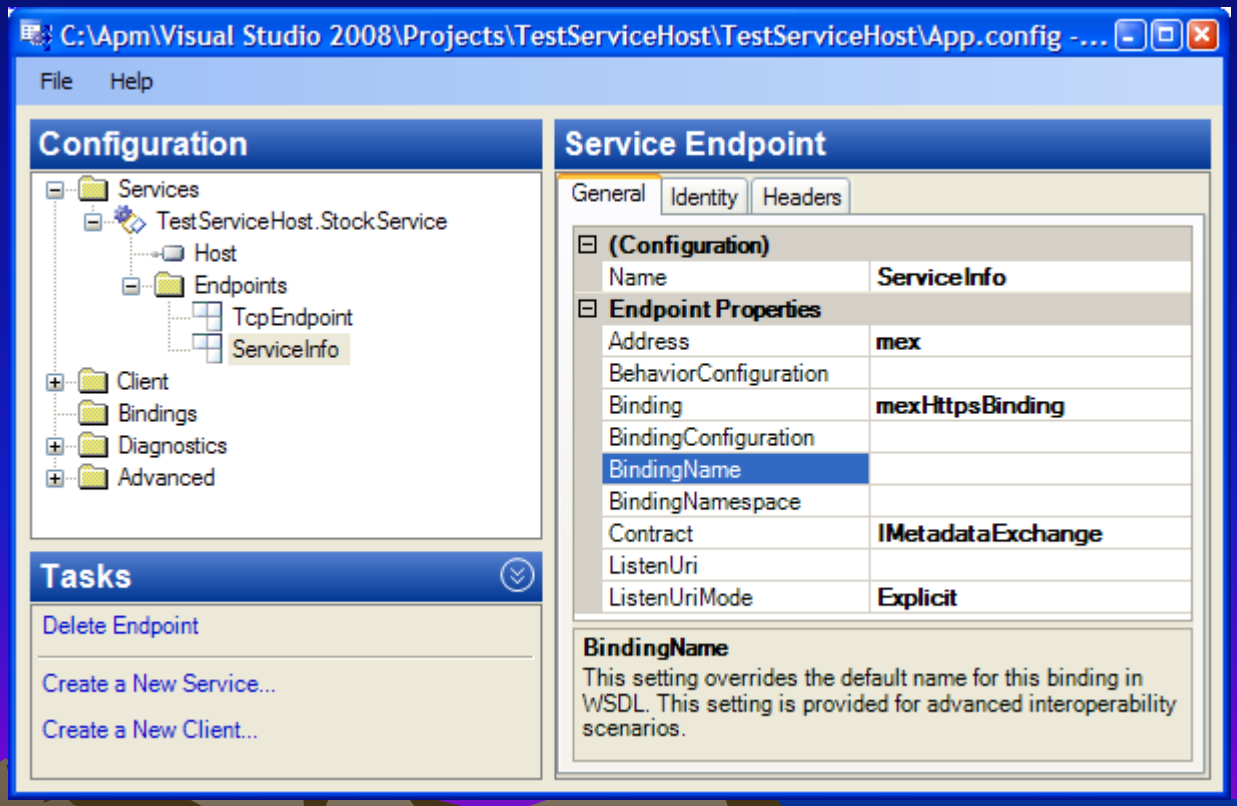
Configuration File

```
<configuration>
  <system.serviceModel>
    <services>
      <service behaviorConfiguration="StockServiceBehavior"
        name="StockService">
        <host>
          <baseAddresses>
            <add baseAddress=net.tcp://localhost:8700/TestService/StockService />
            <add baseAddress=http://localhost:9000/TestService/StockService />
          </baseAddresses>
        </host>
        <endpoint address="" binding="netTcpBinding“
                                                contract="MyWCFSvc.IStockService" />
        <endpoint address="mex" binding="mexHttpBinding"
                                                contract="IMetadataExchange" />
      </service>
    </services>
    <behaviors>  <serviceBehaviors>
      <behavior name="StockServiceBehavior">
        <serviceMetadata httpGetEnabled="true" />
      </behavior>
    </serviceBehaviors>  </behaviors>
  </system.serviceModel>
</configuration>
```

Configuration File Editing

Configuration files are generated by Visual Studio in WCF Services projects
- Web WCF Service Application (Service to be installed in IIS)
- WCF Service Library (Service in a .dll, ready for hosting)
- Add new item (WCF Service), inside any Windows application project

Configuration files can be edited by an external tool (also integrated in VS):
SvcConfigEditor.exe



Metadata Exchange httpGetEnabled

With the httpGetEnabled behavior we can see in the browser some information about the service

StockService Service - Microsoft Internet Explorer

http://localhost:9000

StockService Service

You have created a service.

To test this service, you will need to create a client and use it to call the service. You can do this using the svcutil.exe tool from the command line with the following syntax:

```
svcutil.exe http://localhost:9000/TestServiceHost/StockService?wsdl
```

This will generate a configuration file and a code file that contains the client class. Add the two files to your client application and use the generated client class to call the Service. For example:

C#

```
class Test
{
    static void Main()
    {
        StockServiceClient client = new StockServiceClient();

        // Use the 'client' variable to call operations on the service

        // Always close the client.
        client.Close();
    }
}
```

Internet 100%

Client Implementation

Generate a proxy and configuration file using the command line tool svcutil.exe

```
> svcutil  http://localhost:9000/TestService/StockService/mex
        -config: app.config
        -out: generatedProxy.cs
```

or

> use Visual Studio “Add Service Reference …”

```
class Program {
  static void Main(string[ ] args) {
    string ticker = "MSFT";

    Console.WriteLine("Asking for {0}", ticker);
    StockServiceClient proxy = new StockServiceClient();
    Info info = proxy.GetStockPrice(ticker);
    Console.WriteLine("Response:  name = {0}  value = {1}", info.name, info.value);
    proxy.Close();
  }
```

Client without a Proxy

```
… ServiceContract …
… DataContract …

class Client {
   static void Main(string[ ] args) {
      channelFactory<IStockService> chFactory = new channelFactory<IStockService> (
                                    new NetTcpBinding(),
                                    new EndpointAddress(
                                          “net.tcp://localhost:8700/TestService/StockService”) );

      IStockService svc = chFactory.CreateChannel();

      Info info = svc.GetStockPrice(“IBM”);

      . . .
}
```

Main Predefined Bindings (1)

- basicHttpBinding
  - Classic (XML) Web Services: compatible with basic profile 1.1 from WSI
  - Uses Http as transport
- wsHttpBinding
  - Web Services compatible with many of the WS-* specifications (Transactions, Security, MTOM, ReliableMessaging, …)
  - Uses Http as transport
- ws2007HttpBinding
  - Allows other versions and specifications for WS-* services
- wsDualHttpBinding
  - Allows duplex communications in Http with a contract also defined at the client side
- webHttpBinding
  - Implements Web Services with REST/POX style and JSON/XML encoding
  - Uses Http as transport

Main Predefined Bindings (2)

- netHttpBinding
  - Uses the Websocket protocol to implement the server side
- netTcpBinding
  - Uses a binary serialization and TCP transport
  - It is the most performant binding
- netNamedPipeBinding
  - Similar to the previous, but uses a pipe as a communication channel
  - Most adequate for service and client on the same machine
- netMsmqBinding
  - Uses asynchronous communications through a message queue (MSMQ) in a transparent way
- msmqIntegrationBinding
  - Allows invocations and responses directly manipulating a message queue

Main Predefined Bindings (3)

- udpBinding
  - Uses the UDP network protocol
- netPeerTcpBinding
  - Uses peer-to-peer network connections
- wsFederationHttpBinding
  - Allows the implementation of Web Services using advanced specifications from WS-* and a federated identity
  - Uses Http as transport
- ws2007FederationHttpBinding
  - Similar to the previous but allowing other specification versions with federated security