Programming Services

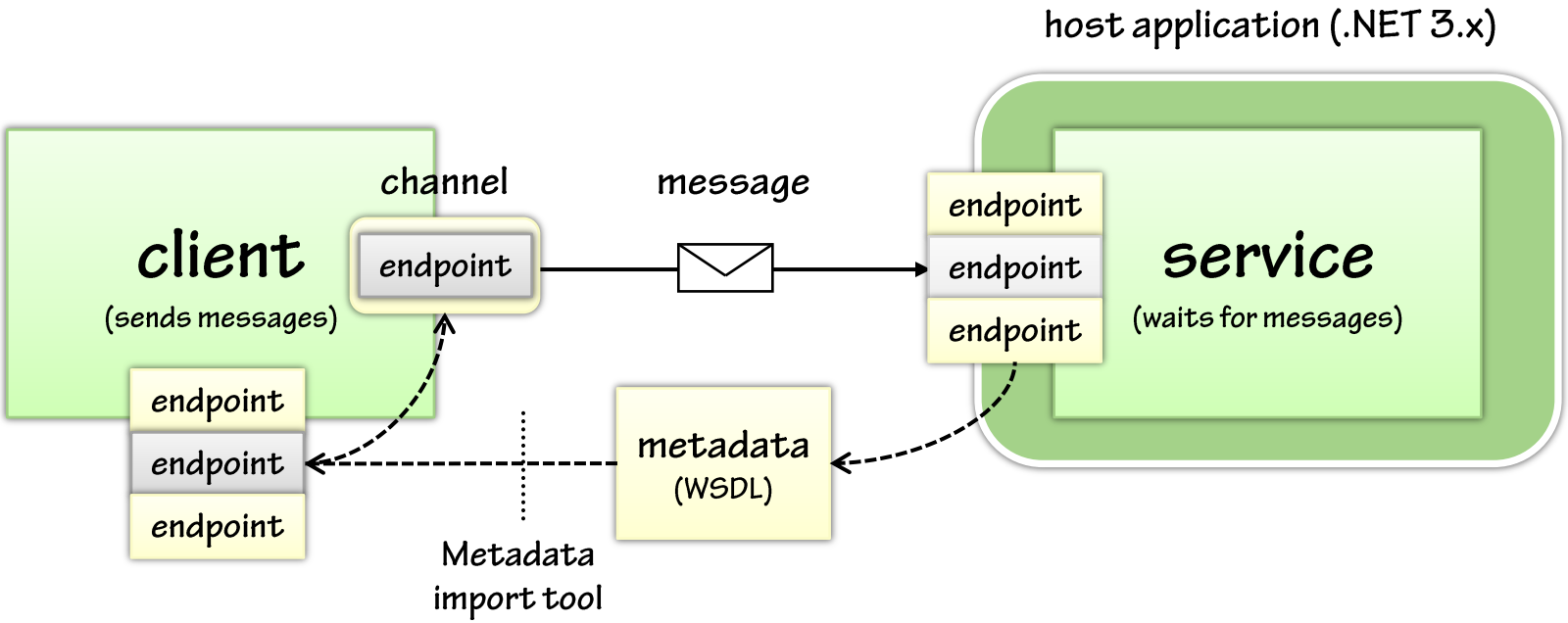
The ABC's of programming WCF services

Outline

- **WCF service model architecture**
- **Programming services**
  - Defining message structures
  - Defining service contracts
  - Implementing services
  - Hosting services
- **Configuring services**
  - Endpoints
  - Behaviors
- **Publishing service metadata**
- **Service exceptions**

The WCF service model architecture

You write services that expose endpoints



Most types are found in the System.ServiceModel namespace

Programming WCF services

- Programming a WCF service consists of the following steps:

1. Define data contracts
2. Define service contracts
3. Implement the service
4. Host the service
5. Configure endpoints & behaviors

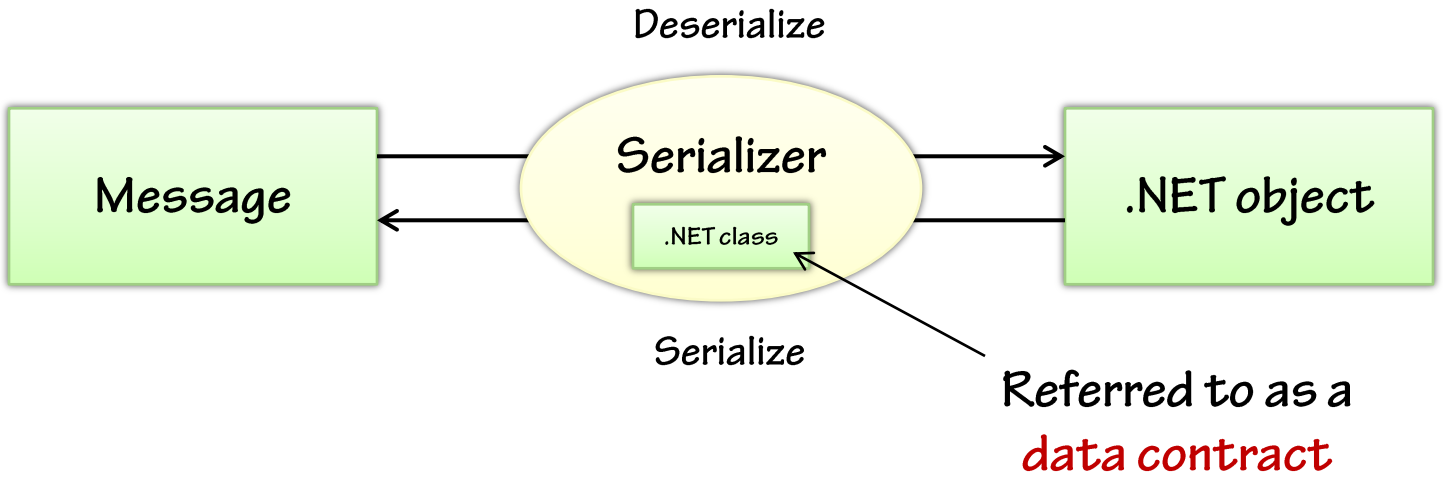
WCF messages

- **WCF models messages with System.ServiceModel.Channels.Message**
  - Message instances can be encoded as XML, MTOM, JSON, binary etc
  - Message instances can be (optionally) mapped to/from .NET objects

```
public abstract class Message : IDisposable {
    // numerous overloads for creating messages
    public static Message CreateMessage(...);
    // reads the body as XML
    public XmlDictionaryReader GetReaderAtBodyContents();
    // deserializes the body into a .NET object
    public T GetBody<T>(XmlObjectSerializer serializer);
    // numerous methods/overloads for writing messages
    public void WriteMessage(XmlDictionaryWriter writer);
    ...
}
```

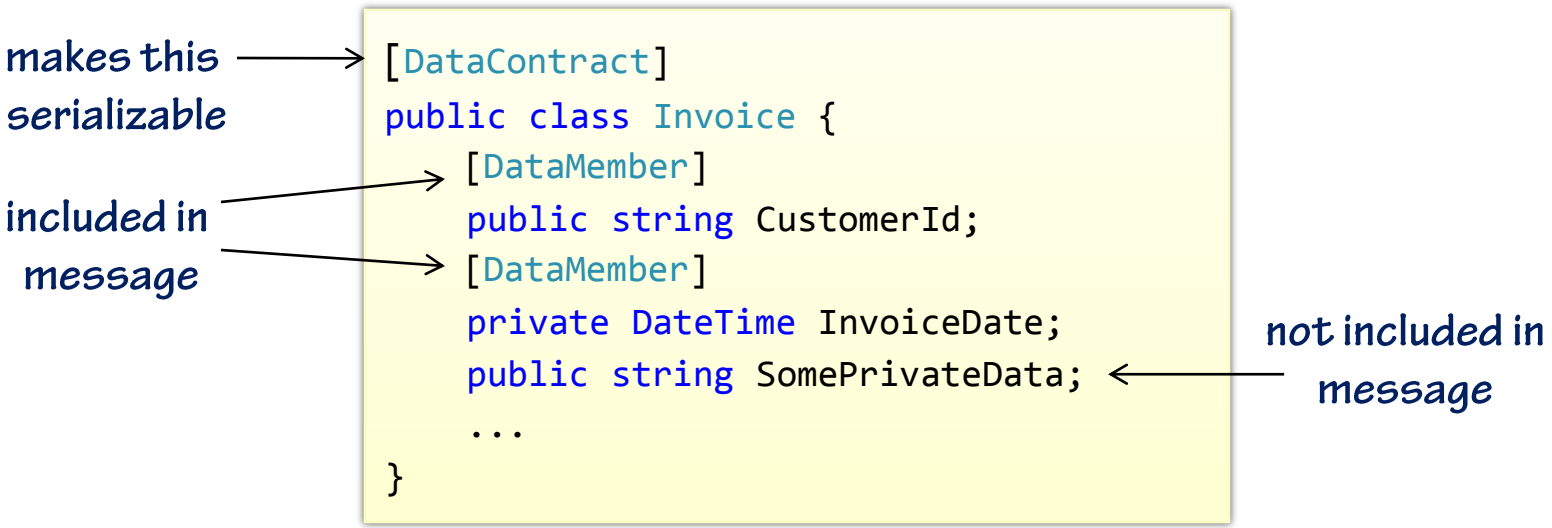
Typed vs. untyped messages

- **Services can be designed in terms of the generic Message type**
  - You can think of these as untyped messages
- **You can "type" messages by defining .NET types that they map to**
  - You annotate these types with special mapping attributes
  - At runtime a serializer maps the .NET objects into messages



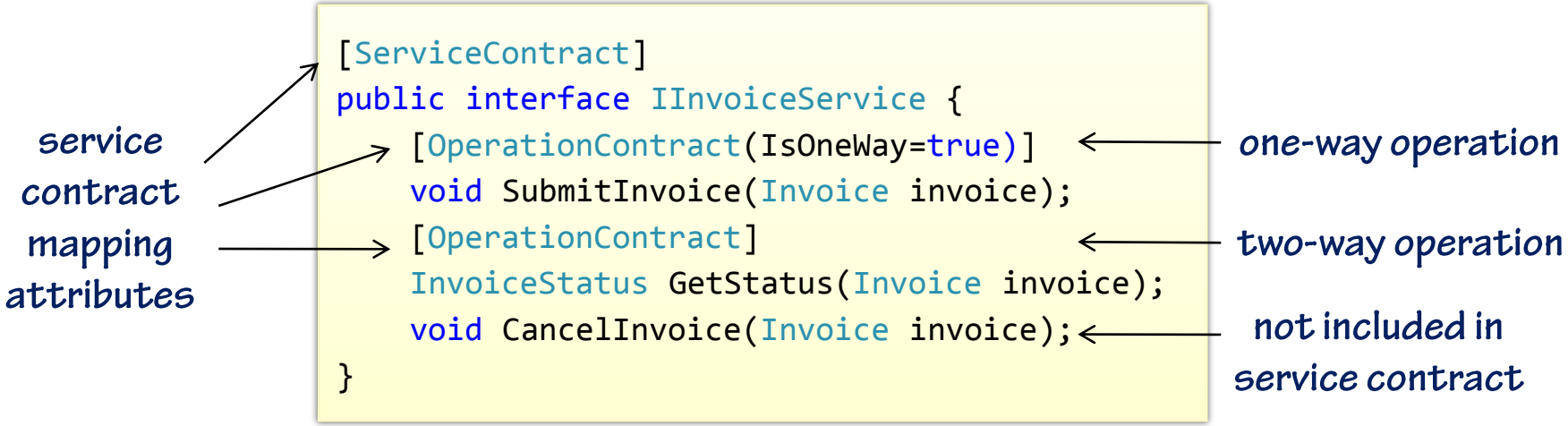
[DataContract] basics

- **DataContractSerializer is the default serializer (but others exist)**
  - Looks for the [DataContract] attributes during serialization
  - Mapping attributes found in System.Runtime.Serialization



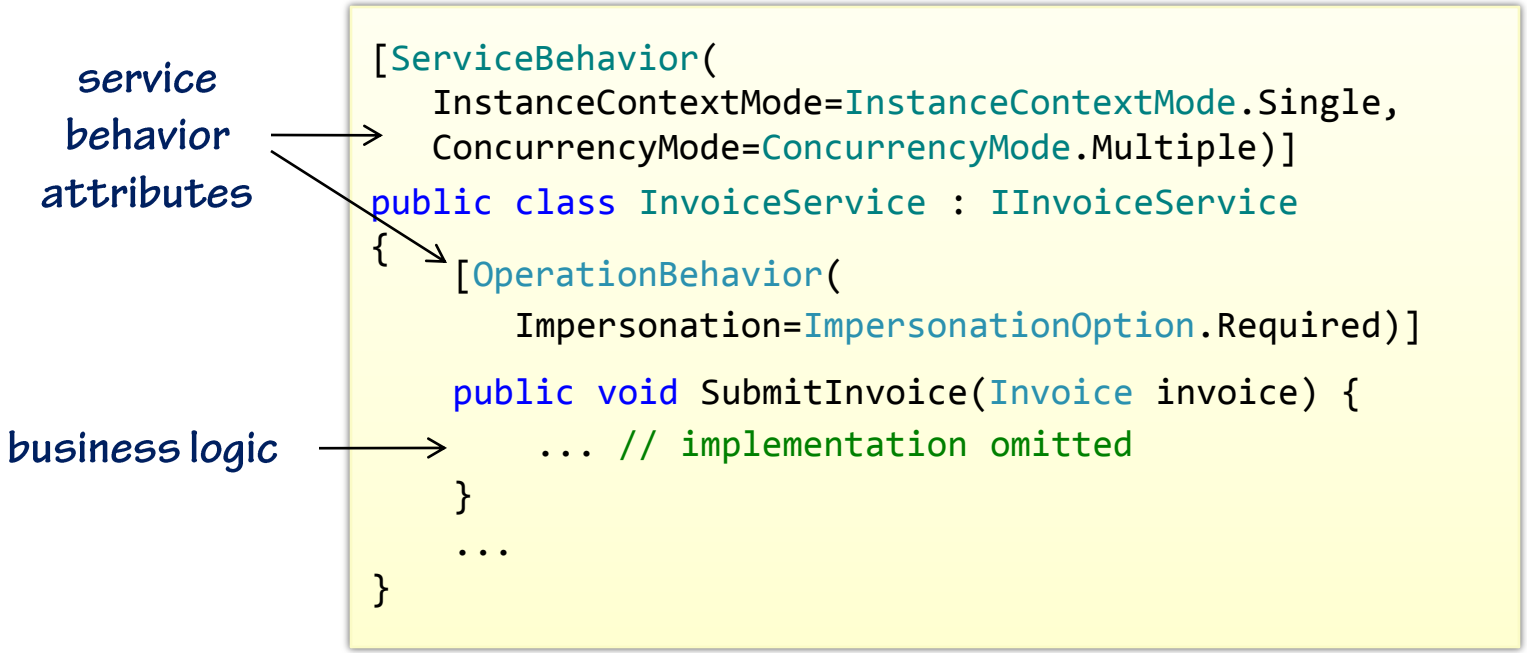
Defining service contracts

- **You model service contracts with .NET interface definitions**
  - Interface defines a grouping of operations
  - Method signatures model the basic message exchange pattern
  - You annotate the interface with the service contract attributes



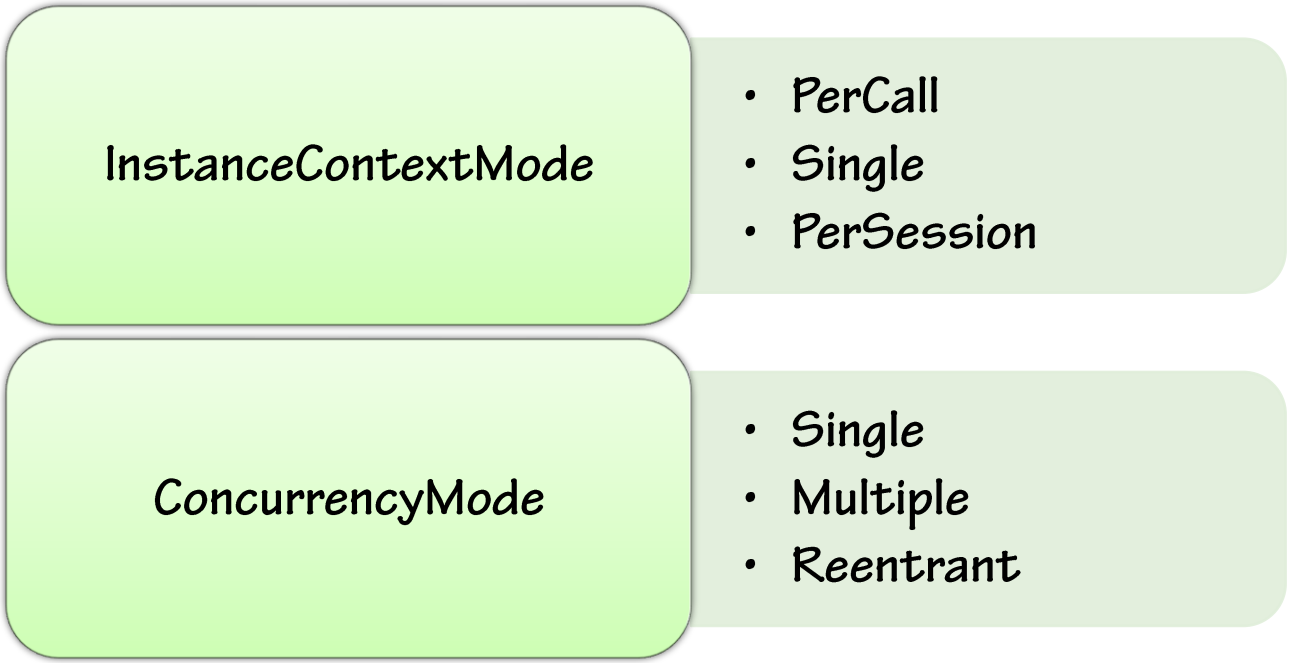
Implementing the service

- **You implement a service by deriving from at least one service contract**
  - You can influence local execution details with behavior attributes



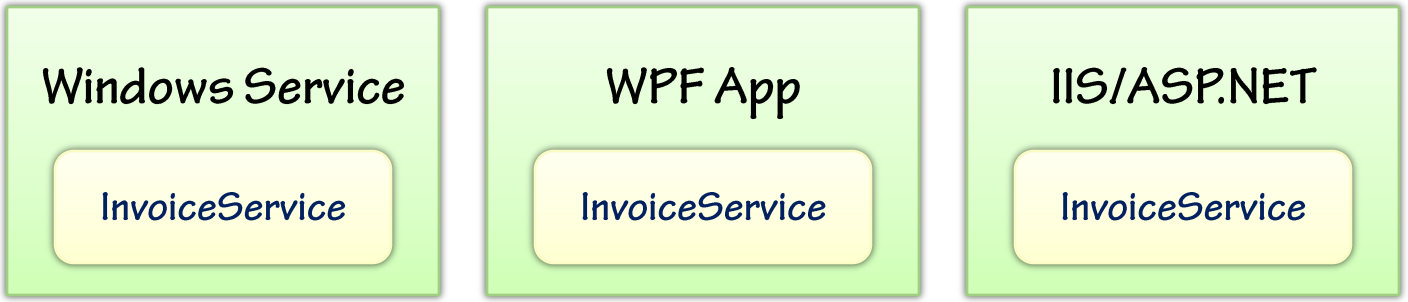
Instancing & threading

- You control service instancing and threading with [ServiceBehavior]



Hosting WCF services

- **WCF services can be hosted in any .NET 3.x application**
  - Simply use the ServiceHost class within your application
  - Called self-hosting because you manage the ServiceHost instance
- **WCF services can also be hosted in IIS/ASP.NET applications**
  - Called managed hosting because you don't touch ServiceHost



Hosting WCF services

Hosting a service using ServiceHost

```
class Program {
    static void Main(string[] args) {
        ServiceHost host =
            new ServiceHost(typeof(InvoiceService));
        ... // configure the host before opening

        try {
            host.Open();
            Console.ReadLine();
            host.Close();
        }
        catch (Exception e) {
            Console.WriteLine(e);
            host.Abort();
        }
    }
}
```

you manage the lifetime of ServiceHost

Hosting in IIS

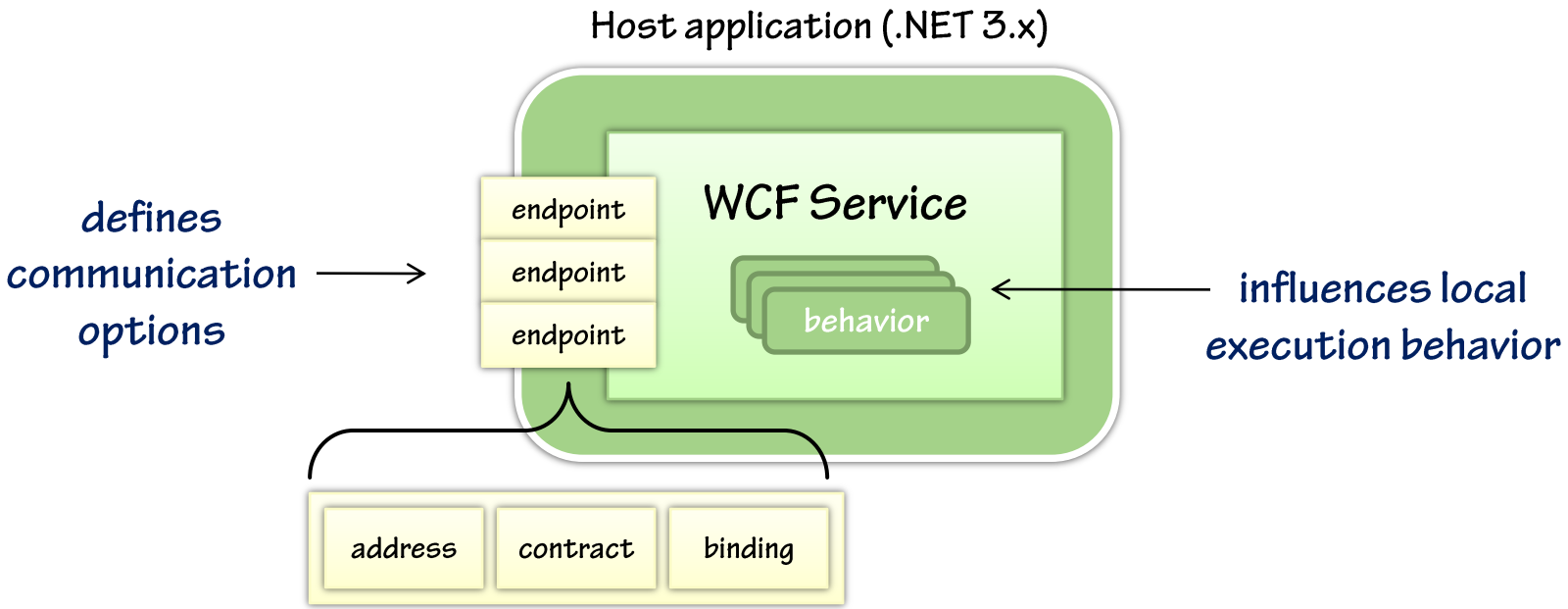
invoice.svc

```
<%@ ServiceHost Service="InvoiceService" %>
```

ASP.NET manages ServiceHost for you!

Configuring a service within a host

- **Before you open the ServiceHost, you must configure the service**
  - You can configure the service with endpoints and behaviors



Code vs. configuration

- **WCF lets you configure services in either code or configuration**
  - Anything you can do in code, you can do in config, and vice-versa
  - Use SvcConfigEditor.exe to work with the WCF config section

WCF configuration section

```
<configuration>
  <system.serviceModel>
    <!-- this is where you configure your WCF applications -->
  </system.serviceModel>
</configuration>
```

Defining service endpoints (code)

- **Call AddServiceEndpoint to expose endpoints on the hosted service**
  - Specify address, binding, and contract for each endpoint

specify address, binding, contract →

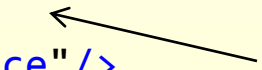
binding class names are PascalCase in code →

```
...
host.AddServiceEndpoint(
    typeof(IInvoiceService),
    new BasicHttpBinding(),
    "http://server/invoiceservice");
host.AddServiceEndpoint(
    typeof(IInvoiceService),
    new NetTcpBinding(),
    "net.tcp://server:8081/invoiceservice");

host.Open();
...
```

Defining service endpoints (config)

```
<configuration>
  <system.serviceModel>
    <services>
      <service name="InvoiceService">
        <endpoint address="http://server/invoiceservice"
                  binding="basicHttpBinding"
                  contract="IInvoiceService"/>
        <endpoint address="net.tcp://server:8081/invoiceservice"
                  binding="netTcpBinding"
                  contract="IInvoiceService"/>
      </service>
    </services>
  </system.serviceModel>
</configuration>
```



binding names are camelCase in config

WCF addresses

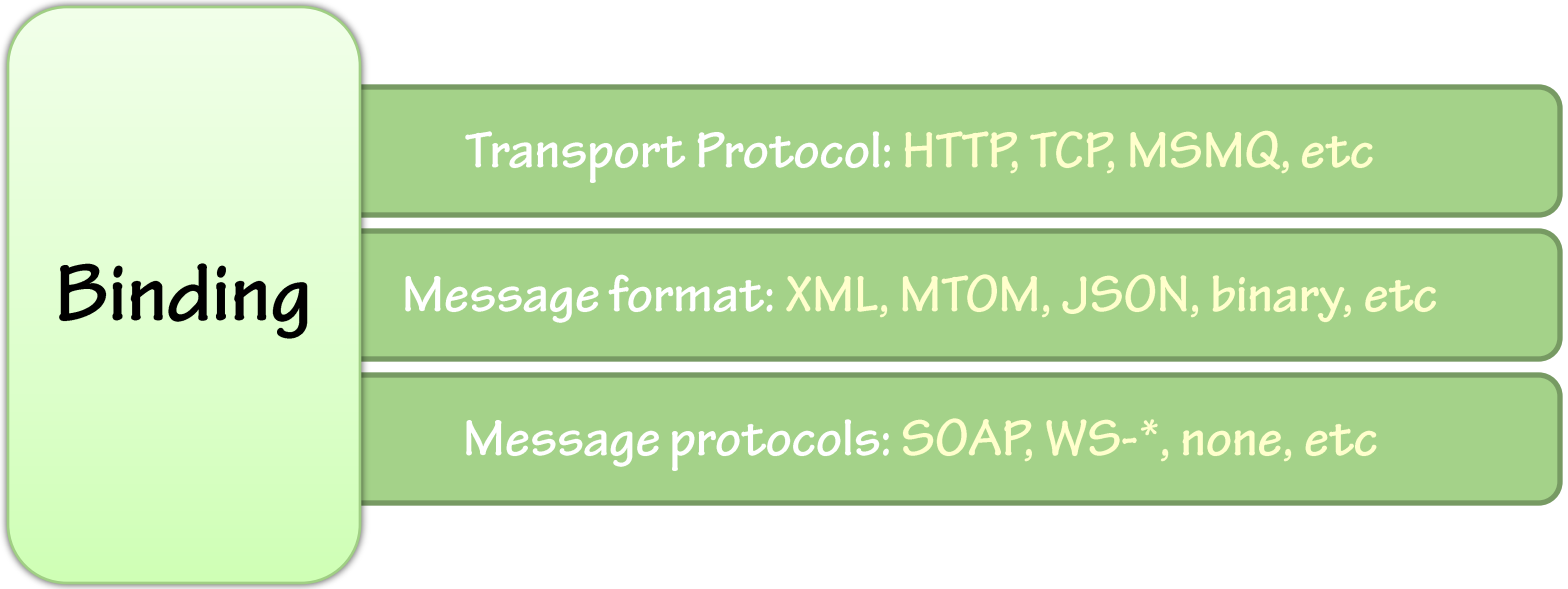
- **Each endpoint is usually configured with a unique address**
  - Endpoints can share an address if they share the same binding instance
  - But they must be configured with different contracts
- **Each endpoint specifies an address (URI) for the runtime to listen on**
  - The exact format of the URI is dependent on the transport

Transport	Scheme	Example
HTTP	http://	http://pluralsight.com/myservice
TCP	net.tcp://	net.tcp://pluralsight.com:8081/myservice
Pipes	net.pipe://	net.pipe://pluralsight.com/myservice
MSMQ	net.msmq://	net.msmq://pluralsight.com/private/myservice

Relative addresses are possible when host is configured with base address

WCF bindings

- A binding is a recipe that specifies three key communication details



WCF provides a suite of built-in bindings

WCF built-in bindings

- The following table describes some of the common built-in bindings

Binding Name	Description
WebHttpBinding	Provides support for REST-style communications
BasicHttpBinding	Provides support for "basic" SOAP communications
WSHttpBinding	Provides support for the full range of SOAP + WS-* protocols
NetTcpBinding	Provides support for SOAP + WS-* over TCP
NetPeerTcpBinding	Provides support for SOAP + WS-* over TCP peer-to-peer
NetNamedPipesBinding	Provides support for SOAP + WS-* over named pipes
NetMsmqBinding	Provides support for SOAP + WS-* over MSMQ
CustomBinding	Allows you to define custom binding configurations

Configuring bindings (code)

- Each built-in binding can be tailored to your needs to some degree

use the built-in BasicHttpBinding →

enable SSL and basic authentication →

use configured binding instance →

```
BasicHttpBinding b = new BasicHttpBinding();
b.Security.Mode = BasicHttpSecurityMode.Transport;
b.Security.Transport.ClientCredentialType =
   HttpClientCredentialType.Basic;

host.AddServiceEndpoint(
    typeof(IInvoiceService), b,
       "http://server/invoiceservice");
...
```

Configuring bindings (config)

apply the binding configuration

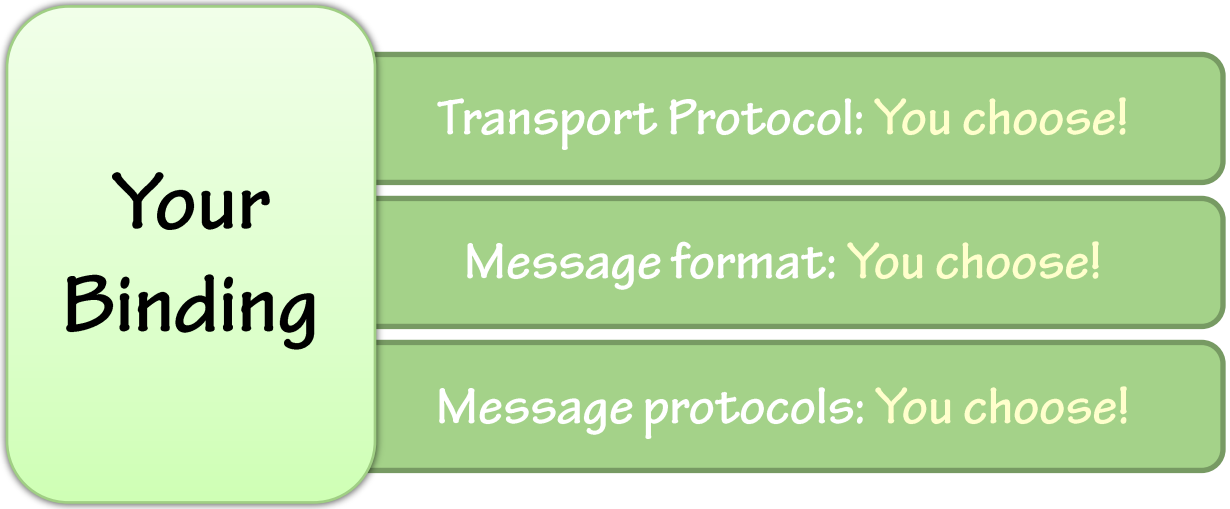
custom configuration for basicHttpBinding

enable SSL and basic authentication

```
<configuration>
  <system.serviceModel>
    <services>
      <service name="InvoiceService">
        <endpoint
          address="https://server/invoiceservice"
          binding="basicHttpBinding"
          bindingConfiguration="MyBindingConfiguration"
          contract="IInvoiceService"/>
        ...
      </service>
    </services>
    <bindings>
      <basicHttpBinding>
        <binding name="MyBindingConfiguration">
          <security mode="Transport">
            <transport clientCredentialType="Basic" />
          </security>
        </binding>
      </basicHttpBinding>
    </bindings>
  </system.serviceModel>
</configuration>
```

Custom bindings & integration

- **The built-in bindings only address common communication scenarios**
  - What about bindings for FTP, SMTP, SAP, or Siebel?
- **WCF allows for custom bindings to meet your communication needs**
  - Use CustomBinding or a custom class (derive from Binding)



Offers unlimited integration potential

Applying service behaviors (code)

- **There are several ways to apply behaviors to your services**
  - Via service attributes, ServiceHost, or configuration

create behavior object and configure properties →

add to Behaviors collection →

```
...
ServiceHost host =
   new ServiceHost(typeof(InvoiceService));

ServiceMetadataBehavior smb =
   new ServiceMetadataBehavior();
smb.HttpGetEnabled = true;

host.Description.Behaviors.Add(smb);

host.Open();
...
```

Applying service behaviors (config)

apply behavior configuration to service

define service behavior configuration

```
<configuration>
  <system.serviceModel>
    <services>
      <service name="InvoiceService"
          behaviorConfiguration="MetadataBehavior">
          ...
      </service>
    </services>
    <behaviors>
      <serviceBehaviors>
        <behavior name="MetadataBehavior">
          <serviceMetadata httpGetEnabled="true" />
        </behavior>
      </serviceBehaviors>
    </behaviors>
  </system.serviceModel>
</configuration>
```

Publishing service metadata

- **WCF can automatically publish service metadata (WSDL) for you**
  - Enable via ServiceMetadataBehavior (see previous slide)
- **WCF also makes it easy to implement WS-MetadataExchange (MEX)**
  - A standard service contract for discovering other endpoints
  - Enable by adding IMetadataExchange endpoints
  - Choose one of the built-in MEX bindings for HTTP, TCP, pipes

define MEX endpoint →

```
<configuration>
  <system.serviceModel>
    <services>
      <service name="InvoiceService"
          behaviorConfiguration="MetadataBehavior">
        <endpoint
          address="net.tcp://server:80801/invoiceservice/mex"
          binding="mexTcpBinding"
          contract="IMetadataExchange"/>
        ...
```

Service exceptions

- **Exceptions are technology specific**
  - Hence, they cannot cross the service boundary
- **WCF automatically translates unhandled exceptions into SOAP faults**
  - For security reasons, only a generic fault message is returned to client

generic SOAP fault message

```
<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/">
  <s:Body>
    <s:Fault>
      <faultcode xmlns:a="..." xmlns="">a:InternalServiceFault</faultcode>
      <faultstring xml:lang="en-US" xmlns="">The server was unable to process the request
      due to an internal error...</faultstring>
    </s:Fault>
  </s:Body>
</s:Envelope>
```

Including unhandled exception details

- **You can instruct WCF to include the exception details in faults**
  - Enable with [ServiceBehavior] or the <serviceDebug> behavior
  - Useful for debugging purposes, not recommended in production

```
<configuration>
    <system.serviceModel>
        <behaviors>
            <behavior name="Default">
               <serviceDebug includeExceptionDetailInFaults="true" />
            </behaviors>
        </behaviors>
    </system.serviceModel>
</configuration>
```

allows exception details to flow

FaultException

- **The FaultException class represents an explicit SOAP fault**
  - Throw in a service operation to return a SOAP fault
  - WCF clients can catch a FaultException to handle the error
  - You can also throw/catch typed faults (more on this later)

```
...
public void DoSomething(string input) {
    ...
    throw new FaultException("Something bad happened");
}
```

fault reason

Summary

- **The service model architecture revolves around endpoint definitions**
- **Programming WCF services consists of the following steps:**
  - Define message structures
  - Define service contracts
  - Implement the service
  - Host the service
- **You can configure services with endpoints & behaviors**
- **WCF makes it easy to publish WSDL and to expose MEX endpoints**
- **WCF provides various options for dealing with exceptions**

References

- **The ABC's of Programming WCF, Skonnard**
  - http://msdn.microsoft.com/msdnmag/issues/06/02/WindowsCommunicationFoundation/default.aspx
- **WCF Essentials, Lowy**
  - http://www.code-magazine.com/Article.aspx?quickid=0605051
- **Introduction to Building WCF Services, Vasters**
  - http://msdn.microsoft.com/webservices/indigo/default.aspx?pull=/library/en-us/dnlong/html/introtowcf.asp
- **Pluralsight's WCF Wiki**
  - http://pluralsight.com/wiki/default.aspx/Aaron/WindowsCommunicationFoundationWiki.html