Service Contracts

Mapping methods to messages

Overview

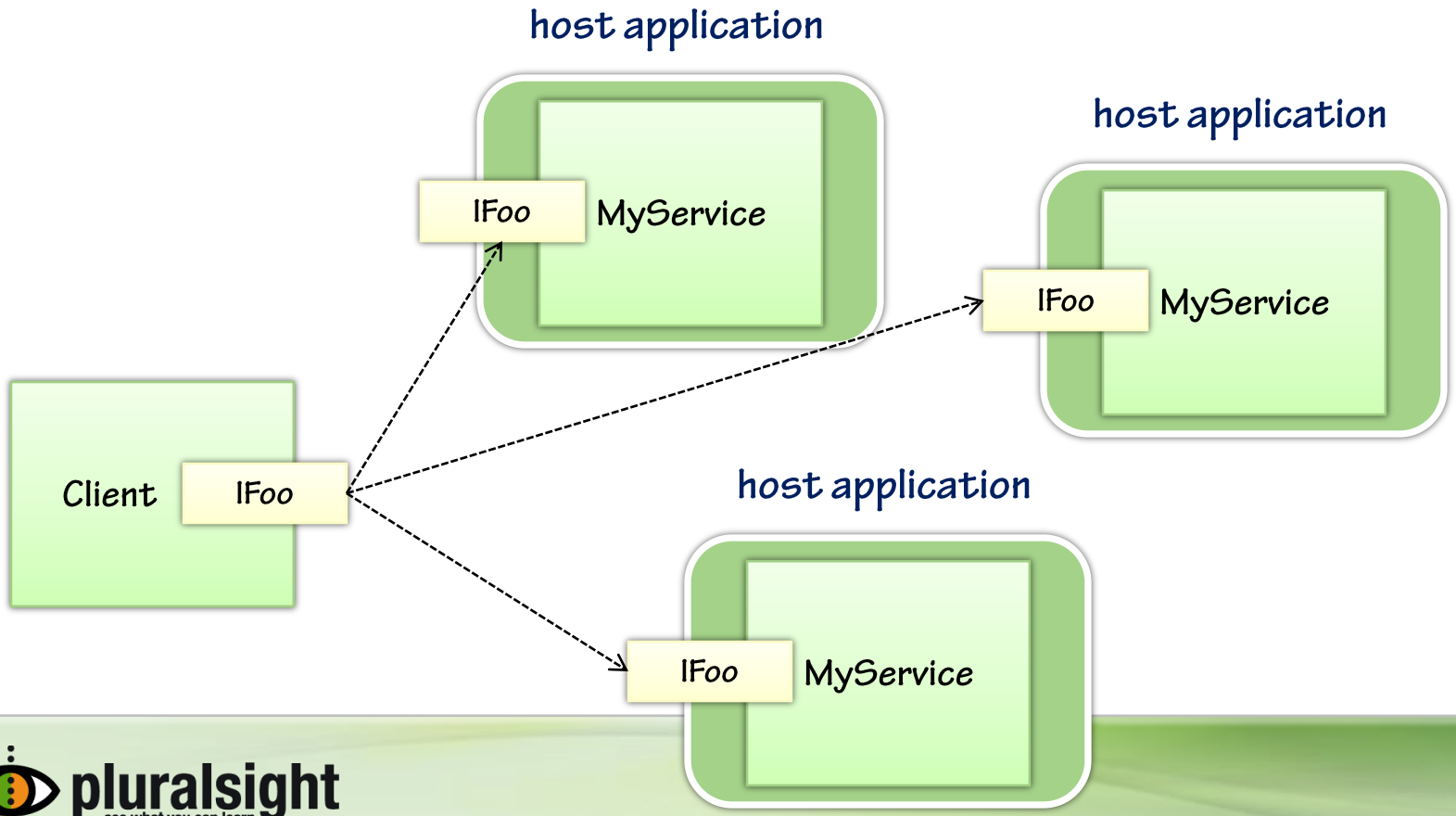
- **Architecture**
  - Understanding service contracts
- **Service contract details**
  - Dispatching
  - Importing/exporting
  - Mapping attributes
- **Designing operations**
  - One-way
  - Duplex
- **Advanced topics**
  - Message contracts
  - Generic contracts

What is a service contract?

- **A service contract is simply a logical group of operations**
  - The operations are usually related in some way
- **Each operation is assigned an action, which is its official name**
  - The action is a Uri and it's carried in each message
  - The runtime dispatches based on the action by default
- **Each operation also defines a message exchange**
  - The parameter list defines the request message
  - The return type defines the (optional) response message

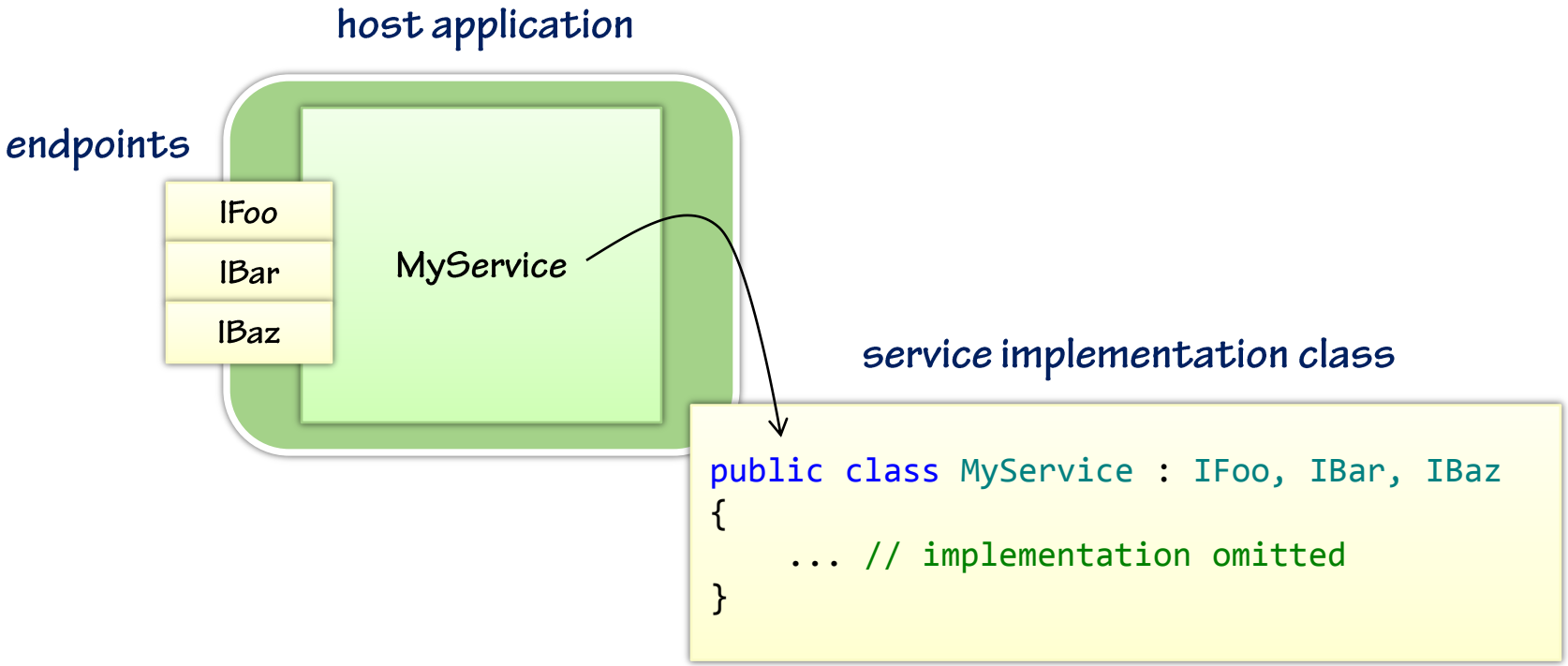
Service contract architecture

- **Multiple services can all implement the same service contract**
  - Allows a single client to talk to all services the same way



Service contract architecture

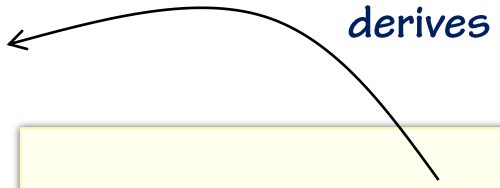
- **A single WCF service can implement more than one service contract**
  - You expose each service contract through a distinct endpoint



Service contract architecture

- **Service contract interfaces can derive from one another**
  - But each interface in the hierarchy must carry [ServiceContract]
  - You can use the bottom-most contract exposes all operations
  - Or you can expose individual contracts via different endpoints

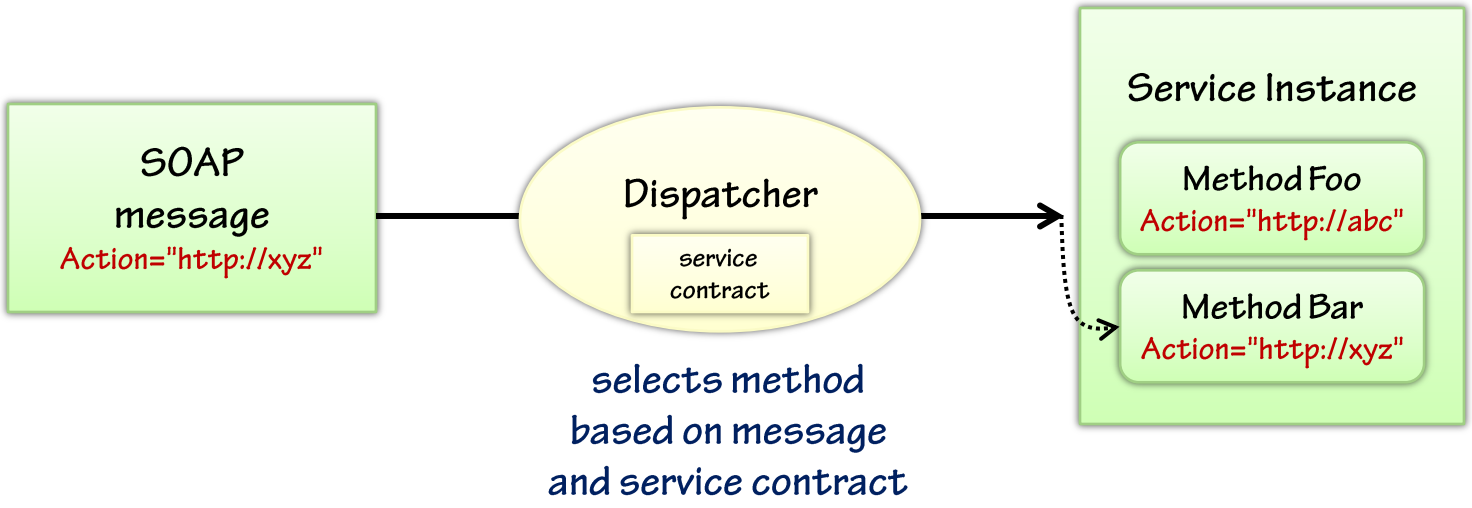
```
[ServiceContract]
public interface IFoo
{
    [OperationContract]
    void One();
    [OperationContract]
    void Two();
}
```




```
[ServiceContract]
public interface IBar : IFoo
{
    [OperationContract]
    void Three();
}
```

Runtime dispatching

- **Service contracts drive the runtime dispatching process**
  - Action is normally used for selecting the method but that's extensible
  - The WebHttpBehavior looks at the HTTP method + the URI
  - The dispatcher uses a serializer to map messages into objects

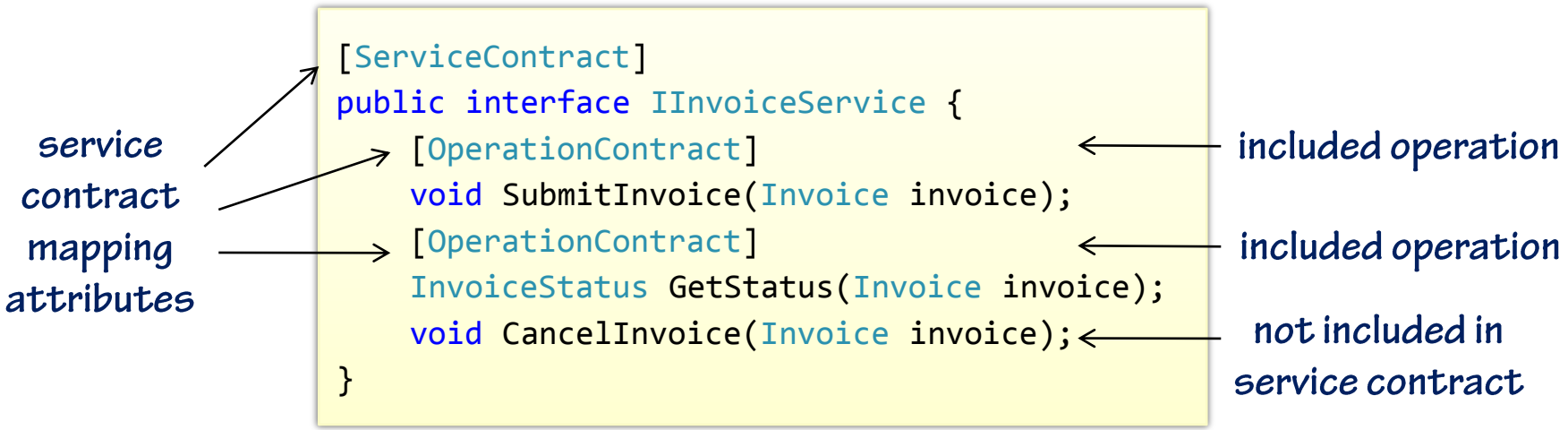


Importing/exporting service contracts

- **WCF provides SvcUtil.exe for moving between .NET types and WSDL**
  - You can export a WSDL definition that describes a WCF service contract
  - You can import a WSDL definition to generate a WCF service contract
  - Metadata behavior provides automatic metadata generation (?wsdl)

[ServiceContract] basics

- **You define service contracts in .NET with an interface or a class**
  - Annotate the type with [ServiceContract]
  - Annotate each method you wish to include with [OperationContract]



Default mapping

- **Default mapping for [ServiceContract]**
  - Default target namespace is http://tempuri.org/
  - Type name becomes the service contract name
  - DataContractSerializer is assumed by default
- **Default mapping for [OperationContract]**
  - Method name becomes the operation name
  - Action = target namespace + service contract name + operation name
  - All methods are request-reply by default

Basic customization

[ServiceContract]

- Name
- Namespace

[OperationContract]

- Name
- Action
- ReplyAction
- IsOneWay

Service contract customization

change contract name & namespace

```
[ServiceContract(Name="InvoiceContract",
    Namespace="http://pluralsight/invoice")]
public interface IInvoiceService {

    [OperationContract(IsOneWay=true)]
    void SubmitInvoice(Invoice invoice);

    [OperationContract(Name="SubmitWithUserId",
        Action="urn:submit-with-userid")]
    void SubmitInvoice(Invoice invoice, string userId);
    ...
}
```

make one-way

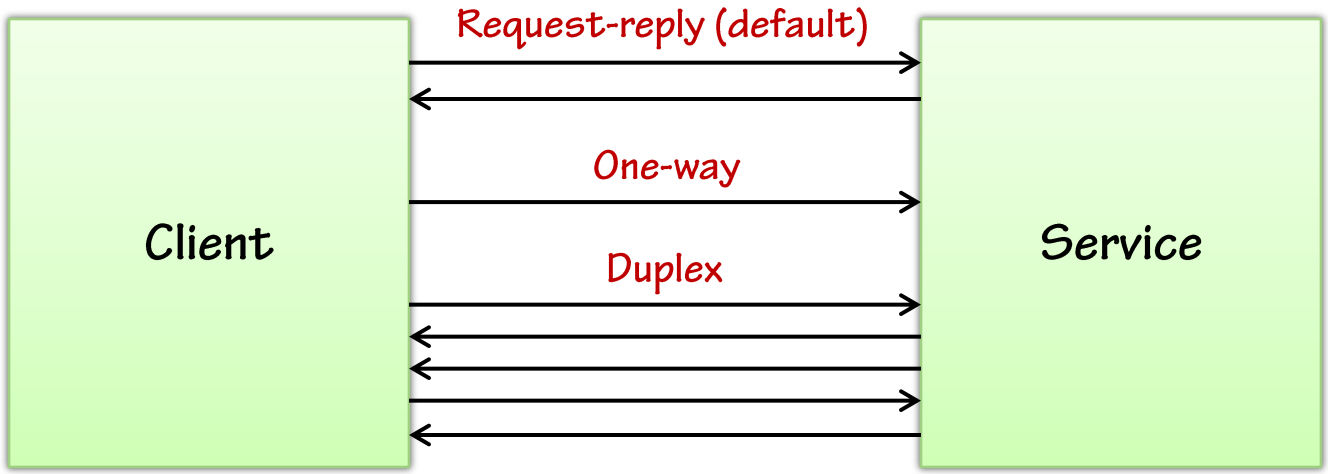
change name, set explicit action

how you deal with method overloading

Action for SubmitInvoice: http://pluralsight/invoice/IInvoiceContract/SubmitInvoice

Designing operations

- WCF supports three different types of message exchange patterns



One-way operations

- **You make operations one-way using IsOneWay=true**
  - Method return type must be void
  - Service cannot return faults to client
- **One-way operations are dispatched differently than request-reply**
  - Dispatcher returns control to client as soon as message is queued
  - Client has no way direct way to know if call succeeded
- **MSMQ is an inherently one-way transport**
  - Hence, when using it, all operations must be one-way

Duplex contracts

- **A duplex contract is a relationship between two service contracts**
  - The service implements an incoming contract
  - The client implements a callback contract
  - You associate the callback contract with the incoming contract

Associate with incoming contract →

Callback contract →

```
[ServiceContract(CallbackContract=
    typeof(IQuoteNotification))]
public interface IRequestStockQuotes {
    [OperationContract]
    void RequestNotifications(string symbol);
}

[ServiceContract]
public interface IQuoteNotification {
    [OperationContract]
    void SendQuote(StockQuote quote);
}
```

Duplex-capable bindings

- **Duplex contracts require a bi-directional communication channel**
  - NetTcpBinding & NetNamedPipeBinding are bi-directional by nature
  - The standard HTTP bindings are not bi-directional
- **WCF provides a special HTTP binding for duplex contracts**
  - WSDualHttpBinding creates two underlying HTTP channels
  - One for receiving messages and another for sending messages
- **WCF provides a special channel factory for creating duplex channels**
  - Use DuplexChannelFactory<T> for creating duplex channels
  - SvcUtil.exe creates proxy classes that derive from DuplexClientBase

Calling the callback channel

- **You can retrieve a proxy to the callback channel via OperationContext**
  - Client endpoint details provided in the request message

```
public class RequestStockQuotesService : IRequestStockQuotes
{
    public void RequestNotifications(string symbol)
    {
        StockQuote quote = GetRealtimeQuote(symbol);
        IQuoteNotification callback =
            OperationContext.Current.GetCallbackChannel<
            IQuoteNotification>();
        callback.SendQuote(quote);
    }
}
```

Retrieves proxy to callback channel

Using a duplex proxy on the client

Client-side service implementation

```
public class QuoteNotificationService : IQuoteNotification {
    public void SendQuote(StockQuote quote) {
        Console.WriteLine("Quote: {0} is at {1}",
            quote.Symbol, quote.Last);
    }
}
```

supply the client-side InstanceContext

duplex-aware proxy

```
class Program {
    static void Main(string[] args) {
        InstanceContext ctx = new InstanceContext(
            new QuoteNotificationService());
        RequestStockQuotesServiceProxy proxy =
            new RequestStockQuotesServiceProxy(ctx);
        proxy.RequestQuote("MSFT");
        ...
    }
}
```

this call causes a callback

Mapping methods to messages

- **The default mapping for an [OperationContract] to messages**
  - A wrapper element is expected/used for request/response
  - Wrapper elements named after operation name
- **Request wrapper element named the same as operation name**
  - Parameters are serialized within the wrapper element
- **Return type serialized within a response wrapper element**
  - Named after operation name + "Response"
  - Return object serialized as operation name + "Result"

Mapping methods to messages

```
[DataContract]
public class MyType {
    [DataMember]
    public string Something;
}
[ServiceContract]
public interface ISampleService {
    [OperationContract]
    MyType DoSomething(MyType input);
}
```

request message

```
<s:Envelope xmlns:s="...">
  <s:Body>
    <DoSomething xmlns="http://tempuri.org/">
      <input xmlns:b="...">
        <b:Something>something</b:Something>
      </input>
    </DoSomething>
  </s:Body>
</s:Envelope>
```

response message

```
<s:Envelope xmlns:s="...">
  <s:Body>
    <DoSomethingResponse xmlns="http://tempuri.org/">
      <DoSomethingResult xmlns:b="...">
        <b:Something>something</b:Something>
      </DoSomethingResult>
    </DoSomething>
  </s:Body>
</s:Envelope>
```

[MessageContract]

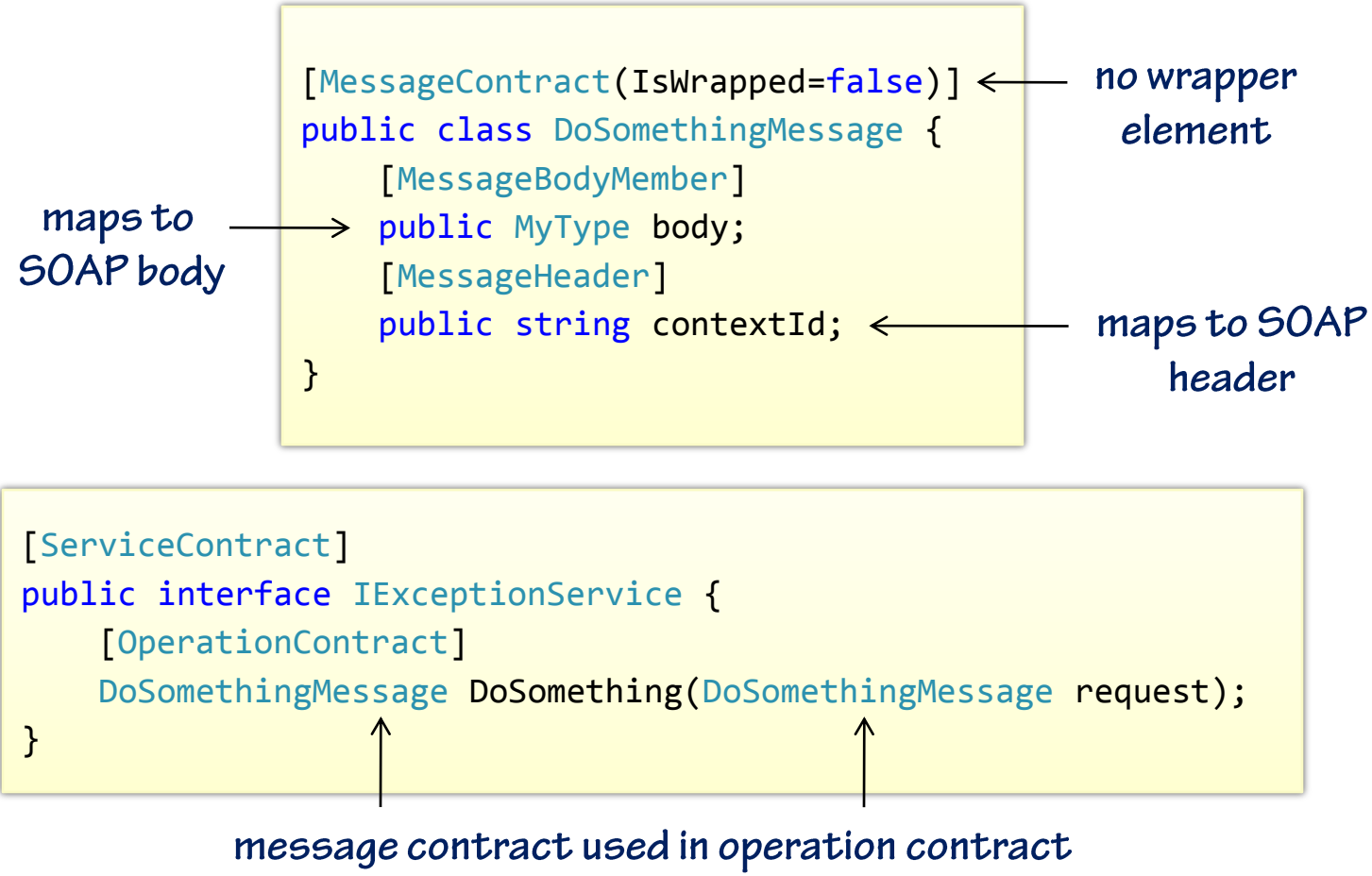
- **You can customize this mapping using message contract types**
  - Message contracts map data contracts to SOAP envelopes
  - Annotate the class with [MessageContract]
  - Map members to the body using [MessageBodyMember]
  - Map members to headers using [MessageHeader]
- **Then use the message contract types in the method signature**
  - The parameter list should contain a single [MessageContract] type
  - If the request is a message contract, the response must be also

Defining message contracts

- **The message contract attributes allow you to control the mapping**
  - You can control wrapper behavior, order, header processing, etc.

[MessageContract]	[MessageBodyMember]	[MessageHeader]
• IsWrapped • WrapperName • WrapperNamespace	• Name • Namespace • Order	• Name • Namespace • MustUnderstand • Actor • Relay

[MessageContract] example



The universal operation

- **You can also choose to process the raw WCF Message object**
  - Setting Action="*" defines a catch-all operation
  - Dispatcher calls this method when it doesn't find a matching action
  - Only one method per contract can be annotated this way

```
[ServiceContract]
public interface IUniversalOneWay {
    [OperationContract(Action="*", IsOneWay=true)]
    void ProcessMessage(Message msg);
}
```

Notice the method receives a Message (any message)

Same thing but for a request-reply operation

```
[ServiceContract]
public interface IUniversalTwoWay {
    [OperationContract(Action="*", ReplyAction="*")]
    Message ProcessMessage(Message msg);
}
```

Summary

- A service contract is a logical group of operations
- Service contracts influence dispatching & message processing
- Message contracts allow you to control message details
- SvcUtil.exe can map between service contracts and WSDL
- Service contracts can have one-way and request-reply operations
- WCF also supports duplex contracts (bidirectional)

References

- **The ABC's of Programming WCF**
  - http://msdn.microsoft.com/msdnmag/issues/06/02/WindowsCommunicationFoundation/default.aspx
- **Specifying and Handling Faults in Contracts and Services (MSDN)**
  - http://msdn2.microsoft.com/en-us/library/ms733721.aspx
- **Pluralsight's WCF Wiki**
  - http://pluralsight.com/wiki/default.aspx/Aaron/WindowsCommunicationFoundationWiki.html