Module 1
Overview of Service and
Cloud Technologies
Developing Windows Azure
and Web Services

Updated 11th April 2014

Developing Windows Azure and Web Services
Course and Exam Contents

Microsoft Official Exam Guide

- Deploying Web Applications and Services 19%
- Accessing Data 24%
- Creating and Consuming Web API-based services 18%
- Querying and Manipulating Data by Using the Entity Framework 20%
- Designing and Implementing WCF Services 19%

Jun 2013 to Jan 2014
130 minutes total
42 questions in total
22 in main section
2 case studies
(9, 11 questions)

Warnings!
Some of the code in questions miss spaces, commas, or semicolons. Try not to let that put you off:

```
var doc = new XDocument();
```

50% code / 50% configuration
33% data, 33% WCF, 33% other

Microsoft Exam 70-487 Study Guide
1.3
Estimate of Number of Exam Questions per Module

<table>
<thead>
<tr>
<th>Module</th>
<th>Qs</th>
</tr>
</thead>
<tbody>
<tr>
<td>1: Overview of Service and Cloud Technologies</td>
<td>1</td>
</tr>
<tr>
<td>2: Querying and Manipulating Data Using Entity Framework</td>
<td>4</td>
</tr>
<tr>
<td>3: Creating and Consuming ASP.NET Web API Services</td>
<td>3</td>
</tr>
<tr>
<td>4: Extending and Securing ASP.NET Web API Services</td>
<td>3</td>
</tr>
<tr>
<td>5: Creating WCF Services</td>
<td>7</td>
</tr>
<tr>
<td>6: Hosting Services</td>
<td>3</td>
</tr>
<tr>
<td>7: Windows Azure Service Bus</td>
<td>1</td>
</tr>
<tr>
<td>8: Deploying Services</td>
<td>4</td>
</tr>
<tr>
<td>9: Windows Azure Storage</td>
<td>2</td>
</tr>
<tr>
<td>10: Monitoring and Diagnostics</td>
<td>1</td>
</tr>
<tr>
<td>12: Scaling Services</td>
<td>1</td>
</tr>
<tr>
<td>A: Designing and Extending WCF Services</td>
<td>4</td>
</tr>
<tr>
<td>B: Implementing Security in WCF Services</td>
<td>2</td>
</tr>
<tr>
<td>C: “Classic” XML and ADO.NET</td>
<td>4</td>
</tr>
<tr>
<td>D: LINQ</td>
<td>2</td>
</tr>
<tr>
<td>Total questions in exam</td>
<td>42</td>
</tr>
</tbody>
</table>

1.4
Data APIs in .NET (2002-2007)

“Classic” ADO.NET

- .NET Framework 1.0, 1.1, and 2.0

- .NET Data Provider
  (SqlClient, OracleClient, etc.)

- SQL Server
- Oracle
- Etc.

- Custom Classes and Collections

- DataAdapter

- DataSet (change tracking, filtering, sorting, XML/XSD, strongly-typed)

* .NET 2.0 adds minor improvements like TableAdapters
Overview
Object-Relational Mapping

What are ORMs?
- Objects are more natural to work with for programmers...
- ...but relational data is better for storage
- Mapping converts CRUD on objects to CRUD on relational data

Philosophy of ORM
- If you do most work through stored procedures (SELECT, etc.)
  you will gain very little from using an ORM so use “Classic”
  ADO.NET instead

The objects should be “persistence ignorant”
- Members are mostly properties to store column values
- Can have methods for validation and business logic
- Should NOT have methods to store data

Overview
Data APIs in .NET (2008-2011)

LINQ and Entity Framework
- .NET Framework 3.5 SP1 and .NET 4
1.7 Overview

Entity Framework vs. LINQ to SQL

**LINQ to SQL, .NET 3.5**
- Created by C# team
- Simple ORM; one-to-one object-to-table mapping (although it does support a discriminator column for simple inheritance scenarios)
- SQL Server only
- Will be supported, but not improved

**Entity Framework, .NET 3.5 SP1**
- Created by SQL Server team
- Complex, powerful, flexible ORM
- Heterogeneous data sources
- Future of Microsoft Data APIs

1.8 Overview

Data APIs in .NET (2012+)

**.NET Framework 4.5**
- Appendix C: “Classic” ADO.NET and XML
- Appendix D: LINQ (including common providers like LINQ to XML)
- Appendix E: Entity Framework 4 (Visual Studio 2010)
- Module 2: Entity Framework 5 (Visual Studio 2012)
Overview

Entity Framework Assemblies

- Entity Framework is implemented in two assemblies
  - System.Data.Entity.dll is EF 4 (based on ObjectContext)
  - EntityFramework.dll is EF 4.1+ (based on DbContext)

- Some Visual Studio templates do not include the new EF 4.1+ assembly
  - e.g. WPF Applications, Windows Forms Applications, Console

- You must add the NuGet package for EntityFramework

Overview

Data Access APIs: Why Use...

- “Classic” ADO.NET
  - 1) legacy code, 2) performance, 3) mostly use stored procedures

- ADO.NET Entity Framework
  - Database-First or Model-First: separate conceptual model from storage model with complex mappings defined in .EDMX
  - Code-First with DbContext: for simple one-to-one mapping models and runtime generation of model and database

- WCF Data Services or ASP.NET Web API OData
  - Expose data via OData (queryable REST-style service)

- Windows Azure Storage (scalable, REST-style)

Exam Topic: Choose data access technologies
- Choose a technology (ADO.NET, Entity Framework, WCF Data Services, Azure storage) based on application requirements
Overview
Which Databases Work with Entity Framework?

**Supported .NET Data Providers**
- MySQL, Oracle, Progress, VistaDB, SQL Lite, PostgreSQL, Virtuoso, IBM DB2, Informix, U2, Sybase, Sybase SQL Anywhere, Synergy, Firebird, Npgsql, and many others

[Images of database logos]

**Third-Party Providers**
[Link to MSDN page](http://msdn.microsoft.com/en-us/data/dd363565.aspx)

Further Study
Entity Framework and LINQ

**To go deeper with Entity Framework**
- Programming Entity Framework, 2nd Edition
- Experienced author, Julie Lerman
- [http://thedatafarm.com/blog/](http://thedatafarm.com/blog/)

**To go further into EF 4.1+**

**LINQ to Objects using C# 4.0**
- Using and Extending LINQ to Objects and Parallel LINQ
Further Study
Windows Communication Foundation (WCF)

To go deeper with WCF
- Programming WCF Services, 3rd Edition
- Experienced author, Juval Löwy

Why this book?
- Focuses on the possibilities and practical aspects of using WCF 4 so it complements the MOC
- Contains many useful utilities, tools, and helper classes such as ServiceModelEx

“Every Object Should be a WCF Service”

Programming WCF Services, 3rd Edition
http://shop.oreilly.com/product/9780596805494.do

Further Study
ASP.NET Web API

To go deeper with Web API
- Designing Evolvable Web APIs with ASP.NET
- Experienced authors
- Due September 2013 but available now as an Early Release

To go deeper with general concept of REST/HTTP style services
- RESTful Web Services Cookbook

RESTful Web Services Cookbook
http://www.amazon.co.uk/RESTful-Web-Services-Cookbook-ebook/dp/B004362EIq

Designing Evolvable Web APIs with ASP.NET
http://shop.oreilly.com/product/0636920026617.do
Further Study
Windows Azure

Windows Azure is changing so fast that printed books are not the best choice

* Use the official online documentation and blogs

* You can also download the 211 page PDF about “Building Real-World Cloud Apps with Windows Azure” using the link below

Windows Azure

Building Real-World Cloud Apps with Windows Azure
Module 2
Querying and Manipulating Data Using Entity Framework
Developing Windows Azure and Web Services

Updated 11th April 2014

Entity Framework Contents

<table>
<thead>
<tr>
<th>Topic</th>
<th>Slide</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADO.NET “Classic”</td>
<td>3</td>
</tr>
<tr>
<td>Connection Strings</td>
<td>6</td>
</tr>
<tr>
<td>Entity Data Models</td>
<td>11</td>
</tr>
<tr>
<td>Stored Procs and Functions</td>
<td>17</td>
</tr>
<tr>
<td>Code First (and POCO)</td>
<td>22</td>
</tr>
<tr>
<td>Inheritance Hierarchies</td>
<td>30</td>
</tr>
<tr>
<td>Designer Tools</td>
<td>33</td>
</tr>
<tr>
<td>Loading Patterns</td>
<td>38</td>
</tr>
<tr>
<td>Entity SQL</td>
<td>39</td>
</tr>
<tr>
<td>EntityClient</td>
<td>46</td>
</tr>
<tr>
<td>Querying Summary</td>
<td>48</td>
</tr>
<tr>
<td>Performance</td>
<td>49</td>
</tr>
<tr>
<td>Transactions</td>
<td>50</td>
</tr>
</tbody>
</table>

Exam Topic: Create an Entity Framework data model
- Structure the data model using Table per type, table per class, table per hierarchy
- Choose and implement an approach to manage a data model (code first vs. model first vs. database first)
- Implement POCO objects
- Describe a data model by using conceptual schema definitions, storage schema definition, and mapping language (CSDL, SSDL, MSL)

Exam Topic: Query data by using Data Provider for EF
- Query data by using Connection, DataReader, Command from the System.Data.EntityClient namespace
- Perform synchronous and asynchronous operations

Exam Topic: Query and manipulate data by using EF
- Query data by using DbContext
- Build a query that uses deferred execution
- Implement lazy loading and eager loading
- Query data by using Entity SQL
- Update, and delete data by using DbContext
- Create and run compiled queries

Exam Topic: Implement transactions
- Manage transactions by using the API from System.Transactions
- Implement distributed transactions
- Specify transaction isolation level
ADO.NET “Classic”
Connections, Commands, DataReaders

Must have connection open

```csharp
using System.Data;
using System.Data.SqlClient;

var con = new SqlConnection(conStr);
var cmd = new SqlCommand(sql, con);
con.Open(); // open connection before executing commands
```

Common CommandBehaviors

- CloseConnection, SequentialAccess, SingleResult, SingleRow

```csharp
var reader = cmd.ExecuteReader(CommandBehavior.SingleResult);
while (reader.Read()) // returns true if another row exists
{
    // process row
}
reader.Close(); // close reader before reading parameters
int outputParam = (int)cmd.Parameters[2].Value;
con.Close(); // or use CommandBehavior.CloseConnection
```

ADO.NET “Classic”
How to Enable Asynchronous Operations

Asynchronous access can improve performance and responsiveness of an application (SQL Server only)

To enable asynchronous execution of commands you must enable the feature in connection string

```csharp
var conn = new SqlConnection("...;Asynchronous Processing=True;"最常见的模板，也叫"经典模板"，是专注于SQL Server的。当需要提高性能和应用的响应性时，可以启用异步操作。要启用命令执行的异步执行，必须在连接字符串中启用此特性。
```

```csharp
using System.Data;
using System.Data.SqlClient;

var conn = new SqlConnection("...;Asynchronous Processing=True;"最常见的模板，也叫"经典模板"，是专注于SQL Server的。当需要提高性能和应用的响应性时，可以启用异步操作。要启用命令执行的异步执行，必须在连接字符串中启用此特性。
```
ADO.NET “Classic”
How to Execute Asynchronous Operations

ADO.NET 4 and earlier

```csharp
IAsyncResult task = cmd.BeginExecuteNonQuery();
// do other work; can check task.IsCompleted
int i = cmd.EndExecuteNonQuery(task);
```

ADO.NET 4.5

```csharp
Task<int> t = cmd.ExecuteNonQueryAsync();
// do other work
int i = await t;
```

Connection Strings
EDM Embedded in Assembly

By default the three XML files that define an EDM are embedded in the output assembly as resources

```
<add name="AdventureWorksEntities" providerName="System.Data.EntityClient"
provider=System.Data.SqlClient;
provider connection string="data source=.\sqlexpress;
initial catalog=AdventureWorks;
integrated security=True;
multipleactiveresultsets=True;
App=EntityFramework"/>
```

Entity Framework
2.7 Connection Strings
EDM Stored As Loose Files

Any of the files can be stored outside the assembly but the connection string must be changed

```xml
<add name="AdventureWorksEntities"
    providerName="System.Data.EntityClient"
    connectionString="metadata=\
        \Model1.cSDL|\Model1.ssDL|\Model1.mSL;
    provider=System.Data.SqlClient;
    provider connection string='
        data source=\sqlexpress;
        initial catalog=AdventureWorks;
        integrated security=True;
        multipleactiveresultsets=True;
        App=EntityFramework"
/>```

2.8 Connection Strings
Load Metadata from Resource or File System

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>assemblyFullName</td>
<td>ResourceLib, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null</td>
</tr>
<tr>
<td></td>
<td>If you specify wildcard (*) it searches:</td>
</tr>
<tr>
<td></td>
<td>1) calling assembly</td>
</tr>
<tr>
<td></td>
<td>2) referenced assemblies</td>
</tr>
<tr>
<td></td>
<td>3) assemblies in the bin folder</td>
</tr>
<tr>
<td>resourceName</td>
<td>The name of the resource, e.g. AdventureWorksModel.cSDL</td>
</tr>
<tr>
<td></td>
<td>If not specified, all metadata resources will be loaded</td>
</tr>
</tbody>
</table>

```
metadata=res:///<assemblyFullName>/<resourceName>
```

Entity Framework Connection Strings
Connection Strings

EntityConnectionStringBuilder

Should be used in conjunction with a SqlConnectionStringBuilder (or underlying provider)

- The connection string for the underlying data source is supplied by the ProviderConnectionString property which is not checked for valid keyword/value pairs

```
var sqlBuilder = new SqlConnectionStringBuilder();
sqlBuilder.DataSource = serverName;
// set other provider-specific properties

var efBuilder = new EntityConnectionStringBuilder();
efBuilder.Provider = providerName;
efBuilder.ProviderConnectionString = sqlBuilder.ConnectionString;
efBuilder.ProviderConnectionString = sqlBuilder.ToString();
string s = efBuilder.ConnectionString;
```


Connection Strings

Configuring Provider Connection Strings

- Connection strings provide the necessary information for an application to connect a data source
  - Server, database, security information, connection pool size
- Many parameters have multiple possible keywords for backwards compatibility e.g. these all mean the same
  - Data Source, server, address, addr, Network Address
- SQL Server (servername\instancename)

```
Data Source=FLORIDA\MIAMI;
Initial Catalog=Northwind;
Integrated Security=SSPI;
```

```
CSDL (Conceptual Schema Definition Language)

```xml
<EntityType Name="Accessory">
    <Key>
        <PropertyRef Name="ProductID" />
    </Key>
    <Property Name="ProductID" Type="Int32" Nullable="false" />
    <Property Name="SystemName" Type="String" Nullable="false" MaxLength="100" />
    <NavigationProperty Name="Product" Relationship="...">
        <FromRole="Accessory" ToRole="Product" />
    </NavigationProperty>
</EntityType>
```

SSDL (Store Schema Definition Language)

```xml
<EntityType Name="Accessory">
    <Key>
        <PropertyRef Name="ProductID" />
    </Key>
    <Property Name="ProductID" Type="int" Nullable="false" />
    <Property Name="SystemName" Type="nvarchar" Nullable="false" MaxLength="100" />
    <Property Name="rowguid" Nullable="false" Type="uniqueidentifier" StoreGeneratedPattern="Identity" />
    <Property Name="ModifiedDate" Type="timestamp" StoreGeneratedPattern="Computed" />
</EntityType>
```
Entity Data Models
Entity to Table

**MSL (Mapping Specification Language)**

```xml
<EntitySetMapping Name="AccessoryMapping"
    StoreEntitySet="Accessory"
    TypeName="Accessory">
    <ScalarProperty Name="ProductID"
        ColumnName="ProductID" />
    <ScalarProperty Name="SystemName"
        ColumnName="SystemName" />
</EntitySetMapping>
```

**Use ComplexTypes to group scalar properties for reuse**

```xml
<Schema Namespaces="AdventureWorksModel" Alias="Self">

    <ComplexType Name="Address">
        <Property Name="Street" Type="String" />
        <Property Name="City" Type="String" />
        <Property Name="Region" Type="String" />
        <Property Name="PostalCode" Type="String" />
    </ComplexType>

    <EntityType Name="Customer">
        <Property Name="CustomerID" ...="" />
        <Property Name="PostalAddress" Type="Self.Address" Nullable="false" />
        ...
    </EntityType>

</Schema>
```

- Complex types are defined and used in the conceptual layer and have NO affect on storage structure.
Entity Data Models
Mapping Complex Properties in MSL

Use MappingFragment to map a complex property

```xml
<EntitySetMapping Name="Customers">
  <EntityTypeMapping TypeName="Customer">
    <MappingFragment StoreEntitySet="Customer">
      <ScalarProperty Name="CustomerID" ColumnName="CID" />
      <ComplexProperty Name="PostalAddress" TypeName="Address">
        <ScalarProperty Name="Street" ColumnName="StreetAddress" />
        <ScalarProperty Name="City" ColumnName="City" />
        ...
      </ComplexProperty>
      ...
    </MappingFragment>
  </EntityTypeMapping>
</EntitySetMapping>
```

Entity Data Models
Conditional Mapping Fragments

Use a Condition element to filter rows from table into an entity

- In this example the Person table has a discriminator column
- If the value is S it means the person is a student

```xml
<EntityTypeMapping TypeName="Student">
  <MappingFragment StoreEntitySet="Person">
    <ScalarProperty Name="PersonID" ColumnName="pid" />
    <ScalarProperty Name="FirstName" ColumnName="fn" />
    <Condition ColumnName="PersonCategory" Value="S" />
  </MappingFragment>
</EntityTypeMapping>

<EntityTypeMapping TypeName="Employee">
  <MappingFragment StoreEntitySet="Person">
    <ScalarProperty Name="PersonID" ColumnName="pid" />
    <ScalarProperty Name="FirstName" ColumnName="fn" />
    <Condition ColumnName="PersonCategory" Value="E" />
  </MappingFragment>
</EntityTypeMapping>
```
2.17 Stored Procedures and Functions
Adding as a Method on ObjectContext or DbContext

```csharp
Bill b = db.GetBill(23).First();
```

**CSDL**

```xml
<FunctionImport Name="GetBill">
  <Parameter Name="StartProductID" Mode="In" Type="Int32" />
</FunctionImport>
```

**MSL**

```xml
<FunctionImportMapping FunctionImportName="GetBill" FunctionName="AWModel.Store.uspGetBillOfMaterials" />
```

**SSDL**

```xml
<Function Name="uspGetBillOfMaterials" ... Schema="dbo">
  <Parameter Name="StartProductID" Type="int" Mode="In" />
</Function>
```

VS2010 adding a stored procedure adds <Function> to the SSDL, but you must also “import” the function to the CSDL which then adds the mapping and method
VS2012 adding a stored procedure does all four parts

2.18 Stored Procedures and Functions
Using Stored Procedures to Modify Entities

**MSL**

```xml
<EntityTypeMapping TypeName="AdventureWorksModel.Product">
  <ModificationFunctionMapping>
    <UpdateFunction>
      <FunctionName="AWModel.Store.uspUpdateContact" />
      <ScalarProperty Name="ProductID" ParameterName="ProductID" Version="Current" />
    </UpdateFunction>
  </ModificationFunctionMapping>
</EntityTypeMapping>
```

- Version can be Current or Original

**SSDL**

```xml
<Function Name="uspUpdateProduct" ... Schema="dbo">
  <Parameter Name="ProductID" Type="int" Mode="In" />
</Function>
```
What Are Store-Defined Functions?

Define a SQL statement that is executed in the database

```xml
<Function Name="UpdateProductInOrder" IsComposable="false">
  <CommandText>
    UPDATE Orders
    SET ProductId = @productId
    WHERE OrderId = @orderId;
  </CommandText>
  <Parameter Name="productId" Type="int" Mode="In" />
  <Parameter Name="orderId" Type="int" Mode="In" />
</Function>
```

Expose the function as a method on the ObjectContext or DbContext by importing it into the conceptual model.

---

### <Function> Attributes

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name (required)</td>
<td>Name of the stored procedure or SSDL-defined function</td>
</tr>
<tr>
<td>StoreFunctionName</td>
<td>If different from Name</td>
</tr>
<tr>
<td>Schema</td>
<td>Name of the schema in which the stored procedure is defined</td>
</tr>
<tr>
<td>Aggregate</td>
<td>True if function returns an aggregate</td>
</tr>
<tr>
<td>BuiltIn</td>
<td>True if function is not a user-defined function or stored procedure</td>
</tr>
<tr>
<td>NiladicFunction</td>
<td>True if function has no input parameters and therefore doesn't need parentheses to call</td>
</tr>
<tr>
<td>IsComposable</td>
<td>A function is composable if the output can be the input for the other function, e.g., could be used in a FROM clause</td>
</tr>
</tbody>
</table>
2.21

Stored Procedures and Functions

Summary

```csharp
public ObjectResult<Department> GetDeptsByGroup(string group) {
    return base.ExecuteFunction<Department>("GetDeptsByGroup", ...)
```

```xml
_STEP:FunctionImport_Children>
<Parameter Name="GetDeptsByGroup" />
<Function Name="GetDeptsByGroup"/>
<DefiningExpression>
/* E-SQL */
</DefiningExpression>
</FunctionImport_Children>

_STEP:FunctionImportMapping_Children>
<FunctionImportMapping FunctionImportName="GetDeptsByGroup" FunctionName="AWModel.Store.uspSelectDepartments" />
</FunctionImportMapping_Children>

_STEP:EntityTypeMapping_Children>
<EntityTypeMapping TypeName="AdventureWorksModel.Product">
<ModificationFunctionMapping>
<DeleteFunction FunctionName="AWModel.Store.uspDeleteProduct" />
<ScalarProperty Name="Name" ParameterName="Name" Version="Current" />
</ModificationFunctionMapping>
</EntityTypeMapping_Children>

_STEP:FunctionImport_Children>
<FunctionImport_Children>
<Parameter Name="GetDeptsByGroup" />
<Function Name="GetDeptsByGroup"/>
<DefiningExpression>
/* T-SQL */
</DefiningExpression>
</FunctionImport_Children>

_STEP:FunctionImport_Children>
<FunctionImport_Children>
<Parameter Name="GetDeptsByGroup" />
<Function Name="GetDeptsByGroup"/>
<DefiningExpression>
/* T-SQL */
</DefiningExpression>
</FunctionImport_Children>

STEP:FunctionImport_Children>
<Parameter Name="GetDeptsByGroup" />
<Function Name="GetDeptsByGroup"/>
<DefiningExpression>
/* T-SQL */
</DefiningExpression>
</FunctionImport_Children>
```

2.22

Code First

What is Microsoft ADO.NET Entity Framework 4.1?

• aka “Magic Unicorn Edition” for VS2010 and later

• EF 4.1 introduces two new features
  • The DbContext API is a simplified abstraction over ObjectContext and a number of other types
  • Code First is a new development pattern that provides an alternative to the Database First and Model First patterns

• Code First is focused around defining your model using .NET classes
  • These classes can then be mapped to an existing database or be used to generate a database schema
  • Additional configuration can be supplied using Data Annotations or via a fluent API

EF 4.1 Released
Code First  
Define the Entities for the Model

Create the entities for the model by defining POCO ("plain old CLR object") classes

```csharp
public class Category {  
    public int CategoryID { get; set; }  
    public string CategoryName { get; set; }  
    public virtual ICollection<Product> Products { get; set; }
}
```

```csharp
public class Product {  
    public int ProductID { get; set; }  
    public string ProductName { get; set; }  
    public string CategoryID { get; set; }  
    public virtual Category Category { get; set; }
}
```

Relationships should be virtual to support lazy-loading

Tutorial: Code First with EF 4.1  
http://codefirst.codeplex.com/

Code First  
Define a Context for the Model

Define a context that derives from System.Data.Entity.DbContext and exposes a typed DbSet<TEntity> for each class in my model

```csharp
using System.Data.Entity;

public class NorthwindContext : DbContext
{
    public DbSet<Category> Categories { get; set; }
    public DbSet<Product> Products { get; set; }
}
```

You will need to add a reference to the EntityFramework.dll assembly and import System.Data.Entity namespace
2.25

Code First
Mapping to an Existing Database

The easiest way to point Code First to an existing database is to add a .config connection string with the same name as your derived DbContext

```xml
<connectionStrings>
  <add name="NorthwindContext"
      providerName="System.Data.SqlClient"
      connectionString="Data Source=.;\SINGLE;Initial Catalog=Northwind;Integrated Security=true;"
  />
</connectionStrings>
```

2.26

Code First
Modifying Data

Use the DbContext

```csharp
using (var db = new NorthwindContext())
{
    var food = new Category { CategoryName = "Foods" };
    db.Categories.Add(food);
    int recordsAffected = db.SaveChanges();
}
```

If you do not specify a connection string for an existing database then DbContext by convention creates a database for you on localhost\SQLEXPRESS

- The database will be named after the fully qualified name of your derived context
Code First Annotations

You can apply annotations to your model

```csharp
using System.ComponentModel.DataAnnotations;

public class Category
{
    [Key]
    public int CategoryID { get; set; }
    [MaxLength(20, ErrorMessage = "20 chars max!")]
    public string CategoryName { get; set; }
}
```

Annotations include

- Key, StringLength, MaxLength, ConcurrencyCheck, Required, Timestamp, ComplexType, Column, Table, InverseProperty, ForeignKey, DatabaseGenerated, NotMapped

System.ComponentModel.DataAnnotations Namespace

Code First Fluent API

Considered a more advanced feature and we would recommend using Data Annotations unless your requirements require you to use the fluent API

```csharp
public class NorthwindContext : DbContext
{
    protected override void OnModelCreating(DbModelBuilder modelBuilder)
    {
        modelBuilder.Entity<Supplier>().Property(s => s.Name).IsRequired();
    }
}
```

Configuring/Mapping Properties and Types with the Fluent API
http://msdn.microsoft.com/en-us/data/jj595617#2.4
Primary key convention

- Code First infers that a property is a primary key if a property on a class is named “ID” (not case sensitive), or the class name followed by “ID”
- If the type of the primary key property is numeric or GUID it will be configured as an identity column

Custom primary key convention

- Now any property in our model named Key will be configured as the primary key of whatever entity it's part of

```csharp
public class ProductContext : DbContext
{
    static ProductContext()
    {
        Database.SetInitializer(
            new DropCreateDatabaseIfModelChanges<ProductContext>());
    }
    public DbSet<Product> Products { get; set; }
    protected override void OnModelCreating(DbModelBuilder modelBuilder)
    {
        modelBuilder.Properties()
            .Where(p => p.Name == "Key")
            .Configure(p => p.IsKey());
    }```
2.31
Code First
Migration Support

For example, if you wanted to add a new column to a Blogs table called Url

```csharp
public partial class AddBlogUrl : DbMigration
{
    public override void Up()
    {
        AddColumn("Blogs", "Url", c => c.String());
    }
    public override void Down()
    {
        DropColumn("Blogs", "Url");
    }
}
```

Not on the list of exam 70-487 objectives

EF 4.3 Released
http://blogs.msdn.com/b/adonet/archive/2012/02/09/ef-4-3-released.aspx

2.32
Inheritance Hierarchies
Table-per-type vs. table-per-hierarchy

If we start with a model

The database schema can be implemented in two ways

- Table-per-hierarchy (TPH)
- Table-per-type (TPT): default when generating database from model (install EF Power Pack for more choices)
Inheritance Hierarchies

2.33

Table-per-hierarchy (TPH)

Characteristics

- One table with discriminator column

<table>
<thead>
<tr>
<th>GadgetId</th>
<th>Discriminator</th>
<th>Vendor</th>
<th>FormFactor</th>
<th>DisplayResolution</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>M</td>
<td>Nokia</td>
<td>Monoblock</td>
<td>NULL</td>
</tr>
<tr>
<td>2</td>
<td>N</td>
<td>Sony</td>
<td>NULL</td>
<td>1024*768</td>
</tr>
</tbody>
</table>

Advantages

- High performance of CRUD operations
- Minimum number of tables in the database

Disadvantages

- Data redundancy which leads to the data integrity violation if data was changed manually
- Complexity of adding and deleting new entities because it is necessary to add or delete columns to/from the result table

2.34

Inheritance Hierarchies

Table-per-type (TPT)

Characteristics

- One table for each entity

<table>
<thead>
<tr>
<th>GadgetId</th>
<th>Vendor</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Nokia</td>
</tr>
<tr>
<td>2</td>
<td>Sony</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MobilePhoneId</th>
<th>FormFactor</th>
<th>NetbookId</th>
<th>Resolution</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Monoblock</td>
<td>2</td>
<td>1280*768</td>
</tr>
</tbody>
</table>

Advantages

- Data integrity is preserved with no data redundancy
- Flexibility in object model modification

Disadvantages

- The speed of the CRUD operation decreases when the number of classes in the hierarchy grows
- A large number of tables in the database

Entity Framework Mapping Scenarios
Designer Tools
Entity Data Model Tools

- Entity Framework Designer and edmgen.exe
  - Can create a new model
  - Can validate an existing model

- Entity Framework Wizards
  - Can only perform single, one-time tasks

- Third party designers can be used, for example, LLBLGen Pro designer edits models for
  - LINQ to SQL
  - Entity Framework
  - LLBLGen Pro
  - nHibernate

Designer Tools
Generate Database from Model Wizard

- Remember: it does NOT affect underlying database
  - It creates the scripts, but you must manually execute them to create the database objects

- If you use complex types, the column names will be a concatenation of the complex type name and the property names, for example
  - Address_Street
  - Address_City
**Designer Tools**

**Comparing ObjectContext and DbContext**

 DbContext is simply a wrapper around ObjectContext
- For any DbContext instance, cast to IObjectContextAdapter interface and read the ObjectContext property

```csharp
var db = new NorthwindEntities(); // a DbContext
var oc = (db as IObjectContextAdapter).ObjectContext;
```

 DbContext uses DbSet<T> for the entity sets

```csharp
public class NorthwindEntities : DbContext { ...
    public DbSet<Product> Products { ...}
}
```

 DbSet<T> inherits from DbQuery<T> and implements some common interfaces

```csharp
public class DbSet<TEntity> : DbQuery<TEntity>, IDbSet<TEntity>, IQueryable<TEntity>, IEnumerable<TEntity>, IQueryable, IEnumerable
    where TEntity : class
```

 Use ToString to see the T-SQL (or whatever)

```csharp
string TSQL = query.ToString();
```
**Designer Tools**

**Navigation properties**

*Navigation properties in designer-generated code*

```csharp
public partial class Customer
{
    public virtual ICollection<Order> Orders { get; set; }
}
```

```csharp
public partial class Order
{
    public virtual Customer Customer { get; set; }
}
```

---

**Loading Patterns**

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>All in query</td>
<td>Compose a LINQ to Entities or Entity SQL query that uses navigation properties</td>
</tr>
<tr>
<td>Explicit loading</td>
<td>Check Property() or Collection().IsLoaded and then call Property() or Collection().Load to explicitly control the loading of related entities (requires MARS and multiple round-trips to the database)</td>
</tr>
<tr>
<td>Lazy loading</td>
<td>Related entities are automatically loaded when you access a navigation property (requires MARS and multiple round-trips to the database) Note: if you create an .edmx it sets LazyLoadingEnabled = true</td>
</tr>
<tr>
<td>Eager loading</td>
<td>Use Include() to define a query path that controls which related entities to return so only a single request to the database is required</td>
</tr>
</tbody>
</table>

```csharp
var products = db.Entry(category).Collection(c => c.Products);
if (!products.IsLoaded) products.Load();
```

```csharp
var contacts = from contact in db.Contacts
               .Include("SalesOrderHeaders.SalesOrderDetails")
               .Include(c => c.Address);
```

// to use lambda in Include using System.Data.Entity;
Entity SQL
Why use it?

Some edge cases
- To build very dynamic queries
- To use a database-specific function exposed by the provider
- To use QueryViews and Model-Defined Queries
- Some queries can be easier to express in Entity SQL

Entity SQL
VALUE Keyword

If your query returns a subset of “columns” then the DbDataRecord type must be used (VALUE not allowed)
- Returns IEnumerable<DbDataRecord>

```
FROM NorthwindEntities.Products AS p
```

If your query returns an entire “row” then the wrapping of the entity type can be prevented using VALUE
- Returns IEnumerable<DbDataRecord>

```
SELECT p FROM NorthwindEntities.Products AS p
```
- Returns IEnumerable<Product>

```
SELECT VALUE p FROM NorthwindEntities.Products AS p
```

Entity SQL Language

Entity SQL
VALUE Keyword

Entity SQL Language
Entity SQL

Projections and the VALUE Keyword

If your query returns a single “column” then simple types such as String can be used by using VALUE

- Returns IEnumerable<DbDataRecord>

```sql
SELECT p.ProductName FROM NorthwindEntities.Products AS p
```

- Returns IEnumerable<string>

```sql
SELECT VALUE p.ProductName FROM NorthwindEntities.Products AS p
```

Entity SQL

it

In a query builder method you refer to the current ObjectQuery command by using an alias

By default, the string “it” is the alias that represents the current command

```csharp
var db = new NorthwindEntities(); // a DbContext
var oc = (db as IObjectContextAdapter).ObjectContext;

ObjectQuery<Product> productQuery = oc.Products.Where("it.StandardCost > @cost",
new ObjectParameter("cost", 23));

productQuery.Name = "product";
ObjectQuery<Product> filteredProduct = productQuery
.OrderBy("product.ProductID");
```
Entity SQL
Paging with TOP, LIMIT, SKIP

The TOP sub-clause specifies that only the first set of rows will be returned

```
SELECT VALUE TOP(10) contact
FROM AdventureWorksEntities.Contacts AS contact
```

Physical paging can be performed by using LIMIT and SKIP sub-clauses in ORDER BY clause

- LIMIT or SKIP cannot be used separately from ORDER BY

```
SELECT VALUE p
FROM AdventureWorksEntities.Products AS p
ORDER BY p.ListPrice SKIP 50 LIMIT 10
```

- An Entity SQL query is invalid if both the TOP modifier and the SKIP sub-clause are present in the same expression

Entity SQL
Entity Set Operators

- **EXCEPT**
  - Distinct values from the left that are not also from the right

- **INTERSECT**
  - Distinct values from both left and right sides

- **UNION (and UNION ALL)**
  - Distinct values combined from left and right sides (UNION ALL includes duplicates)

- **IN**
  - Determines whether a value matches any value in a collection

- **EXISTS**
  - Determines if a collection is empty

Comparison Semantics
Entity SQL
CREATEREF, DEREF, REF

❖ A REF is a reference to a persisted entity
  • It’s a pointer to an entity so you have to de-reference it (using the DEREF keyword or by referencing a property) to get to the actual entity
  • To get one of these REFs you can either start with an entity (the REF keyword) or you can make one from a key value (the CREATEREF keyword)

❖ To create references to the top 5 rows of Customers

```
SELECT TOP(5) REF(c) FROM NorthwindContext.Customers AS c
```

❖ Why?
  • A REF is a ‘lightweight’ entity in which we don’t need to spend resources in creating and maintaining the full entity state/values until it is really necessary

---

EntityClient
Using the EntityClient Data Provider

❖ Similar to any other .NET data provider

```

var con = new EntityConnection("name=NW");
con.Open();
```

❖ EntityCommand uses Entity SQL

```
var cmd = new EntityCommand();
cmd.Connection = con;
cmd.CommandText = "SELECT VALUE p.ProductName FROM Products AS p";
```

❖ EntityDataReader must be opened with sequential access

```
var reader = cmd.ExecuteReader(CommandBehavior.SequentialAccess);
```

EntityClient Provider for the Entity Framework
EntityClient
Query Plan Caching

- By default, Entity SQL queries are cached to improve performance if you re-execute the same statement.
- The strings must be identical (case-sensitive) but can use parameters.
- To disable, set EntityCommand object’s EnablePlanCaching property to false.

Query Plan Caching

Querying Summary
When To Use...

- LINQ to Entities
  - Use this most of the time
  - Materializes data into entities tracked by DbContext

- Entity SQL with ObjectQuery (use ObjectContext)
  - When you need a feature supported by Entity SQL that is not supported by LINQ to Entities
  - Materializes data into entities tracked by ObjectContext
  - Set MergeOption.NoTracking to improve performance

- Entity SQL with EntityClient
  - For best read-only performance because it does NOT materialize data into entities that are change tracked
Performance

Deployed with .NET Framework 4.5
- Automatic compilation of LINQ to Entities queries
- Support for: enums, table-valued functions, spatial data types (DbGeography and DbGeometry)
- Multiple-diagrams per model allows you to have several diagrams that visualize subsections of your overall model
- Shapes can have colour applied

Transactions
ACID

- Atomic aka “all or nothing”
  - Groups multiple actions into a single, indivisible transaction
- Consistent
  - Leave the system in a consistent state after the transaction, for example, if we credit $100, we must also debit $100
- Isolated (different levels)
  - Apply locks while the transaction is occurring to isolate it
- Durable (or volatile)
  - A durable resource manager is one that can recover a failed transaction (for example, SQL Server uses logs)
  - A volatile resource manager cannot recover

Entity Framework 5.0 Performance Improvements
Transactions
Durable and Volatile

The durability of a resource manager refers to whether the resource manager supports failure recovery

- If it supports failure recovery, it persists data to durable storage during Phase1 (prepare) such that if it goes down, it can re-enlist in the transaction upon recovery and perform the proper actions based on notifications from the transaction manager.

In order for a resource to participate in a transaction, it must enlist in the transaction:

- You use the EnlistVolatile methods for volatile resources, and the EnlistDurable method for durable resources.

Transactions
Isolation Levels

ReadUncommitted (weakest isolation)
- Allows dirty reads, non-repeatable reads, and phantom data

ReadCommitted
- Allows non-repeatable reads and phantom data

RepeatableRead
- Allows phantom data

Serializable (strongest isolation)
- Completely isolates the transaction until it completes

Defaults
- SQL Server & ADO.NET “Classic” default to ReadCommitted
- TransactionScope defaults to Serializable

Implementing a Resource Manager

Volatile Resource Managers in .NET Bring Transactions to the Common Type
SQL Server 2005 introduced a new isolation level

- Once snapshot isolation is enabled, updated row versions for each transaction are maintained in tempdb

```sql
ALTER DATABASE MyDatabase
SET ALLOW_SNAPSHOT_ISOLATION ON
```

- The term “snapshot” reflects the fact that all queries in the transaction see the same version, or snapshot, of the database, based on the state of the database at the moment in time when the transaction begins

- No locks are acquired on the underlying data rows or data pages in a snapshot transaction, which permits other transactions to execute without being blocked by a prior transaction

```csharp
var options = new TransactionOptions {
    IsolationLevel = System.Transactions.IsolationLevel.Snapshot
};
```

Transactions
TransactionScope

```csharp
var options = new TransactionOptions {
    IsolationLevel = IsolationLevel.ReadCommitted,
    Timeout = TimeSpan.FromMinutes(2)
}; // auto-rollback if it doesn’t complete in two minutes
using (var tsOuter = new TransactionScope( TransactionScopeOption.Required, options)) {
    // perform action A
    OutputTransInfo(); // => {guid}:1
    DoWork();
    // perform action B
    tsOuter.Complete();
}
```

```csharp
private void DoWork() {
    using (var tsInner = new TransactionScope( TransactionScopeOption.RequiresNew)) {
        OutputTransInfo(); // => {guid}:2
        // perform action C
        // perform action D
        tsInner.Complete();
    }
}
```

```csharp
private void OutputTransInfo() {
    Console.WriteLine(Transaction.Current.TransactionInformation.LocalIdentifier);
}
```
Transactions

TransactionScopeOption

- **Required (default)**
  - Uses the ambient transaction if one already exists, otherwise, it creates a new transaction before entering the scope
  - Use Transaction.Current to get the ambient transaction
- **RequiresNew**
  - A new transaction is always created for the scope
- **Suppress**
  - The ambient transaction context is suppressed when creating the scope and all operations within the scope are done without an ambient transaction context

```csharp
using (var tsPreventRollback = new TransactionScope(
    TransactionScopeOption.Suppress)) {
    // call method that could throw an exception
}
```

Transactions

DependentTransaction class

- A DependentTransaction is a clone of a Transaction object created using the DependentClone method
  - Its sole purpose is to allow the application to come to rest and guarantee that the transaction cannot commit while work is still being performed on the transaction (for example, on a worker thread)
  - When the work done within the cloned transaction is finally complete and ready to be committed, it can inform the creator of the transaction using the Complete method
- The DependentCloneOption enumeration is used to determine the behavior on commit
  - BlockCommitUntilComplete
  - RollbackIfNotComplete
You can set a named savepoint within a transaction

- The savepoint defines a location to which a transaction can return if part of the transaction is conditionally cancelled

SqlTransaction.Save method (ADO.NET “Classic”)

- Creates a savepoint in the transaction that can be used to roll back a part of the transaction, and specifies the savepoint name
  
  ```csharp
  trans.Save("SP_Alpha");
  ```

- Equivalent to the T-SQL SAVE TRANSACTION statement

- Call the Rollback method to roll back to the savepoint instead of rolling back to the start of the transaction
  
  ```csharp
  trans.Rollback("SP_Alpha");
  ```

SqlTransaction.Save Method

Module 3
Creating and Consuming
ASP.NET Web API Services
Developing Windows Azure
and Web Services

Updated 11th April 2014

ASP.NET Web API
Contents

**Exam Topics: Design a Web API**
- Define HTTP resources with HTTP actions
- Plan appropriate URI space, map URI space using routing
- Choose appropriate HTTP method (get, put, post, delete) to meet requirements
- Choose appropriate format (Web API formats) for responses to meet requirements
- Plan when to make HTTP actions asynchronous

**Exam Topics: Implement a Web API**
- Accept data in JSON format (in JavaScript, in an AJAX callback)
- Use content negotiation to deliver different data formats to clients
- Define actions and parameters to handle data binding
- Use HttpResponseMessage to process client requests and server responses
- Implement dependency injection, along with the dependency resolver, to create more flexible applications
- Implement action filters and exception filters to manage controller execution
- Implement asynchronous and synchronous actions
- Implement streaming actions

**Exam Topics: Consume Web API web services**
- Consume Web API services by using HttpClient synchronously and asynchronously
- Send and receive requests in different formats (JSON/HTML/etc.)
HTTP and REST

3.3

SOAP Services versus RESTful Services

When developers talk about “web” services, they usually mean SOAP services, for example, ASMX files

• Actually, SOAP services can use any protocol, since SOAP defines a message format, not an architectural style, which is why WCF supports so many bindings

• RESTful services are true web services since they are built on the architecture of the web

• Proponents of RESTful services are sometimes called RESTafarians and are often quite passionate about how “evil” SOAP is and how “beautiful” REST is

• In WCF 3.5 Microsoft embraced REST so that you can use WCF to create both SOAP and RESTful services
  • In ASP.NET 4.5 Microsoft added Web API which is even better

3.4

21st Century Service-Oriented Architecture

• In 2000 Roy Fielding wrote a doctoral dissertation
  • The web is the world’s largest and most scalable distributed application platform
  • He described the architecture of the web and distilled from it an architectural style based on the factors that led to its success
  • He named this style REST and suggested its use to build services

• WCF isn’t tied to SOAP so Microsoft was able to quickly embrace REST once its simple power was understood

• RESTafarians believe that REST should be your first choice when building services
HTTP and REST

Architecture of the Web

**Principles**
- Addressable resources (URIs)
- Uniform interface for interacting with resources (HTTP verbs: GET, POST, DELETE, PUT, etc.)
- Standard resource formats (HTML, JPEG, XML, JSON, etc.)
- Statelessness between clients and servers (provides scalability and manageability)
- Hyperlinking for navigation between resources (relationships)

**GET**
- The cacheability of the GET verb contributes to the scalability of the web
- GETs are also considered “safe” in the sense that they should not cause side effects i.e. they don’t change resources

**SOAP**

- SOAP doesn’t follow the architecture of the web at all
  - Rather than URIs, SOAP uses *actions*, which are a thin veneer over method calls
  - SOAP services usually have only one URI and many different actions
  - SOAP is really an interoperable cross-platform remote procedure call (RPC) system

- When using HTTP, SOAP only uses one HTTP verb, POST
  - POSTs cannot be cached, so it’s not as scalable as GET

- But SOAP wasn’t designed for the web and goes out of its way to be protocol independent
REST and REST

REST versus SOAP

REST services combine nouns (e.g. resources defined by
URIs) with verbs (e.g. GET, DELETE)

```
PUT /AW.svc/Products(123)
Host: http://localhost:801
Content-Length: 223
Content-Type: application/json
```

```json
{ "ProductID" : "123",
...
```

SOAP services use a message to contain the nouns (e.g. the payload in the body) with verbs (the action in the header)

```
<s:Envelope xmlns:s=" ... >
  <s:Header>
    <To>http://.../Sample.svc</To>
    <Action>AddProduct</Action>
  </s:Header>
  <s:Body>
    <Product>
      <ProductID>123</ProductID>
      <ProductName>Fish</ProductName>
    </Product>
  </s:Body>
</s:Envelope>
```

HTTP and REST

POST versus PUT

• “The actual function performed by the POST method is
determined by the server” and “POST is designed to allow a
uniform method to cover the following functions: [...] Extending a database through an append operation”

• So POST can be used to insert and the server should respond with 201 (Created), or POST can be used for any meaning

• PUT “If the Request-URI refers to an already existing resource, the enclosed entity SHOULD be considered as a modified version of the one residing on the origin server. If the Request-URI does not point to an existing resource, and that URI is capable of being defined as a new resource by the requesting user agent, the origin server can create the resource with that URI”

• So PUT can be used to insert or update and the server should respond with either 201 (Created) or 200 (OK) or 204 (No content)

Method Definitions
http://www.w3.org/Protocols/rfc2616/rfc2616-sect9.html
Choose “common sense” URIs so developers can quickly work out how to access any resource and your service becomes almost “self-documenting”

- Design your service API as if you were designing the URLs for a web site i.e. make them logical enough that an end user could work out how to use them if shown a few examples

<table>
<thead>
<tr>
<th>Task</th>
<th>HTTP Method</th>
<th>Relative URI</th>
</tr>
</thead>
<tbody>
<tr>
<td>Retrieve all entities</td>
<td>GET</td>
<td>/api/orders</td>
</tr>
<tr>
<td>Retrieve single entity</td>
<td>GET</td>
<td>/api/orders/id</td>
</tr>
<tr>
<td>Retrieve by custom</td>
<td>GET</td>
<td>/api/orders?category=category</td>
</tr>
<tr>
<td>Create new entity</td>
<td>POST</td>
<td>/api/orders</td>
</tr>
<tr>
<td>Update entity</td>
<td>PUT</td>
<td>/api/orders/id</td>
</tr>
<tr>
<td>Remove entity</td>
<td>DELETE</td>
<td>/api/orders/id</td>
</tr>
</tbody>
</table>

With REST there is no need to design the semantics of the actions you want to perform because HTTP already defines them for you

- GET (or HEAD): retrieves a resource (or without the body)
- POST: Microsoft recommends this means create a new resource
- PUT: Microsoft recommends this means replace an existing resource with a new version (idempotent\(^1\))
- PATCH: update a resource with a subset of new values
- DELETE: remove a resource (idempotent\(^1\))
- CONNECT, TRANSFER, OPTIONS, TRACE

The Atom protocol defines

- MERGE: update a resource with a subset of new values

\(^1\) Idempotent means that the effect of calling it many times is the same as calling it once
REST has no restrictions on resource formats

- A REST service’s resource types are known as media types

The media type is returned in the HTTP header as the Content-Type

- XML: application/xml (text/xml is old standard)
- Feeds: application/rss+xml or application/atom+xml
- HTML and XHTML: text/html or application/xhtml+xml
- JavaScript Object Notation: application/json (most popular media type because it is the most compact)

ASP.NET Web API has built-in support for returning XML, JSON, and x-www-form-urlencoded, and you can add support for JSONP and custom formatters

ASP.NET Web API
Serializing Objects as XML, Atom, JSON

To generate this JSON (or similar XML, Atom, etc.)

```json
{"Name":"Alice","Age":23,"Pets":["Fido","Polly","Spot"]}
```

- Return an anonymous type instance from an ApiController’s action method

```csharp
public object Get()
{
    return new {
        Name = "Alice", Age = 23,
        Pets = new List<string> { "Fido", "Polly", "Spot" }
    };
}
```

- Warning! The XML serializer that would be used with Chrome (because it adds Accept: XML) breaks with anonymous types so you must use a named class instead and mark with attributes
### ASP.NET Web API

#### Controlling Transfer Mode and Bytes

- WCF and Web API can send messages using either buffered (the default) or streamed transfers
  - Set TransferMode.Streamed
- You can set the maximum amount of bytes sent
  ```csharp
  config.MaxBufferSize = 1024 * 1024 * 4; // 4 MB
  ```
- Or received
  ```csharp
  config.MaxReceivedMessageSize = 1024 * 1024 * 3; // 3 MB
  ```
- Both defaults are 64 KB
  ```csharp
  var config = new HttpSelfHostConfiguration(baseAddress);
  ```

[Stack Overflow question](http://stackoverflow.com/questions/6462571/c-sharp-wcf-web-api-4-maxreceivedmessagesize)

### ASP.NET Web API

#### async and await work as a pair

- By using the new async and await keywords, you can use resources to create an asynchronous method almost as easily as you create a synchronous method
  ```csharp
  public async Task<int> AccessTheWebAsync()
  {
    var client = new HttpClient();
    Task<string> getStringTask = client.GetStringAsync("http://msdn.microsoft.com");
    DoIndependentWork(); // executes while async op works
    string urlContents = await getStringTask;
    return urlContents.Length;
  }
  ```

- async modifier (required)
- Task<T> return type and Async suffix for name (optional)
  ```csharp
  using System.Threading.Tasks;
  using System.Net.Http;
  ```
By default, the JSON and XML formatters write all objects as values

- This is a particular problem if your object graph contains cycles, because the serializer will throw an exception when it detects a loop in the graph

To preserve object references in JSON, add the following code to Application_Start method in the Global.asax file

```csharp
var json = GlobalConfiguration.Configuration.Formatters.JsonFormatter;
json.SerializerSettings.PreserveReferencesHandling = Newtonsoft.Json.PreserveReferencesHandling.All;
```

To preserve object references in JSON, applyDataContract to classes as you would with WCF

Handling Circular Object References
http://www.asp.net/web-api/overview/formats-and-model-binding/json-and-xml-serialization#handling_circular_object_references

This fix is decorate attributes on model class to control the serialization behavior on model or property level

- To ignore a property (so it isn’t serialized)

```csharp
public class Category
{
    [JsonIgnore] // JSON.NET serializer
    [IgnoreDataMember] // XML DataContractSerializer
    public virtual ICollection<Product> Products { get; set; }
}
```

- To preserve references

```csharp
[JsonObject(IsReference = true)]
public class Category

[DataContract(IsReference = true)]
public class Product
```

Loop Reference handling in Web API
http://code.msdn.microsoft.com/Loop-reference-handling-in-caaffa7
If you have this C# class

```csharp
public class Person {
    public string FirstName { get; set; }
    public int Age { get; set; }
}
```

The default serializer will generate this JSON

```
"FirstName":"Bob","Age":25
```

- It feels awkward to work with JavaScript objects where the first letter of a property name is a capital letter

The serializer will let us be smarter

```csharp
var json = GlobalConfiguration.Configuration.Formatters.JsonFormatter;
json.SerializerSettings.ContractResolver = new CamelCasePropertyNamesContractResolver();
```

ASP.NET Web API Tip #3: camelCasing JSON

---

If you have an Entity Data Model and have created an ApiController using the scaffolding make sure you

- Switch off proxy creation so that EF doesn’t make DynamicProxy objects to track changes for you (they can’t be serialized!)

```csharp
db.Configuration.ProxyCreationEnabled = false;
```

- Switch off lazy loading so that EF doesn’t automatically load related entities when serializing the objects you have chosen to return

```csharp
db.Configuration.LazyLoadingEnabled = false;
```

- Use explicit loading instead (call Include method in query)

```csharp
var products = db.Products.Include(p => p.Category);
```
Module 4
Extending and Securing
ASP.NET Web API Services
Developing Windows Azure
and Web Services

Updated 11th April 2014

<table>
<thead>
<tr>
<th>Topic</th>
<th>Slide</th>
</tr>
</thead>
<tbody>
<tr>
<td>OData</td>
<td>3</td>
</tr>
<tr>
<td>WCF Data Services</td>
<td>10</td>
</tr>
<tr>
<td>HTTP Methods</td>
<td>13</td>
</tr>
<tr>
<td>OData .NET Clients</td>
<td>16</td>
</tr>
<tr>
<td>Web API Security</td>
<td>19</td>
</tr>
</tbody>
</table>

From the 20480 HTML5 course review the following
- 20480.05.Ajax
- 20480.C.Cross.Domain.Requests

Exam Topic: Create and implement a WCF Data Service
- Address resources
- Implement filtering
- Create a query expression
- Access payload formats (including JSON)
- Use data service interceptors and service operators

Exam Topics: Secure a Web API
- Implement HTTP Basic authentication over SSL
- Implement Windows Auth
- Enable cross-domain requests
- Prevent cross-site request forgery (XSRF)
- Implement, and extend, authorization filters to control access to the application
Comparing WCF, Web API, and WCF Data Services

OData

Overview

OData is a standard for building HTTP services that follow standards for querying the data model:

• It defines a query syntax using URIs similar to SQL

Two technologies for creating an OData service:

• WCF Data Services (.NET 3.5 +)
• ASP.NET Web API OData (.NET 4.5 +)

WCF Data Services and OData At-a-Glance
http://msdn.microsoft.com/en-us/data/aa937697

WCF Data Services

WCF Data Services Blog
http://blogs.msdn.com/b/astoriateam/
OData
URL Query Syntax Basics

To select or order by multiple columns use a comma-separated list

Case-sensitive!

Must use $ prefix for keywords
- $select, $filter, $orderby, $top, $skip
- /$count: returns an int
- $inlinecount: a count is included with the feed

OData Core
http://www.odata.org/documentation/odata-v3-documentation/odata-core/

OData
$format

Allows query to specify return data format
- $format=json
- $format=atom
- $format=xml
- $format=service-specific-format

Warning! WCF Data Services ignores $format
- Instead you must add an Accept header to the HTTP request with application/json at the beginning of the list
The syntax of a $expand query option is a comma-separated list of Navigation Properties

- Each Navigation Property can be followed by a forward slash and another Navigation Property to enable identifying a multi-level relationship

```
/Categories?$expand=Products

/Categories?$expand=Products/Suppliers

/Products?$expand=Category,Suppliers
```

Expand System Query Option ($expand)
http://www.odata.org/documentation/uri-conventions#ExpandSystemQueryOption

---

Returns an EDMX document that contains a complete description of the feeds, types, properties, relationships exposed by the service in EDM

- It isn’t queryable so if you want to find Types that have an Address property you have to retrieve the whole EDMX and search the xml yourself

```
http://services.odata.org/Northwind/Northwind.svc/$metadata
```

Queryable OData Metadata
4.9
OData
URI Query Syntax Examples

<table>
<thead>
<tr>
<th>URI</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>/Customers('ALFKI')/ContactName</td>
<td>An XML element that contains the ContactName property value for a specific Customer</td>
</tr>
<tr>
<td>/Customers('ALFKI')/ContactName/$value</td>
<td>Only the string “Maria Anders” without the XML element</td>
</tr>
<tr>
<td>/Customers('ALFKI')/Orders</td>
<td>All the orders that are related to a specific Customer</td>
</tr>
<tr>
<td>/Orders(10643)/Customer</td>
<td>A reference to the Customer entity to which a specific Order entity belongs</td>
</tr>
<tr>
<td>/Orders?$filter=not endswith(ShipPostalCode,'100')</td>
<td>All the orders the postal codes of which do not end in 100</td>
</tr>
<tr>
<td>/Categories(1)/$links/Products</td>
<td>Links to the data instead of the actual data e.g. <code>&lt;uri&gt;http://…/Products(4)&lt;/uri&gt;</code></td>
</tr>
<tr>
<td>/Categories?$select=Name, Products&amp;$expand=Products</td>
<td>Must select Products if expanding Products</td>
</tr>
</tbody>
</table>

4.10
WCF Data Services
How to Create

- Project - Add New Item - WCF Data Service
- Create a context class that represents your data
  - ADO.NET Entity Data Model is easiest
  - Or any class that has properties of type IQueryable<T> where T is an “entity” (and optionally implements IUpdatable)
- Use context class in DataService<TContext>
- Set permissions for Entity Sets and Service Operations

```csharp
public class BlogService : DataService<BlogDbContext>
{
    public static void InitializeService(DataServiceConfiguration config)
    {
        config.SetEntitySetAccessRule("Blogs", EntitySetRights.All);
        config.SetServiceOperationAccessRule("MyServiceOperation", ServiceOperationRights.All);
    }
```
Differences in OData V2 and V3

- In V2 you could ask for application/json in Accept
- In V3 you must ask for application/json;odata=verbose
- WCF Data Services 5.0 will return a 415 in response to a request for application/json if the service configuration allows a v3 response and if the client does not specify a MaxDataServiceVersion header or specifies a MaxDataServiceVersion header of 3.0

Clients should always send a value for the MaxDataServiceVersion header to ensure forward compatibility

```
GET http://odata.example.org/Items?$top=5 HTTP/1.1
Accept: application/json
MaxDataServiceVersion: 2.0
```

What happened to application/json in WCF DS 5.0?

It’s possible to modify the headers with jQuery

```
$.ajax(url, {
  dataType: "json",
  beforeSend: function (xhr) {
    xhr.setRequestHeader("Accept", "application/json;odata=verbose");
    xhr.setRequestHeader("MaxDataServiceVersion", "3.0");
  },
  success: function (data) {
    // do something interesting with the data you get back.
  }
});
```

How to enable $format support in WCF Data Services

- You get the incoming request, remove the $format query option from the URI, and change the Accept header to application/json

Getting JSON Out of WCF Data Services
WCF Data Services
Intercepting Queries and Changes

WCF Data Services enables an application to intercept request messages so that you can add custom logic.

- Define a query interceptor for the Orders entity set

```csharp
[QueryInterceptor("Orders")]
public Expression<Func<Order, bool>> OnQueryOrders() {
}
```

- Check UpdateOperations parameter to determine type

```csharp
[ChangeInterceptor("Products")]
public void OnChangeProducts(Product product, UpdateOperations operation) {
    if (operation == UpdateOperations.Delete) {
        // Code goes here...
    }
}
```

Interceptors (WCF Data Services)

WCF Data Services
Custom Data Service Providers

A Data Service Provider is simply a .NET class that sits between the Data Services Framework and the underlying data source that’s being exposed.

```csharp
public class MyDataSource : IUpdatable {
    public IQueryable<Product> Products { get { ... }
}
```

To make your service updatable

To make your service queryable

Custom Data Service Providers
HTTP Methods
MERGE and PATCH

To update a column of a record without overwriting other columns, use MERGE or PATCH verbs and only pass the changed column values.

```
PATCH /AW.svc/Contacts(23)
Host: AdventureWorks.com
Content-Type: application-json
{ State: 'CA' }
```

Warning!

- By default the WCF Data Services client library passes all properties using a MERGE, not just the ones that have changed.
- Use SaveChangesOptions.ReplaceOnUpdate to change to using a PUT (which requires all properties to be sent).

WCF Data Services: Optimizing bandwidth usage and performance with updates

HTTP Methods
Support for CRUD Operations

To enable CRUD operations, IIS must allow the following methods on the .svc extension:

- PUT
- DELETE
HTTP Methods

X-HTTP-Method

- Some network intermediaries block HTTP verbs like DELETE or PUT or MERGE
  - "Verb tunnelling" or "POST tunnelling" gets around this
- Uses HTTP POST to "wrap" another verb

```
POST /Categories(5)
HTTP/1.1
Host: AdventureWorks.com
X-HTTP-Method: DELETE
```

- This feature is automatically enabled in WCF Data Services
  - To enable in a .NET client
    ```csharp
    DataServiceContext.UsePostTunneling = true;
    ```

OData .NET Clients

Loading Related Entities

- DataServiceContext does not support lazy loading so you must use the LoadProperty method to explicitly load related entities just before reading them
  - Note: DataServiceContext simulates ObjectContext in EF
    ```csharp
    DataServiceContext.LoadProperty(category, "Products");
    foreach(var item in category.Products) {
    }
    ```
- Or use Expand method to pre-load ("eager-loading"), similar to Include method, but re-named to look like the OData $expand keyword
    ```csharp
    var query = DataServiceContext.Categories.Expand("Products");
    ```
4.19
OData .NET Clients
Troubleshooting

To find out how a LINQ to OData query will translate into an OData URL use RequestUri

```csharp
var query = from p in DataServiceContext.Products
            where p.Color == "Red"
            select p;
string uri = ((DataServiceQuery)query).RequestUri.ToString();
```

http://localhost:1034/AW.svc/Products()?$filter=Color eq 'Red'

4.20
OData .NET Clients
Set Headers in the Client Request

Create an event handler for SendRequest2

```csharp
DataServiceContext.SendingRequest2 += db_SendingRequest2;
```

Add the header in the event handler

```csharp
private void db_SendingRequest2(object sender, SendingRequest2EventArgs e)
{
    // Add an Authorization header that contains an OAuth WRAP access token to the request
    e.RequestMessage.SetHeader(  
        "Authorization",  
        "WRAP access_token=" + e.PropertyValue);  
}
```

How to: Set Headers in the Client Request (WCF Data Services)
Web API Security

Overview

For the most common scenario—JavaScript in a Web page accessing a Web API service on the same site—discussing security for ASP.NET Web API is redundant

- Provided that you authenticate your users and authorize access to the Web Forms/Views holding the JavaScript that consumes your services, you’ve probably provided all the security your services need

There’s one exception (and it’s an important one):

- ASP.NET doesn’t automatically protect you against Cross-Site Request Forgery (CSRF/XSRF) attacks

Further Study

Using this book, you can gain a solid understanding of the security techniques relevant to ASP.NET Web API

- All the underlying concepts are introduced from basic principles and developed to the point where you can use them confidently, knowing what you are doing

Enabling and Customizing ASP.NET Web API Services Security

Pro ASP.NET Web API Security: Securing ASP.NET Web API
http://www.amazon.co.uk/Pro-ASP-NET-Web-API-Security/dp/1430257822/
Module 5
Creating WCF Services
Developing Windows Azure and Web Services

Updated 11th April 2014

Creating WCF Services
Contents

<table>
<thead>
<tr>
<th>Topic</th>
<th>Slide</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Contracts</td>
<td>3</td>
</tr>
<tr>
<td>Service Contracts</td>
<td>17</td>
</tr>
<tr>
<td>Bindings</td>
<td>21</td>
</tr>
<tr>
<td>Clients</td>
<td>25</td>
</tr>
<tr>
<td>Faults</td>
<td>30</td>
</tr>
<tr>
<td>RESTful Services</td>
<td>34</td>
</tr>
<tr>
<td>Endpoints</td>
<td>39</td>
</tr>
<tr>
<td>Behaviors</td>
<td>43</td>
</tr>
<tr>
<td>Sessions</td>
<td>51</td>
</tr>
<tr>
<td>Summaries</td>
<td>53</td>
</tr>
</tbody>
</table>

Exam Topics: Create a WCF service
- Create contracts (service, data, message, callback, and fault)

Exam Topics: Version a WCF service
- Version different types of contracts (message, service, data)
- Configure address, binding, and routing service versioning

Exam Topics: Consume WCF services
- Generate proxies by using SvcUtil
- Generate proxies by creating a service reference
- Create and implement channel factories

Exam Topics: Host and manage services
- Manage services concurrency (single, multiple, reentrant)
- Choose an instancing mode (per call, per session, singleton)

Exam Topics: Configure WCF services by using configuration settings or the API
- Configure service behaviors
- Configure service endpoints
- Configure binding
- Specify a service contract
- Expose service metadata (XSDs, WSDL, metadata exchange)
**5.3 Data Contracts**

### Serializing Enumerations

```csharp
// optional: rename the type in WSDL
[DataContract(Name = "CarCondition")]
public enum CarConditionEnum {
    [EnumMember]
    New,
    [Optional: rename the value
    [EnumMember(Valuae = "Used")]
    PreviouslyOwned,
    [EnumMember]
    Rental,
    Broken,
    Stolen
}
```

If you use an enum value that is not marked as an EnumMember and it is serialized then WCF will throw an exception.

```csharp
var car = new Car {
    Model = "Volvo",
    Condition = CarConditionEnum.PreviouslyOwned
}
```

### Enumeration Types in Data Contracts

- HelpLink
- InnerException (of type ExceptionDetail)
- Message
- StackTrace
- Type

---

**5.4 Data Contracts**

### Use Serializable Types

- Your data contracts must only use serializable types
  - For example, Exception is NOT serializable, which is why Microsoft created an ExceptionDetail type for use when returning a serialized exception during faults

**ExceptionDetail Properties**

- HelpLink
- InnerException (of type ExceptionDetail)
- Message
- StackTrace
- Type

---

EnumMemberAttribute Class

ExceptionDetail Class
5.5 Data Contracts
Serializing Object References

DataContractSerializer serializes by value (default)

```csharp
[DataMember] public Address BillTo = someAddress;
[DataMember] public Address ShipTo = someAddress;
```

To get the DataContractSerializer to preserve object references (especially useful for circular references)

```csharp
[DataContract(IsReference=true)]
public class Order

[DataMember]
public Address BillTo = someAddress;

[DataMember]
public Address ShipTo = someAddress;
```

5.6 Data Contracts
Data Member Default Values

When a reference type is null, xsi:nil is used in XML

```csharp
[DataMember]
public string FirstName = null;
```

To exclude element when values are equal to defaults

```csharp
[DataMember]
public int Height = 0;
[DataMember]
public int Weight = 10;
```

```csharp
[DataMember(EmitDefaultValue=false)]
public int Height = 0;
[DataMember(EmitDefaultValue=false)]
public int Weight = 10;
```

DataMemberAttribute.EmitDefaultValue
Data Contracts

Data Member Order

- Members ordered base type first, then alphabetically
  
  ```
  [DataMember] public string FirstName;
  [DataMember] public string LastName;
  [DataMember] public byte Age;
  ```

- To order members explicitly
  
  ```
  [DataMember(Order = 1)] ... FirstName;
  [DataMember(Order = 2)] ... LastName;
  [DataMember(Order = 3)] ... Age;
  ```

- What order would this use?
  
  ```
  [DataMember] ... FirstName;
  [DataMember(Order = 1)] ... LastName;
  [DataMember] ... Age;
  ```

- Because members without order written first

Data Members Order

Data Contracts

XML Namespaces

- It is best practice to provide a namespace for your data contracts rather than use the default (tempuri.org)
  
  ```
  [DataContract(Namespace="http://www.firebrand.com/hr/2012/11")]
  public class Employee
  ```

- You can do this globally by using the assembly-level attribute ContractNamespace
  
  ```
  [assembly:ContractNamespace("http://www.firebrand.com/hr/2012/11",
  ClrNamespace = "Firebrand")]
  ```

namespace Firebrand
{
  public class Employee
  ```
Data Contracts
Versioning

“Strict schema validity”
- This means in both directions (new-to-old and old-to-new) so data and service contracts must be considered immutable
- If a new version is required then a new data contract must be created with a different name or namespace and the service contract should add a new operation

Automatic versioning is the default
- Missing data members don’t cause exceptions so to throw a SerializationException if a data member is missing when deserializing set IsRequired to true

```csharp
[DataMember(IsRequired=true)] //default is false
public string FirstName { get; set; }
```

**DataMemberAttribute.IsRequired Property**

Data Contracts
Round-Tripping (1/2)

If strict schema validity is NOT required, and you need to be able to round-trip an instance of a data contract with older clients, use IExtensibleDataObject

- IExtensibleDataObject provides a data structure to store extra data encountered during deserialization
- In a roundtrip operation where data is received, processed, and sent back, any extra data is returned to the sender intact
- If you do not implement the interface, any extra data is ignored and lost during a roundtrip operation
- Useful to store data received from future versions of the contract
- Note: svcutil and VS-generated code implement this but you need to implement it yourself if reusing data contracts

**IExtensibleDataObject Interface**
Data Contracts
Round-Tripping (2/2)

 Bermark versions e.g. Person and PersonV2 with same Name and Namespace

• Best practice to apply an order to new members

```csharp
[DataContract(Name="Person", Namespace="http://.../")] public class Person : IExtensibleDataObject {
    public ExtensionDataObject ExtensionData { get; set; }
}

[DataContract(Name="Person", Namespace="http://.../")] public class PersonV2 : IExtensibleDataObject {
    public ExtensionDataObject ExtensionData { get; set; }
    [DataMember(Order=2)]
    public int Age { get; set; } // new member
}
```

5.12

Data Contracts
Returning Collections from Operations

• Add Service Reference...

Reuse only applies to data contracts, not service & operation contracts

• ChannelFactories

The client should have a reference to the contracts so will know exactly what collection type to use.
Data Contracts
Putting Message Properties in SOAP Header

🌟 Use MessageHeader to put a property in SOAP header

```csharp
[MessageContract(WrapperNamespace="http://www.aw.com")]
public class Person {
    [MessageHeader(Namespace="http://www.aw.com")]
    public int ID { get; set; }
    [MessageBodyMember(Namespace="http://www.aw.com")]
    public string FirstName { get; set; }
}
```

🌟 Warning!

- If you use MessageContract then you can only have a single input parameter on the operation of type Message or a class like the one above that has MessageContract applied.

Data Contracts
Putting Arrays in SOAP Header

🌟 Arrays normally get wrapped

```csharp
[MessageContract]
public class Software {
    [MessageHeader]
    public Key[] Keys { get; set; }
}
```

🌟 Use MessageHeaderArray to avoid wrappers

```csharp
[MessageContract]
public class Software {
    [MessageHeaderArray]
    public Key[] Keys { get; set; }
}
```

🌟 Cannot be used with collections
Data Contracts
Protecting Individual Properties in a Message

To protect a property such as a credit card number
- Change theDataContract to a MessageContract because
  DataContracts do not support ProtectionLevel

```csharp
[MessageContract]
public class Person {
    [MessageBodyMember(ProtectionLevel = ProtectionLevel.EncryptAndSign)]
    public string CreditCard { get; set; }
}
```

ProtectionLevel: None, Sign, EncryptAndSign
- Also on ServiceContract, OperationContract, FaultContract
- MessageBodyMember uses highest protection set on any member
- MessageHeader can be individually set

Warning! You must configure certificates on service and client to perform the encryption

How to: Set the ProtectionLevel Property

using System.Net.Security;

Data Contracts
Switching Serializers

WCF can use two different serialization technologies
-DataContractSerializer (default) or XmlSerializer

Apply XmlSerializerFormat to interface/class/operation
- Use XmlSerializer attributes to customize XML generated

```csharp
[ServiceContract]
[XmlSerializerFormat]
public interface IHRService {
    [OperationContract]
    public void Hire(Employee emp)
}
```

```csharp
[ServiceContract]
[XmlSerializerFormat]
public class Employee {
    [XmlAttribute]
    public int Age { get; set; }
    [XmlIgnore]
    public decimal Salary { get; set; }
}
```

Warning! AnyDataContract attributes are ignored and all public fields and properties will be serialized unless you apply XmlIgnore attribute

Using the XmlSerializer Class
Service Contracts
Best Practice

Best practice is to separate contract from implementation

- Use an interface for the contract

```csharp
[ServiceContract]
public interface ISampleService
```

- Use a class for the implementation

```csharp
public class SampleService : ISampleService
```

You can (but shouldn’t) use a class for both contract and implementation

```csharp
[ServiceContract] // bad practice
public class SampleService
```

Service Contracts
Renaming Services and Operations

To rename a service and its operations

```csharp
[ServiceContract(Name="Employee")]
public interface Serf
{
    [OperationContract(Name="Fire")]
    void Terminate();
}
```

Other parameters

- Service: CallbackContract, Namespace, ConfigurationName
- Operation: Action, IsOneWay, ReplyAction

OperationContractAttribute Class

ServiceContractAttribute Class
Service Contracts
Making an Operation More Flexible

For an operation to accept any SOAP message declare a single input parameter as a Message instead of a data contract type

```csharp
[OperationContract]
void ProcessData(Person p);
```

```csharp
[OperationContract]
void ProcessData(Message m);
```

To read a Message

- `m.GetReaderAtBodyContents()` returns `XmlDictionaryReader`
- `m.GetBody()` returns the message body as a typed object

```csharp
if (!m.IsEmpty) {
    // check for null message in service or client; on client-side should also check IsFault
    Person p = m.GetBody<Person>();
}
```

### Using the Message Class

- [For more information](http://msdn.microsoft.com/en-us/library/system.servicemodel.channels.message.aspx)

### Message Class

- [For more information](http://msdn.microsoft.com/en-us/library/ms734675(v=vs.110).aspx)

---

Service Contracts
Using Message as Return Type

For maximum flexibility in the return values for an operation, return a Message, for example

```csharp
public Message GetData()
{
    Person p = new Person { Name = "John Doe", Age = 42 };,
    return Message.CreateMessage(ver, "GetDataResponse", p);
}
```

To determine what Accepts header was sent by the client and then return the data in the required format

```csharp
```

### IncomingWebRequestContext.Accept

- [For more information](http://msdn.microsoft.com/en-us/library/system.servicemodel.web.incomingwebrequestcontext.accept.aspx)
### Message Encodings

- **Text**: ASCII, UTF16, UTF8 (default)
- **Binary**: uses a WCF proprietary algorithm

#### Message Transmission Optimization Mechanism (MTOM)

- MTOM is a mechanism for transmitting large binary attachments with SOAP messages as raw bytes, allowing for smaller messages while still interoperating with non-.NET systems.
- Best for 10kb+ payloads

```xml
<system.serviceModel>
  <bindings>
    <wsHttpBinding>
      <binding messageEncoding="Mtom">...
    </wsHttpBinding>
  </bindings>
</system.serviceModel>
```

### Choosing a Binding

1. Basic Profile 1.1 as used by ASP.NET XML Web Services (.asmx)
2. Every binding uses SOAP messages except `webHttpBinding`
3. WS2007 is a later WS-* version with improved reliable sessions

<table>
<thead>
<tr>
<th>Binding</th>
<th>Interop</th>
<th>Security (default)</th>
<th>Encoding</th>
<th>Sessions</th>
<th>WS-*</th>
<th>Transactions</th>
<th>Duplex</th>
<th>Streaming</th>
</tr>
</thead>
<tbody>
<tr>
<td>BasicHttp</td>
<td>Basic</td>
<td>None, Transport, Message</td>
<td>Text, Mtom</td>
<td>x x x x</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WebHttp</td>
<td>REST</td>
<td>None, Transport</td>
<td>Text</td>
<td>x x x x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WSHttt</td>
<td>WS</td>
<td>None, Transport, Message</td>
<td>Text, Mtom</td>
<td>✓ ✓ ✓ ✓</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WS2007Http</td>
<td>WS</td>
<td>None, Transport, Message</td>
<td>Text, Mtom</td>
<td>✓ ✓ ✓ x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WSDualHttp</td>
<td>WS</td>
<td>None, Message</td>
<td>Text, Mtom</td>
<td>✓ ✓ ✓ ✓</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WSFederationHttp</td>
<td>WS-Fed</td>
<td>None, Transport, Message</td>
<td>Text, Mtom</td>
<td>✓ ✓ ✓ x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NetTcp</td>
<td>.NET</td>
<td>None, Transport, Message</td>
<td>Binary</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NetNamedPipes</td>
<td>.NET</td>
<td>None, Transport</td>
<td>Binary</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NetMsmq</td>
<td>.NET</td>
<td>None, Transport, Message</td>
<td>Binary</td>
<td>✓ x ✓ x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Others...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**System-Provided Bindings**


[Choosing a Message Encoder](http://msdn.microsoft.com/en-us/library/aa751889(v=vs.100).aspx)
5.23 Bindings

Custom Bindings

- **Order must be**
  - Protocol(s)
  - MessageEncoding
  - Transport

- **EXCEPT**
  - Transports that use a stream-oriented protocol such as TCP and named pipes support stream-based transport upgrades, so you can add a windowsStreamSecurity or sslStreamSecurity element between the encoding and transport

- **REMEMBER**: transport is always LAST in list

```
<customBinding>
  <binding name="MyBinding" ... >
    <protocol1 ... />
    <protocol2 ... />
    <somethingMessageEncoding ... />
    <somethingTransport ... />
  </binding>
</customBinding>
```

Choosing a Transport

System-Provided Bindings

5.24 Bindings

MSMQ Transport Types

- **msmqTransport**
  - Suitable for cross-machine communication between WCF applications

- **msmqIntegrationTransport**
  - Suitable for cross-machine communication between a WCF application and *existing* Message Queuing applications

Queueing in WCF
Creating .NET Proxies

A proxy to a service can be created at design-time with Add Service Reference... or SvcUtil.exe

- ...or created at run-time with channel factories

```csharp
var address = new EndpointAddress(...);
var binding = new BasicHttpBinding();
var factory = new ChannelFactory<ICalc>(binding, address);
ICalc proxy = factory.CreateChannel();
// after using the proxy, you should close it
(proxy as ICommunicationObject).Close();
```

- Can combine interfaces to avoid casting

```csharp
public interface ICalcWithComm : ICalc, ICommunicationObject {}
```

```csharp
var factory = new ChannelFactory<ICalcWithComm>(...);
proxy.Close(); // because proxy now makes BOTH interfaces visible
```

ChannelFactory<TChannel> Class

Using a Configuration with Channel Factories

Channel factories can load named endpoints from configuration files

```xml
<client>
  <endpoint name="MyEndpoint"
    address="net.tcp://server/AWService"
    binding="netTcpBinding"
    contract="Firebrand.IAWService"
  />
```

```csharp
var factory = new ChannelFactory<IAWService>("MyEndpoint");
```

ChannelFactory<TChannel> Constructor (String)
Clients

How To Reset a Proxy on the Client

After an exception on the client the proxy will be in a faulted state

• Calls to Close (or Dispose!) will cause another exception
• Call Abort and then re-create the proxy before trying again

```csharp
try {
    proxy.ProcessData(data);
} catch {
    if (proxy.State == CommunicationState.Faulted) {
        proxy.Abort();
        proxy = new ServiceClient();
    }
}
```

ICommunicationObject.Abort Method

Clients

.NET Framework Client Profile (in VS2010 only)

Subset of .NET Framework optimized for client apps

• Does NOT include ASP.NET and advanced WCF functionality, for example, [WebGet] and [WebInvoke] methods for REST in System.ServiceModel.Web namespace

Project templates that target the client profile

• Empty Project
• WPF and Windows Forms
• Console Application
• WCF Service Library
• Windows Service

To get full WCF features you need to change the target to .NET Framework 4
Clients
WCF Test Client

- C:\Program Files\Microsoft Visual Studio 11.0\Common7\IDE\n  - The left pane of the main window lists all the available services, along with their respective endpoints and operations, and editable App.config file (right-click to edit)

Supported
- Service Invocation: Request/Response and One-way message
- Bindings: all bindings supported by Svcutil.exe
- Controlling Session

NOT supported
- Duplex, transactions, Security: Certificate & Username/Pwd
- Bindings: WSFederation, Https, WebHttpBinding

Faults
Including Exception Details in Faults

Set to true to troubleshoot exceptions in a service
- As an attribute on the class implementation
  ```csharp
  [ServiceBehavior(IncludeExceptionDetailInFaults = true)]
  ```
- As a service behavior in .config
  ```xml
  <serviceDebug includeExceptionDetailInFaults="true" />
  ```

Serializes any service-side exceptions into a SOAP fault
- Allows the service to return any exception that is thrown by the service code even if the exception is not declared using the FaultContractAttribute including the server stack trace
- Specifies that general unhandled execution exceptions are to be converted into a FaultException<ExceptionDetail> and sent as a fault message

5.31 Faults
Catch Exceptions in the Correct Order

FaultException<T> derives from FaultException, and FaultException derives from CommunicationException, so it is important to catch them in the proper order.

```csharp
catch(TimeoutException te) {
    // operation exceeded specified timeout
}
catch(FaultException<T> feT) {
    // generic fault exception
}
catch(FaultException fe) {
    // non-generic fault exception
}
catch(CommunicationException ce) {
    // recoverable communication error
}
```

Sending and Receiving Faults

5.32 Faults
Faults, Exceptions, and Debugging

<table>
<thead>
<tr>
<th>IncludeExceptionDetailInFaults</th>
<th>FaultException&lt;T&gt;</th>
<th>Any other .NET exception</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>SOAP Fault containing T as the Detail property, so catch FaultException&lt;T&gt; on client</td>
<td>SOAP Fault with serialized exception as the Detail property including server stack trace so catch FaultException&lt;Exception Detail&gt; on client</td>
</tr>
<tr>
<td>False</td>
<td>SOAP Fault with no detail, so catch FaultException on client</td>
<td>SOAP Fault with no detail, so catch FaultException on client</td>
</tr>
</tbody>
</table>
Faults

Fault Contracts

To exclude any stack trace information when an exception occurs, throw a new instance of a custom fault exception

```csharp
[ServiceContract]
public interface IPersonService
{
    [OperationContract]
    [FaultContract(typeof(PersonFault))]
    void AddPerson(Person p);
}
```

```csharp
public class PersonService : IPersonService
{
    public void AddPerson(Person p)
    {
        ...
        catch(IOException ex)
        {
            throw new FaultException<PersonFault>(personFault,
                new FaultReason("I/O exception"));
        }
    }
}
```

RESTful Services

Enabling REST

To create a REST-compliant endpoint

- Use webHttpBinding and apply webHttp as an endpoint behavior

```xml
<endpoint address="kermit"
  bindings="webHttpBinding"
  behaviorConfiguration="RESTful"
  contract="Firebrand.ISampleService"
  automaticFormatSelectionEnabled="true"/>
```

- webHttp behavior can auto-document the endpoint and switch on automatic response format selection (XML or JSON)

```
http://localhost:801/Sample/kermit/Help
```
RESTful Services
Invoking REST

**SOAP only uses POST; REST uses POST (by default)**
- Or you can apply [WebInvoke] to map an operation to another HTTP verb such as PUT

```csharp
[OperationContract]
[WebInvoke(Method = "PUT")]
public string AddName(string newName);
```

```
PUT http://localhost:801/Sample/kermit/AddName
...
<newName>Alice</newName>
```

**Apply [WebGet] to pass simple types in a query string**

```csharp
[OperationContract]
[WebGet]
public string AddName(string newName);
```

```
GET http://localhost:801/Sample/kermit/AddName?newName='Alice'
```

RESTful Services
Quirks with REST Operations

**REST does not allow multiple arguments in the body without a “wrapper”**
- Since we have an operation with multiple arguments we could:
  1. change to a single argument using a data contract...

```csharp
public class Numbers {
    public int A { get; set; }
    public int B { get; set; }
}
```

- Note: this is good practice anyway since it allows versioning
- ...or (2) apply [WebGet] since the types are simple and could therefore be passed in a query string

```csharp
[OperationContract]
[WebGet]
public int AddNumbers(Numbers n);
```

```
http://localhost:801/Sample/kermit/AddNumbers?a=4&b=5
```
RESTful Services

Return Content Type Formatting

- Automatic formatting (off by default)
  - Request message’s Accept header
  - Request message’s Content-type
  - Default format setting for operation and WebHttpBehavior

- To force an operation to return JSON

  ```csharp
  [OperationContract]
  [WebGet(ResponseFormat = WebMessageFormat.Json)]
  public DateTime GetLunchTime();
  ```

- Body style (bare or wrapped)
  - Bare only allows one parameter; Wrapped supports multiple

  ```csharp
  [OperationContract]
  [WebInvoke(BodyStyle = WebMessageBodyStyle.Wrapped)]
  ```

Wrapped BodyStyle in WCF Rest

RESTful Services

URI Templates

- To customize the URL path use a UriTemplate
  - Path variables must use string

  ```csharp
  [OperationContract]
  [WebGet(UriTemplate = "weather/{state}/{city}")]  
  string GetWeather(string state, string city);
  ```

  ```csharp
  [OperationContract]
  [WebInvoke(UriTemplate = "order/{id}", Method = "DELETE")]
  void DeleteOrder(string id);
  ```

- To pass non-strings, you must use a query string

  ```csharp
  [OperationContract]
  [WebGet(UriTemplate = "add?a={a}&b={b}" )]
  int AddNumbers(int a, int b);
  ```
Endpoints

Define separate endpoints for different transports

Scenario

- You already have a service with a HTTP endpoint

Requirement

- Expose the service over TCP

Solution

- Leave the existing endpoint as-is and define a new endpoint for TCP

```xml
<service name="Firebrand.SampleService">
  <endpoint address="net.tcp://.../
  binding="netTcpBinding"
  contract="Firebrand.ISampleService" />
  <endpoint address="http://.../
  binding="wsHttpBinding"
  contract="Firebrand.ISampleService" />
</service>
```

Endpoints

ClientVia

A service can use a logical address while using a different address and binding for the transport

```xml
<client>
  <endpoint address="http://www.aw.com/Calc"
  binding="netTcpBinding"
  behaviorConfiguration="viaBehavior"
  contract="ServiceReference.ICalc" />
</client>
```

```xml
<endpointBehaviours>
  <behaviour name="viaBehavior">
    <clientVia viaUri="net.tcp://www.aw.com/ICalc" />
  </behaviour>
</endpointBehaviours>
```

// or use code
client.Endpoint.Behaviors.Add(new ClientViaBehavior(
    new Uri("net.tcp://www.aw.com/ICalc"));

ClientViaBehavior Class
Endpoints

Why Use ClientVia?

Facilitates debugging scenarios

- Normally the client sends the message to the service directly and to ensure against attacks the service checks the requesting address with the actual address from which the request was sent.
- However, this causes problems if you have intermediate services that might be used for debugging or other reasons because the service will receive the message from the intermediary.
- If you set the viaUri to the address of the intermediary, then the service won't reject the forwarded messages.

For another example see link below

WCF Intermediate Service between Client and Server

Endpoints

Standard Endpoints

Standard endpoints enable a developer to define an endpoint that has default values or where one or more endpoint’s properties do not change.

- For example, to enable cross-domain script access for all instances of a web script endpoint:

```xml
<system.serviceModel>
  <serviceHostingEnvironment aspNetCompatibilityEnabled="true" />
  <standardEndpoints>
    <webScriptEndpoint>
      <standardEndpoint crossDomainScriptAccessEnabled="true" />
    </webScriptEndpoint>
  </standardEndpoints>
</system.serviceModel>
```

Standard Endpoints
Behaviors
Applying Named and Default Behaviors

To define a default behavior that automatically applies to ALL services in this host leave its name empty:

```xml
<behaviors>
  <serviceBehaviors>
    <behavior><!-- or <behavior name=""> -->
      <serviceThrottling maxConcurrentCalls="100"
      maxConcurrentSessions="50" />
    </behavior>
  </serviceBehaviors>
</behaviors>
```

- Unnamed behaviors are combined with one named behavior that can be set explicitly on each service.

If a behavior has a name you must explicitly apply it:

- Even if its name is Default!

```xml
<service behaviorConfiguration="Default">

(proxy) .Endpoint.Behaviors.Add(...);
```

Behaviors
How to Apply Behaviors

Behaviors can be applied to many parts of WCF:

- **Service and endpoint behaviors can be applied in `*.config`**

```xml
<behaviors>
  <serviceBehaviors> ...
  <endpointBehaviors> ...
</behaviors>
```

- **Some behaviors can be applied in code using attributes**

```csharp
[ServiceBehavior(...)]
public class SampleService : ISampleService
{
  [OperationBehavior(...)]
  public int AddNumbers(...)
}
```

- **All behaviors can be applied by using procedural code**

`proxy.Endpoint.Behaviors.Add(...);`
Behaviors
ConcurrentMode Service Behavior

Specifies whether a service class supports single-threaded or multi-threaded modes of operation

Single (default)
- Service instance is single-threaded and does not accept reentrant calls

Multiple
- Service instance is multi-threaded

Reentrant
- Service instance is single-threaded and accepts reentrant calls when you call another service
- It is your responsibility to leave your object state consistent before callouts and you must confirm that operation-local data is valid after callouts

Behaviors
InstanceContextMode Service Behavior

PerSession (default)
- A new InstanceContext object is created for each session

PerCall
- A new InstanceContext object is created prior to and recycled subsequent to each call

Single
- Only one InstanceContext object is used for all incoming calls and is not recycled subsequent to the calls
- If a service object is not passed to the ServiceHost constructor, one (and only one) is created automatically on first call
- Your service can only process one message at a time unless you also set the ConcurrentMode value to Multiple

InstanceContextMode Enumeration
Behaviors
Instancing and Concurrency Matrix

<table>
<thead>
<tr>
<th>Concurrency</th>
<th>Instancing</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single</td>
<td>x</td>
<td>n/a</td>
</tr>
<tr>
<td>Reentrant</td>
<td>x</td>
<td>n/a</td>
</tr>
<tr>
<td>Multiple</td>
<td>✓</td>
<td>n/a</td>
</tr>
</tbody>
</table>

One thread, so create an instance per session (default) or per call
One thread, but if it calls out to a service it does not block an incoming call
Shared state must be thread-safe

If Instancing is PerCall, Concurrency is ignored because only one thread will ever be created.

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>A service uses InstanceContextMode .Single. How can you increase the rate by which clients get the response?</td>
<td>Two possible solutions: ConcurrencyMode.Multiple or change the InstanceContextMode</td>
</tr>
<tr>
<td>An operation in a service makes a call to another service and does not perform well.</td>
<td>ConcurrencyMode.Reentrant on ServiceBehavior (not CallbackBehavior)</td>
</tr>
</tbody>
</table>

Behaviors
Stateful Services with Many Clients

🌟To “service many clients and requests simultaneously” and to “share state for subsequent individual client requests”, use multiple concurrency mode and enable per session instancing

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

🌟Since PerSession is the default anyway, we could use

```csharp
[ServiceBehavior( ConcurrencyMode = ConcurrencyMode.Multiple)]
```

🌟To share state between clients use

- InstanceContextMode.Single

Sessions, Instancing, and Concurrency
**Behaviors**  
**Publishing Metadata**

- **HTTP(S) GET metadata “endpoints”** are easily enabled and use the base address

```xml
<serviceBehaviors>
  <behavior>
    <!-- and httpsGetEnabled -->
    <serviceMetadata httpGetEnabled="true" />
  </behavior>
</serviceBehaviors>
```

- **For other metadata endpoints use WS-MetadataExchange**

```xml
<serviceBehaviors>
  <behavior>
    <serviceMetadata />
  </behavior>
</serviceBehaviors>
```

Do NOT specify the full namespace i.e. NOT System.ServiceModel.Description

- `mexHttpsBinding`, `mexNamedPipeBinding`, `mexTcpBinding`

---

**Behaviors**  
**Publishing Custom Metadata**

- You can specify an alternative location for returning the metadata instead of it being automatically generated

```xml
<serviceMetadata httpGetEnabled="true"
  externalMetadataLocation="MyCustomWsdl.wsdl" />
```

- To return custom WSDL for different scenarios, create a service and operation, and reference that instead

```csharp
[ServiceContract]
public interface ICustomWsdl
{
    [OperationContract]
    [WebGet]
    Stream Metadata(string name, string ext);
}
```

```
externalMetadataLocation="http://.../CustomWsdl/Metadata?name=BBC&ext=wsdl"
```
**5.1 Sessions**

**What Is a Session?**

- Correlates a group of messages into a conversation
  - Do NOT confuse with Session state in ASP.NET
- Session can be provided by the transport protocol (e.g. TCP) or by some other mechanism (e.g. WS-*)
- Although PerSession is the default instance mode, a session is only created if the binding supports sessions
  - NO support for sessions: BasicHttpBinding and WebHttpBinding
- SessionId can be read from proxy’s InnerChannel

**5.2 Sessions**

**Initiating and Terminating Operations**

- Operations can be marked so they either initiate a session or terminate a session
  - Service must require sessions
  - By default, IsInitiating=true and IsTerminating=false

```csharp
[ServiceContract(SessionMode = SessionMode.Required)]
public interface IMyService
{
    [OperationContract]
    void CreateShoppingCart();
    [OperationContract(IsInitiating = false)]
    void AddItemToCart();
    [OperationContract(IsInitiating = false, IsTerminating = true)]
    void CheckOut();
}
```
Summaries
Data Contracts

Member Opt-In

```csharp
[DataContract(...)]
public class Person
{
    [DataMember(...)]
    public int PersonID { get; set; }
    // Salary not serialized
    public decimal Salary { get; set; }
}
```

Member Opt-Out (3.5 SP1 or later)

```csharp
public class Person
{
    public int PersonID { get; set; }
    // Salary not serialized
    [IgnoreDataMember]
    public decimal Salary { get; set; }
}
```

To move members to header or to protect members

```csharp
[MessageContract(...)]
public class Person
{
    [MessageHeader] public int PersonID { get; set; }
    [MessageBodyMember] public decimal Salary { get; set; }
    [MessageBodyMember(ProtectionLevel = ProtectionLevel.EncryptAndSign)]
    public string CreditCardNumber { get; set; }
}
```

Summaries
Service Contracts and Implementation

On Interface/Class (Contract)

```csharp
[ServiceContract(...)]
public interface IInterfaceName
{
    [OperationContract(...)]
    [FaultContract(...)]
    [TransactionFlow(...)]
    [WebGet(...)]
    void Method(...);
}
```

On Class (Implementation)

```csharp
[ServiceBehavior(...)]
public class ClassName : IInterfaceName
{
    [OperationContract(...)]
    [FaultContract(...)]
    [TransactionFlow(...)]
    [WebGet(...)]
    void Method(...);
}
```

See the next slide for the details of the attributes for services and operations

Many exam questions require you to know what WCF does automatically and the defaults for attributes and configuration.
Summaries
Service Contracts and Implementation Parameters

<table>
<thead>
<tr>
<th>On Interface (Contract)</th>
<th>On Class (Implementation)</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1" alt="Code Snippet" /></td>
<td><img src="image2" alt="Code Snippet" /></td>
</tr>
</tbody>
</table>

5.56
Summaries
System.Runtime.Serialization Assembly

**Namespace:** System.Runtime.Serialization

- [DataContract]
- [DataMember]
- [IgnoreDataMember]
- [EnumMember]
- [KnownType]
- ExtensionDataObject
- IExtensibleDataObject
-DataContractSerializer

**Namespace:** System.Runtime.Serialization.Json

DataContractJsonSerializer
### System.ServiceModel Assembly (1/2)

**Namespace:** System.ServiceModel

- [MessageContract], [MessageHeader], [MessageBodyMember]
- [ServiceContract], [ServiceKnownType], [OperationContract], [FaultContract]
- [ServiceBehavior], [OperationContract]

- BasicHttpBinding, WSHttpBinding, NetTcpBinding, NetMsmqBinding, and so on
- ChannelFactory<T>, DuplexChannelFactory<T>
- ClientBase<T>, DuplexClientBase<T>
- EndpointAddress, EndpointAddress10 (Serializable)
- FaultException<T>, ExceptionDetail, FaultReason, FaultReasonText
- ICommunicationObject
- InstanceContext, OperationContext, SecurityContext
- ServiceHost, ServiceHostBase

### System.ServiceModel Assembly (2/2)

**Namespace:** System.ServiceModel.Activation

-AspNetCompatibilityRequirementsAttribute
-ServiceHostFactory

**Namespace:** System.ServiceModel.Channels

- Message
- MessageBuffer, MessageFault
- MessageHeader, MessageVersion

**Namespace:** System.ServiceModel.Description

- IMetadataExchange
- IEndpointBehavior, IOperationContract
- IServiceBehavior, and so on

**Namespace:** System.ServiceModel.Dispatcher

- ClientOperation, ClientRuntime
- DispatchOperation, DispatchRuntime
- IErrorHandler
- IClientMessageInspector, and so on
Summaries
WCF Configuration Schema

Windows Communication Foundation Configuration Schema
Module 6
Hosting Services
Developing Windows Azure and Web Services

Updated 11\textsuperscript{th} April 2014

### Hosting Services

#### Contents

<table>
<thead>
<tr>
<th>Topic</th>
<th>Slide</th>
</tr>
</thead>
<tbody>
<tr>
<td>WCF Hosting</td>
<td>3</td>
</tr>
<tr>
<td>WCF Hosting in IIS and/or WAS</td>
<td>6</td>
</tr>
<tr>
<td>WCF Hosting in Windows Apps</td>
<td>8</td>
</tr>
<tr>
<td>ASP.NET Compatibility Mode</td>
<td>9</td>
</tr>
<tr>
<td>Syndication (RSS)</td>
<td>11</td>
</tr>
<tr>
<td>Web API Hosting</td>
<td>15</td>
</tr>
</tbody>
</table>

**Exam Topics: Host and manage services**
- Create service hosts
- Choose a hosting mechanism
- Activate and manage a service by using AppFabric
- Host services in an Windows Azure worker role

**Exam Topics: Host and manage Web API**
- Host Web API in an ASP.NET app
- Self-host a Web API in your own process (a Windows service)
- Host services in a Windows Azure worker role
- Restricting message size
- Configure the host server for streaming
### WCF Hosting

#### 6.3 Choosing a Host

<table>
<thead>
<tr>
<th>Host/Transport</th>
<th>HTTP</th>
<th>Others</th>
</tr>
</thead>
<tbody>
<tr>
<td>IIS 6</td>
<td>✓</td>
<td>x</td>
</tr>
<tr>
<td>IIS 7+ only</td>
<td>✓</td>
<td>x</td>
</tr>
<tr>
<td>Windows Process Activation Service (WAS) only</td>
<td>x</td>
<td>✓</td>
</tr>
<tr>
<td>IIS 7+ and WAS</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Windows Service, Console, Windows application, and so on</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

### WCF Hosting

#### 6.4 How to Instantiate a ServiceHost

*If you self-host a service, pass the type of *class name* of the service implementation into the constructor of ServiceHost*

```csharp
host = new ServiceHost(typeof(CalcService));
```

*For PerSession (default) and PerCall instance context modes the host will then automatically instantiate an instance whenever required*
WCF Hosting
How to Instantiate a Singleton ServiceHost

If you only want a single instance (aka singleton) you must use Single instancing mode

```csharp
[ServiceBehavior(InstanceContextMode = InstanceContextMode.Single)]
public class CalcService
```

You can leave the host to create the instance automatically on first client call or you can passing an instance into the constructor of ServiceHost

```csharp
var calc = new CalcService();
host = new ServiceHost(calc);
```

- An exception will be thrown if you do this without setting Single instance mode

WCF Hosting in IIS and/or WAS
Activating with .svc file

Represented by .svc files

```csharp
using System.ServiceModel;
using System.ServiceModel.Activation;
```

```xml
<%@ ServiceHost Service="Firebrand.CalcService" Factory="Firebrand.CalcHostFactory" %>
```

- To dynamically customize the service host create a class that inherits from ServiceHostFactory and specify it as the factory

```csharp
public class CalcHostFactory : ServiceHostFactory

protected override ServiceHost CreateServiceHost(
    Type serviceType, Uri[] baseAddresses)
```

- You do not specify a base address in .config because IIS uses the URL of the .svc file as the base address

- Deploy the service implementation DLL to bin folder
WCF Hosting in IIS and/or WAS
Activating with Configuration

★ You can activate services without a physical file

```
<serviceHostingEnvironment>
  <serviceActivations>
    <add relativeAddress="Fake.svc"
         service="Firebrand.CalcService"
         factory="optional factory class" />
  </serviceActivations>
</serviceHostingEnvironment>
```

★ File extensions supported include .svc, .xoml

• To use others you can register them (but you don’t need to know how)

```
<serviceActivations>
```

WCF Hosting in Windows Applications
UI Thread Problems

★ When using a Windows application to host a service you must ensure that the UI thread does not block the service

```
[ServiceBehavior(UseSynchronizationContext = false)]
```

★ Default is true which causes deadlock problems if the ServiceHost was created on the main thread and you then call the service from the same thread

```
Synchronization Contexts in WCF
```
ASP.NET Compatibility Mode

What Is It?

In compatibility mode, WCF services use the HTTP pipeline through an IHttpHandler implementation so WCF behaves identically to ASMX with respect to the following ASP.NET features (therefore must use HTTP):

- HttpContext: can access Current and its associated state
- File-based authorization: can be secure by attaching file system access control lists (ACLs) to the service’s .svc file
- Configurable URL authorization: ASP.NET’s URL authorization rules are enforced for WCF requests
- HttpModuleCollection extensibility: any HTTP module configured in the HTTP pipeline is able to operate on WCF requests both before and after service invocation
- ASP.NET Impersonation: WCF services run using the current identity of the ASP.NET impersonated thread

ASP.NET Compatibility Mode

How to Use It

Enable compatibility mode in .config

```xml
<system.serviceModel>
  <serviceHostingEnvironment aspNetCompatibilityEnabled="true" />
</system.serviceModel>
```

Service implementations can control if they run if compatibility mode is enabled

```csharp
using System.ServiceModel.Activation;

[AspNetCompatibilityRequirements(RequirementsMode = AspNetCompatibilityRequirementsMode.Required)]
public class CalculatorService : ICalculator
```

- Required, Allowed, NotAllowed

An activation error occurs when the service receives a message if the options conflict

- Causes exception: true+NotAllowed or false+Required
 Syndication Support  
System.ServiceModel.Syndication namespace

• Publish and consume RSS and Atom syndication feeds
  • Reference System.ServiceModel.Web.dll for .NET 3.5

• Classes
  • Atom10FeedFormatter: (De)serializes a feed in Atom 1.0 format
  • Rss20FeedFormatter: (De)serializes a feed in RSS 2.0 format
  • SyndicationFeed: a top-level feed object
  • SyndicationItem: a feed item
  • TextSyndicationContent: content intended to be displayed to an end user
  • UrlSyndicationContent: content that consists of a URL to another resource

 Syndication Support  
Publish Example (1/2)

using System.ServiceModel.Syndication;

• Feed contract

```csharp
public interface INewsFeed {
    [OperationContract]
    [WebGet(UriTemplate = "GetNews?format={format}")]    SyndicationFeedFormatter GetNews(string format);
}
```

```
```

• Feed implementation

```csharp
public SyndicationFeedFormatter GetNews(string format)
{
    var feed = new SyndicationFeed(...);
    feed.Authors.Add(new SyndicationPerson("me@gmail.com"));
    feed.Categories.Add(new SyndicationCategory("Tech News"));
}
```
Create a new list

```csharp
var items = new List<SyndicationItem>();
```

Add one (of many) items

```csharp
var textContent = new TextSyndicationContent("Some text content ..."TOTYPE="html:"); var item1 = new SyndicationItem("Item Title", textContent, new Uri("http://news.bbc.co.uk"), System.Guid.NewGuid().ToString(), DateTime.Now); items.Add(item1);
```

Return the feed in the correct format

```csharp
if (format == "rss")
    return new Rss20FeedFormatter(feed);
else if (format == "atom")
    return new Atom10FeedFormatter(feed);
```

Load the feed

```csharp
var atomReader = XmlReader.Create("http://.../NewsFeedService/GetNews?format=atom"); var feed = SyndicationFeed.Load(atomReader);
```

Enumerate the items

```csharp
foreach (SyndicationItem item in feed.Items)
{
    Debug.Print(item.Title.Type);
    Debug.Print(item.Title.Text);
    Debug.Print(item.Summary.Text);
    // ...
}
```

TextSyndicationContent can contain HTML, XHTML, or plain text indicated by Type being "html", and so on
Web API Hosting
Self-Host Web API

• ASP.NET Web API does not require IIS
• You can self-host a web API in your own host process

Self-Host a Web API
http://www.asp.net/web-api/overview/hosting-aspnet-web-api/self-host-a-web-api

Web API Hosting
Host Web API in Windows Azure

• Open Web Interface for .NET defines an abstraction between .NET web servers and web applications
  - OWIN decouples the web application from the server, which makes OWIN ideal for self-hosting a web application in your own process, outside of IIS—for example, inside an Azure worker role

Host ASP.NET Web API in an Azure Worker Role
http://www.asp.net/web-api/overview/hosting-aspnet-web-api/host-aspnet-web-api-in-an-azure-worker-role
Module 7
Windows Azure Service Bus
Developing Windows Azure and Web Services

Exam Topics: Create and configure a WCF service on Windows Azure
- Create and configure bindings for WCF services (Azure SDK extensions to WCF)
- Relay bindings to Azure using service bus endpoints
- Integrate with the Azure service bus relay

Exam Topics: Implement messaging patterns
- Implement Windows Azure Service Bus and Windows Azure Queues

How to Use the Service Bus Relay Service
Replace *** with your registered namespace

```xml
<services>
  <service name="Service.ProblemSolver">
    <endpoint contract="Service.IProblemSolver"
      binding="netTcpRelayBinding"
      address="sb://***.servicebus.windows.net/solver"
      behaviorConfiguration="sbTokenProvider"/>
  </service>
</services>
```

Use your key provider for the issuer name

```xml
<behaviors>
  <endpointBehaviors>
    <behavior name="sbTokenProvider">
      <transportClientEndpointBehavior>
        <tokenProvider>
          <sharedSecret issuerName="owner"
            issuerSecret="***key***"/>
        </tokenProvider>
      </transportClientEndpointBehavior>
    </behavior>
  </endpointBehaviors>
</behaviors>
```

Securing and authenticating azure service bus relay messages using a shared secret

http://acaseyblog.wordpress.com/2013/03/22/securing-and-authenticating-azure-service-bus-relay-messages-using-a-shared-secret/
Module 8
Deploying Services
Developing Windows Azure and Web Services

Updated 11th April 2014

Deploying Services
Contents (1 of 2)

<table>
<thead>
<tr>
<th>Topic</th>
<th>Slide</th>
</tr>
</thead>
<tbody>
<tr>
<td>Designing for Deployment</td>
<td>4</td>
</tr>
<tr>
<td>Deployment Tools</td>
<td>5</td>
</tr>
<tr>
<td>Windows Azure</td>
<td>6</td>
</tr>
<tr>
<td>NuGet</td>
<td>8</td>
</tr>
</tbody>
</table>

Exam Topics: Design a deployment strategy
- Create an IIS install package
- Deploy to web farms
- Deploy a web application by using Xcopy
- Automate a deployment from TFS or Build Server

Exam Topics: Configure a web application for deployment
- Switch from production/release mode to debug mode
- Use SetParameters to set up an IIS app pool, set permissions and passwords
- Configure WCF endpoints, bindings, and behaviors
- Transform web.config by using XSLT (for example, across development, test, and production/release environments)
- Configure Azure configuration settings

Exam Topics: Choose a deployment strategy for a Windows Azure web application
- Perform an in-place upgrade and VIP swap
- Configure an upgrade domain
- Create and configure input and internal endpoints
- Specify operating system configuration
Deploying Services
Contents (2 of 2)

Exam Topics: Manage packages by using NuGet
- Create and configure a NuGet package
- Install and update an existing NuGet package
- Connect to a local repository cache for NuGet, set up your own package repository

Exam Topics: Create, configure, and publish a web package
- Create an IIS InstallPackage
- Configure the build process to output a web package
- Apply pre- and post-condition actions to ensure that transformations are correctly applied
- Include appropriate assets (web content, certificates)

Exam Topics: Share assemblies between multiple applications and servers
- Prepare the environment for use of assemblies across multiple servers (interning)
- Sign assemblies by using a strong name
- Deploy assemblies to the global assembly cache
- Implement assembly versioning
- Create an assembly manifest
- Configure assembly binding redirects (for example, from MVC2 to MVC3)

Designing for Deployment
Assembly Libraries

 Giul If you want to use an assembly in multiple web applications and services, deploy to the GAC
 Giul If you deploy a private assembly and then release a new version, you might need to force a bindingRedirect

<configuration>
<runtime>
<assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">
<dependentAssembly>
<assemblyIdentity name="myAssembly"
    publicKeyToken="32ab4ba45e0a69a1"
    culture="neutral"/>
<bindingRedirect oldVersion="1.0.0.0"
    newVersion="2.0.0.0"/>
</dependentAssembly>
</assemblyBinding>
</runtime>
</configuration>
Deployment Tools
Web Deploy

For any question about deployment tools, the answer is almost always use Web Deploy because

• It works securely
• It is powerful and flexible by changing the web publish pipeline
• You can install SSL certificates using a custom target

Only choose to use FTP, XCopy, VPN, SSH, and so on if you have a very good reason

Windows Azure
Swapping Staging and Production

A packaged application can be deployed to the staging environment in Windows Azure to be tested before you move it to the production environment in which the application is accessible on the Internet

The staging environment is exactly like the production environment, except that you can only access the staged application with an obfuscated (GUID-based) URL that is generated by Windows Azure

After you have verified that your application is working correctly, it can be deployed to the production environment by performing a Virtual IP (VIP) swap
The Windows Azure service definition file

- Contains the definitions for the roles available to a service,
  specifies the service endpoints, and establishes configuration

```xml
<ServiceDefinition ...
  <WebRole name="web-role-name" ...
  <Endpoints>
    <InputEndpoint name="endpoint-name" protocol="HTTP" port="80" ...>
    <InternalEndpoint ...
    <InstanceInputEndpoint ...

- InputEndpoint: an endpoint to a role from the external world
- InternalEndpoint: available only to other role instances running within the service
- InstanceInputEndpoint: associated with a specific role instance by using port forwarding in the load balancer

WebRole Schema

NuGet
What Is It?

- NuGet is a Visual Studio extension that makes it easy to install and update third-party libraries and tools
- Choose “Library Package Manager” from the Tools menu
Hosting Your Own NuGet Feeds

You can configure Visual Studio to offer that feed instead of or in addition to the official feed.

- A feed can be local (a folder on the local machine or a network folder) or remote (an internet URL).

A local feed

Creating Remote Feeds

Install the NuGet.Server Package into a new ASP.NET Empty Application project.

- The NuGet.Server package just converted your empty website into a site that's ready to serve up the OData package feed.

Configure the Packages folder

```xml
<appSettings>
    <!-- Set the value here to specify your custom packages folder. -->
    <add key="packagesPath" value="C:\MyPackages" />
</appSettings>
```

- Just add packages into the Packages folder and they'll show up.
Module 9
Windows Azure Storage
Developing Windows Azure and Web Services

Updated 11th April 2014

Windows Azure Storage
Contents

Exam Topic: Implement data storage in Windows Azure
- Access data storage in Windows Azure
- Choose data storage mechanism in Windows Azure (blobs, tables, queues, SQL Database)
- Distribute data by using the Content delivery network (CDN)
- Handle exceptions by using retries (SQL Database)
- Manage Windows Azure Caching

Developing Cloud Applications With Windows Azure Storage
http://www.amazon.co.uk/Developing-Cloud-Applications-Windows-Storage/dp/0735667985/
Windows Azure Storage

Your Choices

Windows Azure storage (100TB per account)
- Table: each structured entry (up to 1MB), table (unlimited)
- Blob: block (up to 200GB) or page (up to 1TB) e.g. videos
- Queue: message exchange (up to 64KB)
- Drive: permanent files in separate VM defined by a VHD
- Local: temporary files in a role VM (could be lost at any time)

Why use Windows Azure storage?
- It is comparatively cheap
- BUT you have other choices, for example, instead of WA Queue
  - Service Bus queues, MSMQ, Google App Engine queues, etc.

SQL Database aka “SQL Server in the cloud”
- Three replicated copies of your database with auto-failover

Table Storage Design

PartitionKey
- Tables are partitioned to support load balancing
- A partition is a consecutive range of entities possessing the same partition key value
- The partition key forms the first part of an entity's primary key
- The partition key may be a string value up to 1 KB in size

RowKey
- The second part of the primary key is the row key
- A unique identifier for an entity within a given partition
- Together the PartitionKey and RowKey uniquely identify every entity within a table
- The row key is a string value that may be up to 1 KB in size

Designing a Scalable Partitioning Strategy for Windows Azure Table Storage
Entities must have three properties

- PartitionKey, RowKey: strings of up to 1024 bytes
- TimeStamp: DateTime (can be auto-created but then you won’t be able to get or set the value assigned)

Entities can only use the following types

- string, int, long, DateTime, byte[], bool, double, Guid
- You cannot use your own subtypes
Module 10
Monitoring and Diagnostics
Developing Windows Azure and Web Services

Updated 11th April 2014

Monitoring and Diagnostics
Contents

<table>
<thead>
<tr>
<th>Topic</th>
<th>Slide</th>
</tr>
</thead>
<tbody>
<tr>
<td>Troubleshooting</td>
<td>3</td>
</tr>
<tr>
<td>Tracing</td>
<td>4</td>
</tr>
<tr>
<td>Performance Counters</td>
<td>9</td>
</tr>
<tr>
<td>Stopwatch</td>
<td>12</td>
</tr>
<tr>
<td>Debugging</td>
<td>13</td>
</tr>
<tr>
<td>Error Handlers</td>
<td>14</td>
</tr>
<tr>
<td>Throttling</td>
<td>15</td>
</tr>
</tbody>
</table>

Exam Topics: None
10.3 Tracing
WCF Milestones

❖ System.ServiceModel trace source

• General WCF trace source records processing milestones across
  the WCF communication stack, from entering/leaving transport
to entering/leaving user code

```xml
<system.diagnostics>
  <sources>
    <source name="System.ServiceModel">
      <listeners>
        <add name="traces" type="System.Diagnostics.XmlWriterTraceListener"
             initializeData="c:\logs\traces.svclog" />
    </source>
  </sources>
</system.diagnostics>
```

❖ To correlate traces across tiers

```xml
<source name="System.ServiceModel" propagateActivity="true"
         switchValue="Warning, ActivityTracing"/>
```

Configuring Tracing

10.4 Tracing
Logging Messages

❖ To configure a service to log messages from a client

```xml
<system.diagnostics>
  <sources>
    <source name="System.ServiceModel.MessageLogging">
      <listeners>
        <add name="messages" type="System.Diagnostics.XmlWriterTraceListener"
             initializeData="c:\logs\messages.svclog" />
    </source>
  </sources>
</system.diagnostics>
```

```xml
<system.serviceModel>
  <diagnostics>
    <messageLogging logEntireMessage="true"
                    logMessagesAtServiceLevel="true"
                    logMessagesAtTransportLevel="false"/>
  </diagnostics>
</system.serviceModel>
```

Configuring Message Logging
**10.5 Tracing**

**Messaging Logging Options**

- **logEntireMessage (default is false)**
  - Specifies if the entire message (header and body) is logged
  - Affects service, transport, and malformed logging levels

- **logMessagesAtServiceLevel (default is false)**
  - Specifies whether messages are traced at the service level (before encryption- and transport-related transforms)

- **logMessagesAtTransportLevel (default is false)**
  - Specifies whether messages are traced at the transport level
  - Any filters specified in the config file are applied, and only messages that match the filters are traced
  - The filters element holds a collection of XPath filters

```xml
<messageLogging>
</messageLogging>
```

---

**10.6 Tracing**

**Logging Security Information with Messages**

- To include security tokens aka personally identifiable information (PII) in logged messages
  - Machine.config
    ```xml
    <machineSettings enableLoggingKnownPii="true" ...>
    </machineSettings>
    ```
  - App.config or Web.config
    ```xml
    <system.serviceModel>
    <diagnostics>
    <messageLogging logKnownPii="true" ...>
    </messageLogging>
    </diagnostics>
    </system.serviceModel>
    ```
**Performance Counters**

**How to Enable WCF Performance Counters**

To enable all performance counters exposed by the `ServiceModelService` counter group:

```xml
<diagnostics performanceCounters="ServiceOnly"/>
```

**Options**

- **All**: `ServiceModelService`, `ServiceModelEndpoint` and `ServiceModelOperation`
- **ServiceOnly**: `ServiceModelService`
- **None**

**To read the counters**

- `ServiceModelX 4.0.0.0\CounterName` e.g. `Calls`, `Instances`
- Where `X` is either Service, Endpoint, or Operation

WCF Performance Counters


---

**Performance Counters**

**Performance Counters in PerfMon.exe**

![Image of PerfMon.exe with selected counters](image)
Performance Counters
Performance Counters in Code

To programatically read a performance counter

```csharp
using System.Diagnostics;
using System.Reflection;

string category = "\".NET Data Provider for SqlServer\"";
string counter = "NumberOfReclaimedConnections";
string instance = string.Format("\{0\}[[{1}\"]",
    Assembly.GetEntryAssembly().GetName().Name,
    Process.GetCurrentProcess().Id);
var pc = new PerformanceCounter(category, counter, instance);
```

Stopwatch
Using the Stopwatch class

Two ways to create and use

```csharp
var s = Stopwatch.StartNew();
// do some work
s.Stop();
TimeSpan ts = s.Elapsed;
long ms = s.ElapsedMilliseconds;

var s = new Stopwatch();
s.Start();
```

Stop method

- Stop and retain cumulative elapsed time

Reset method

- Stop and reset the elapsed time to zero

Restart method is equivalent to: Reset then Start
Error Handlers
Processing Messages

The body of a Message instance can only be accessed or written once.

If you have an operation that accepts the Message type and you need to process it multiple times you must

- Call the CreateBufferedCopy method to load into memory
- Call the CreateMessage method of the MessageBuffer

Throttling
MaxReceivedMessageSize

Gets and sets the maximum allowable message size, measured in bytes, that can be received

- Default is 65536 bytes (64 KB)
- For example, you might want to increase the maximum size of a message that a service can receive to 256 KB

```csharp
<wsHttpBinding>
  <binding maxReceivedMessageSize="262144"/>

var binding = new WSHttpBinding();
binding.MaxReceivedMessageSize = 256 * 1024;
```
Module 12
Scaling Services
Developing Windows Azure and Web Services

Updated 11\textsuperscript{th} April 2014

Scaling Services
Contents

<table>
<thead>
<tr>
<th>Topic</th>
<th>Slide</th>
</tr>
</thead>
<tbody>
<tr>
<td>System.Runtime.Caching</td>
<td>3</td>
</tr>
<tr>
<td>Real-Time Synchronization</td>
<td>4</td>
</tr>
<tr>
<td>Transient Fault Handling</td>
<td>7</td>
</tr>
</tbody>
</table>

Exam Topics: Implement caching
- Cache static data, apply cache policy (including expirations)
- Use CacheDependency to refresh cache data
- Query notifications
12.3

System.Runtime.Caching
CacheItemPolicy

★ Represents a set of eviction and expiration details
  • Some methods in the MemoryCache (inherits from ObjectCache) classes accept a CacheItemPolicy instance

```csharp
ObjectCache cache = MemoryCache.Default;
string fileContents = cache["filecontents"].ToString();
if (fileContents == null)
{
    CacheItemPolicy policy = new CacheItemPolicy();
    policy.AbsoluteExpiration = DateTimeOffset.Now.AddSeconds(60.0);
    policy.ChangeMonitors.Add(new HostFileChangeMonitor(new List<string> { root + "\cache.txt" }));
    fileContents = File.ReadAllText(root + "\cache.txt");
    cache.Set("filecontents", fileContents, policy);
}
```

CacheItemPolicy Class

12.4

Real-Time Synchronization
Query Notifications

★ Built upon the Service Broker infrastructure
  • SQL Server 2005 or later

★ System.Data.SqlClient.SqlDependency
  • High-level implementation; least developer effort
  • Designed for small number of middle-tier objects; you should limit the number of listeners

★ System.Data.SqlClient.SqlNotificationRequest
  • Low-level implementation; requires you to implement the entire listening infrastructure yourself
  • Can receive messages even if not running at time of notification

Query Notifications in SQL Server (ADO.NET)
Real-Time Synchronization

Enabling Query Notifications

- Enable Service Broker on database
  ```
  ALTER DATABASE AdventureWorks SET ENABLE_BROKER;
  ```

- Grant the user rights to subscribe to notifications
  ```
  GRANT SUBSCRIBE QUERY NOTIFICATIONS TO Fred;
  ```

- Start listening
  ```
  using System.Data.SqlClient;
  SqlDependency.Start(connectionString);
  ```

- Add command dependency (cannot use SELECT * ...)
  ```
  var cmd = new SqlCommand(
    "SELECT CustomerID, CompanyName FROM dbo.Customers");
  var dep = new SqlDependency(cmd);
  ```
  - Must also use two-part names for objects

Real-Time Synchronization

Handling Query Notifications

- HandleOnChange event
  ```
  dep.OnChange += dep_OnChange;
  ```
  ```
  private void dep_OnChange(object sender, SqlNotificationsEventArgs e) {
    // e.Info, e.Source, e.Type
  }
  ```
  - Or poll the HasChanges property

- Stop listening when done
  ```
  SqlDependency.Stop(connectionString);
  ```
 transient fault handling

what is it?

- when you’re designing a real world cloud app, one of the things you have to think about is how to handle temporary service interruptions
  - you can frequently get little glitches that are typically self-healing, and if you aren’t prepared to handle them intelligently, they’ll result in a bad experience for your customers

- use smart retry/back-off logic to mitigate the effect of transient failures
  - instead of throwing an exception and displaying a not available or error page to your customer, you can recognize errors that are typically transient, and automatically retry the operation that resulted in the error, in hopes that before long you’ll be successful

implementing smart retry logic

- several ways you can implement smart retry logic
  - microsoft patterns & practices group has a transient fault handling application block that does everything for you if you’re using ado.net for sql database access (not through entity framework)

```csharp
var policy = RetryPolicy.CreateSqlAzureTransientErrorDetectionStrategy(
    retryCount: 3, retryInterval: TimeSpan.FromSeconds(5));
using (var conn = new ReliableSqlConnection(connStr, policy))

public class EFConfiguration : DbConfiguration
{
    public EFConfiguration()
    {
        AddExecutionStrategy(() => new SqlAzureExecutionStrategy());
    }
}
```

- entity framework 6 builds this kind of retry logic right into the framework

**Appendix A**

**Designing and Extending WCF Services**

Developing Windows Azure and Web Services

Updated 11th April 2014

---

### Contents

<table>
<thead>
<tr>
<th>Topic</th>
<th>Slide</th>
</tr>
</thead>
<tbody>
<tr>
<td>MEP Contracts</td>
<td>3</td>
</tr>
<tr>
<td>Asynchronous Operations</td>
<td>4</td>
</tr>
<tr>
<td>Extending WCF</td>
<td>7</td>
</tr>
<tr>
<td>Transactions</td>
<td>8</td>
</tr>
<tr>
<td>Discovery</td>
<td>13</td>
</tr>
<tr>
<td>Routing</td>
<td>17</td>
</tr>
</tbody>
</table>

**Exam Topics:** Implement messaging patterns
- Implement one way, request/reply, streaming, and duplex communication

**Exam Topics:** Configure WCF services by using configuration settings or the API
- WCF routing and discovery features

**Exam Topics:** Create a WCF service
- Implement message inspectors
- Implement asynchronous operations in the service

**Exam Topics:** Host and manage services
- Implement transactional services
A.3 MEP Contracts
One-Way Considerations

**With a one-way operation the receiver does not send a reply message there is no response message to carry fault information back to the client**
- You may be able to detect error conditions by using features of the underlying binding, such as reliable sessions

**Most one-way operations return as soon as the outbound data is written to the network connection**
- Operation call can block if ConcurrencyMode is Single and the binding uses sessions
- If the transport cannot find the endpoint a EndpointNotFoundException is thrown
- If the SendTimeout period on the client transport binding is exceeded a TimeoutException is thrown

A.4 Asynchronous Operations
Asynchronous Clients (WCF 3.0)

**To call a service asynchronously without affecting any other clients or changing the service, the proxy generated can implement the .NET APM**

```csharp
var client = new CalcServiceClient();
client.BeginAddNumbers(1, 1, OnComplete, client);

private void OnComplete(IAsyncResult result)
{
    var client = result.AsyncState as CalcServiceClient;
    int answer = client.EndAddNumbers(result);
}
```

**Use svcutil.exe with /a or /async switch to generate asynchronous methods like BeginX and EndX**
Asynchronous Operations
Asynchronous Clients (WCF 3.5)

Instead of calling BeginX and EndX methods, can set up a handler for the XCompleted event and then call XAsync method to start the call

```csharp
var client = new CalcServiceClient();
client.AddNumbersCompleted +=
    new EventHandler<AddNumbersCompletedEventArgs>(OnComplete);
client.AddNumbersAsync(1, 1);

private void OnComplete(object sender, AddNumbersCompletedEventArgs e)
{
    int answer = e.Result;
}
```

Event-based Asynchronous Pattern Overview

Asynchronous Operations
Asynchronous Clients (WCF 4.5)

Instead of handling the old XxxCompleted event and calling the old XxxAsync method, call the XxxAsync method that returns a Task<T>

```csharp
var client = new CalcServiceClient();
var task = client.AddNumbersAsync(1, 1);
// do independent work
var int = await task;
```
Extending WCF

“Dispatch” = server-side
• Implement these interfaces:
  IDispatchMessageFormatter,
  IDispatchMessageInspector,
  IDispatchOperationSelector
• Attach to these objects:
  • DispatchOperation
  • ParameterInspectors, Formatter
  • DispatchRuntime
  • MessageInspectors

“Client” = client-side
• Implement these interfaces:
  IClientMessageFormatter,
  IClientMessageInspector,
  IClientOperationSelector
• Attach to these objects:
  • ClientOperation
  • ParameterInspectors, Formatter
  • ClientRuntime
  • MessageInspectors

Extending WCF’s Client and Dispatcher Runtimes

Transactions
Enabling Transactions

For an operation to automatically participate in an existing transaction or to create a new transaction if one does not exist (default is false)

```
[OperationBehavior(TransactionScopeRequired=true)]
public bool ValidateCredit(string cardNumber)
```

For a service to support transactions it must also require sessions

```
[ServiceContract(SessionMode=SessionMode.Required)]
public interface IDataUpdate
```
Transactions
Service Behaviors for Transactions

To automatically destroy the service instance when the transaction completes (default is false)

```csharp
[ServiceBehavior(ReleaseServiceInstanceOnTransactionComplete=true)]
public class Calculator
```

- You must set at least one operation to require a transaction
- You must NOT set concurrency mode to multiple

To automatically complete the transaction when the session ends i.e. client closes proxy (default is false)

```csharp
[ServiceBehavior(TransactionAutoCompleteOnSessionClose=true)]
public class Calculator
```

Transactions
Operation Behaviors for Transactions

TransactionScopeRequired (default is false)
- Indicates whether the method requires a transaction scope for its execution
- Must be true on at least one operation if ReleaseServiceInstanceOnTransactionComplete is true

TransactionAutoComplete (default is true)
- Indicates whether to automatically complete the current transaction scope if no unhandled exceptions occur
- Set to false to complete or abort transactions directly in the code for the operation
- Handled exceptions can be thrown in the course of the operation without automatically aborting the transaction
- Unhandled exceptions trigger an automatic abort
A.11

Transaction Flow on Operation Contract

TransactionFlowOption

- Specifies whether a service operation accepts incoming transactions from a client, or if the operation requires the client to always flow a transaction
- NotAllowed (default): transaction should not be flowed
- Allowed: transaction may be flowed
- Mandatory: transaction must be flowed

```csharp
[ServiceContract(SessionMode=SessionMode.Required)]
public interface IDataUpdate
[OperationContract]
[TransactionFlow(TransactionFlowOption.Mandatory)]
void ProcessOrders();
```

A.12

Integration with COM+

WCF transaction protocols

- OleTransactions (default and uses MSDTC)
- WSAtomicTransaction11 (good when not using MSDTC, 2007)
- WSAtomicTransactionOctober2004

To integrate with COM+, set up an endpoint configured to use OleTransactions

```xml
<netTcpBinding>
  <binding name="TransactionTcpBinding"
    transactionFlow="true"
    transactionProtocol="OleTransactions"/>
```

Integrating with COM+ Applications Overview

TransactionProtocol Class
Discovery Announcements

**AnnouncementService**
- Used by clients to listen for and act on incoming messages on a standard announcement endpoint (AnnouncementEndpoint)
- Provides event notification when Hello or Bye announcement messages arrive

**AnnouncementClient**
- Used by services to send discovery announcement messages
- An announcement message contains information about the service such as its fully-qualified contract name, any scopes that the service is operating in as well as any custom metadata the service wants to send
- You do not need an AnnouncementClient if you want to service to make announcements automatically when the host opens and closes

Discovery Configuring Announcements and Discovery

You must enable discovery behavior for the service...
...and discovery and announcement endpoints with code...

```csharp
var disc = new ServiceDiscoveryBehavior();
disc.AnnouncementEndpoints.Add(new UdpAnnouncementEndpoint());
host.Description.Behaviors.Add(disc);
host.AddServiceEndpoint(new UdpDiscoveryEndpoint());
```

...or with configuration

```xml
<services>
  <service name="MyService">
    <endpoint kind="udpDiscoveryEndpoint" />
  </service>
</services>

<behaviors>
  <serviceBehaviors>
    <behavior>
      <serviceDiscovery>
        <announcementEndpoints>
          <endpoint kind="udpAnnouncementEndpoint" />
        </announcementEndpoints>
      </serviceDiscovery>
    </behavior>
  </serviceBehaviors>
</behaviors>
```
Find and FindAsync with FindCriteria

Both methods pass in a FindCriteria

- Has several properties, which can be grouped into search criteria, which specify what services you are looking for, and find termination criteria (how long the search should last)

Search criteria include:

- ContractTypeNames: if more than one contract name is specified, only service endpoints matching ALL contracts reply
- Scopes: absolute URIs that are used to categorize individual service endpoints
- ScopeMatchBy: ScopeMatchByExact, ScopeMatchByPrefix, ScopeMatchByLdap, and so on

Termination criteria include Duration and MaxResults

FindResponse

FindResponse

- Represents the response from a find request

FindResponse.Endpoints Property

- Gets a collection of EndpointDiscoveryMetadata for the discoverable services that matched the find request

EndpointDiscoveryMetadata

- Address: Gets or sets the endpoint address
- ContractTypeNames: Gets a collection of contract type names implemented by the service
- ListenUris: Gets the listen URIs for the service
- Extensions, Scopes, and Version: other properties
Routing
Routing Service Client Contracts

- When configuring the router it acts as a client to many potential services so the contract element of an endpoint can use a wildcard

Mapping MEPs to routing interfaces

<table>
<thead>
<tr>
<th>Message Exchange Pattern (MEP)</th>
<th>Interface</th>
</tr>
</thead>
<tbody>
<tr>
<td>Request-Response</td>
<td>IRequestReplyRouter</td>
</tr>
<tr>
<td>One-Way</td>
<td>ISimplexDatagramRouter</td>
</tr>
<tr>
<td>One-Way (with sessions)</td>
<td>ISimplexSessionRouter</td>
</tr>
<tr>
<td>Duplex</td>
<td>IDuplexSessionRouter</td>
</tr>
</tbody>
</table>

Routing Service and Content Based Routing

Routing
Combining Filters and Setting Priorities

- You may have multiple services that need to handle messages filtered by message content, for example
  - Service1 must handle orders worth up to £100
  - Service2 must handle orders worth more than £100

Message Filters
Appendix B
Implementing Security in WCF Services
Developing Windows Azure and Web Services

Updated 11th April 2014

WCF Security
Contents

<table>
<thead>
<tr>
<th>Topic</th>
<th>Slide</th>
</tr>
</thead>
<tbody>
<tr>
<td>Certificates</td>
<td>3</td>
</tr>
<tr>
<td>Comparison</td>
<td>7</td>
</tr>
<tr>
<td>Authentication</td>
<td>9</td>
</tr>
<tr>
<td>Authorization</td>
<td>11</td>
</tr>
<tr>
<td>Impersonation</td>
<td>14</td>
</tr>
<tr>
<td>Auditing</td>
<td>16</td>
</tr>
<tr>
<td>Protection</td>
<td>17</td>
</tr>
<tr>
<td>WCF 4.5 Bindings</td>
<td>20</td>
</tr>
</tbody>
</table>

Exam Topics: Secure a WCF service
- Implement message level security, transport level security
- Certificates

Improving Web Services Security: Scenarios and Implementation Guidance for WCF
**Certificates**

**What is X.509?**

- In cryptography, X.509 is a standard for a public key infrastructure (PKI)
- X.509 specifies, amongst other things:
  - Formats for public key certificates
  - Certificate revocation lists
  - Certification path validation algorithm
- Assumes a strict hierarchical system of certificate authorities (CAs) for issuing the certificates
  - The “chain of trust”
  - If a certificate is revoked, all under it are revoked too

**Working with Certificates**


---

**Certificates**

**What is a Certificate Authority (CA)?**

- A certificate authority or certification authority (CA) is an entity that issues digital certificates
  - The digital certificate certifies the ownership of a public key by the named subject of the certificate
  - This allows others (relying parties) to rely upon signatures or assertions made by the private key that corresponds to the public key that is certified
- An organization’s trusted root certificates can be distributed to all employees so that they can use the company PKI system
Certificates

Sample

Certificate:

Data:

Version: 1 (0x0)
Serial Number: 7829 (0x1e95)
Signature Algorithm: md5WithRSAEncryption
Issuer: C=ZA, ST=Western Cape, L=Cape Town, O=Thawte Consulting cc,
OU=Certification Services Division,
CN=Thawte Certificate CA/emailAddress=server-certs@thawte.com

Validity
Not Before: Jul 9 16:04:02 1998 GMT
Not After : Jul 9 16:04:02 1999 GMT

Subject: C=US, ST=Maryland, L=Pasadena, O=Brent Baccala,
OU=FreeSoft, CN=www.freesoft.org/emailAddress=baccala@freesoft.org

Subject Public Key Info:
Public Key Algorithm: rsaEncryption
RSA Public Key: (1024 bit)
Modulus (1024 bit):
...
e8:35:1c:9e:27:52:7e:41:8f
Exponent: 65537 (0x10001)
Signature Algorithm: md5WithRSAEncryption
...

What are X.509 Certificates Used For?

**Multiple reasons: Authenticate, Encrypt, Sign**

- **X509CertificateInitiatorClientCredential.SetCertificate()**: Used by a **client** to **identify itself**
- **X509CertificateInitiatorServiceCredential.SetCertificate()**: Used by a **service** to **validate a client certificate**
- **X509CertificateRecipientServiceCredential.SetCertificate()**: Used by a **service** to **identify itself**

**You can also set certificates by using configuration**

- If you are creating a service, credentials (including certificates) are specified under the **serviceBehaviors** section
- When you are programming a client, credentials (including certificates) are specified under the **endpointBehaviors** section

[X509CertificateRecipientServiceCredential Methods]
## Comparison Security Modes

<table>
<thead>
<tr>
<th>Security Mode</th>
<th>Encryption</th>
<th>Authentication</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>Transport</td>
<td>SSL (point-to-point)</td>
<td>Basic, Digest, Certificate, Windows, NTLM</td>
</tr>
<tr>
<td>TransportWithMessageCredential</td>
<td>SSL (point-to-point)</td>
<td>In SOAP Header (Certificate, UserName, IssuedToken, Windows*, Custom)</td>
</tr>
<tr>
<td>Message</td>
<td>Only if ProtectionLevel is set in service, operation, fault, or message contracts (end-to-end)</td>
<td>In SOAP Header (Certificate, UserName, IssuedToken, Windows*, Custom)</td>
</tr>
<tr>
<td>TransportCredentialOnly</td>
<td>None</td>
<td>Basic, Digest, Certificate, Windows*, NTLM</td>
</tr>
</tbody>
</table>

```xml
<basicHttpBinding>
  <binding name="kermit">
    <security mode="TransportCredentialOnly">
      <transport clientCredentialType="Windows" />
    </security>
  </binding>
</basicHttpBinding>
```

*Kerberos*

## Comparison Security Types

<table>
<thead>
<tr>
<th>Transport (e.g. HTTPS)</th>
<th>Message (SOAP)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Speed</td>
<td>Faster (and can be hardware accelerated)</td>
</tr>
<tr>
<td>Extensible</td>
<td>No</td>
</tr>
<tr>
<td>Authentication Choices</td>
<td>Basic, Digest, NTLM, Windows, Certificate</td>
</tr>
<tr>
<td>Encryption</td>
<td>Point-to-point</td>
</tr>
<tr>
<td></td>
<td>End-to-end</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Warning!</td>
</tr>
<tr>
<td></td>
<td>You cannot use Message security with a REST endpoint because there won't be a SOAP message to use</td>
</tr>
</tbody>
</table>

**WCF Security Guidelines**

A service’s endpoint identity is a value propagated to any client and used to authenticate the service

- At design time, the client developer determines the service’s identity from the endpoint’s metadata (exposed through WSDL)
- At runtime, the client application checks the claims of the service’s security credentials before sending any messages to the service

Identity types

- Domain Name System (DNS), Certificate or Certificate Reference, RSA, User principal name, Service principal name
- A DNS check enables you to use certificates reissued with a new RSA key but the same DNS or subject name, so the identity check is still valid

Service Identity and Authentication

Support for security context tokens (SCT)

- For clients that support SCT, key exchange and validation can be done once and cached for the session
- For clients that do not, key exchange and validation must be done per call

In environments that require maximum security you can force clients to authenticate on every call by disabling SCT support

```xml
<bindings>
  <wsHttpBinding>
    <binding .../>
    <security ...>
      <message establishSecurityContext="false" ...>
  </wsHttpBinding>
</bindings>
```

**Authorization**

**Restricting Operations Calls By Role**

Use PrincipalPermission to ensure only members of a Windows Group or ASP.NET role can call an operation

```csharp
using System.Security.Permissions;

[PrincipalPermission(SecurityAction.Demand, Role="Sales")]
public int AddNumbers(int a, int b)
```

- Name="Alice,Bob", Role="Admin,Sales", Authenticated="true"
- Note: by default, the role is a Windows group name

**SecurityAction**

- Demand: throw exception if security context does NOT match
- Deny: throw exception if security context DOES match

**PrincipalPermissionAttribute Class**


**Authorization**

**Service Authorization**

**principalPermissionMode**

- None, UseWindowsGroups, UseAspNetRoles, Custom

**roleProviderName** can be any ASP.NET Role Provider

- e.g. use "SqlProvider" to specify a SQL Server database for roles

```xml
<behaviors>
  <serviceBehaviors>
    <behavior name="kermit">
      <!-- below are the defaults -->
      <serviceAuthorization
        impersonateCallerForAllOperations="false"
        principalPermissionMode="UseWindowsGroups"
        roleProviderName=""/
        serviceAuthorizationManagerType="" />
    <authorizationPolicies>
      <add policyType="" />
  </serviceAuthorization>
</serviceBehaviors>
```

**<serviceAuthorization> element**

Authorization

Getting the Identity of the Caller

Self-hosted service needs to retrieve a caller’s identity
- ServiceSecurityContext represents the security context of a remote party
- On the client, represents the service identity and, on the service, represents the client identity

Correct (will always get name of caller)

Wrong (won’t work if not using impersonation)

Impersonation

Executing Operations Under Different Identities

ImpersonateCallerForAllOperations (default is false)
- Gets or sets a value that indicates whether the service performs impersonation for all the operations that it supports
- A value of false indicates that impersonation is specified for each operation

OperationBehavior.Impersonation (default NotAllowed)
- ImpersonationOption.Required: impersonation is performed
- ImpersonationOption.Allowed: impersonation is performed if ImpersonateCallerForAllOperations is true
- ImpersonationOption.NotAllowed: throws exception if ImpersonateCallerForAllOperations is true
Impersonation Level

To access out-of-process resources on behalf of a caller

```csharp
using System.Security.Principal;

```

TokenImpersonationLevel

- **None, Anonymous**: no impersonation
- **Identification**: identify but not impersonate
- **Impersonation**: impersonates the client’s security context on its local system, but not on remote systems
- **Delegation**: impersonates the client’s security context on remote systems

TokenImpersonationLevel Enumeration

Auditing
How to Enable Logging

To audit attempts to access a secure service

```xml
<serviceSecurityAudit
  auditLogLocation="Default"
  messageAuthenticationAuditLevel="None"
  serviceAuthorizationAuditLevel="None"
  suppressAuditFailure="true" />
```

Audit levels

- None (default), Success, Failure, SuccessAndFailure

Audit log location

- Default (default depends on OS), Application, Security

Suppress audit failure

- If we fail to write to the log, no exception is thrown by default

Audit levels
B.17 Protection
Securing Streams

Scenario
• Need to stream sensitive BLOBs over a public network

A possible correct solution
• Use basicHttpBinding with transport security
  • Streaming is only available for: BasicHttpBinding, NetTcpBinding, NetNamedPipeBinding, WebHttpBinding

Wrong solutions include anything that uses SOAP
• Message security: digital signatures for the message body cannot be performed because they require computing a hash over the entire message content; with streaming the content is not available when the headers are constructed
• Reliable sessions: these must buffer the message so it can be resent and must hold a copy; cannot be used with streaming

Algorithm Suite for Message Security

Algorithm suites are described in the WS-SecurityPolicy specification
• Can be set in configuration of a binding for message-level security

```xml
<bindings>
  <wsHttpBinding>
    <binding name="strongerWSBinding">
      <security mode="Message">
        <message algorithmSuite="TripleDes" clientCredentialType="UserName" />
      </security>
    </binding>
  </wsHttpBinding>
</bindings>
```

Other options
• Basic256 (default), Basic128, Basic128Rsa15, Basic128Sha256, TripleDesSha256Rsa15, and so on

SecurityAlgorithmSuite Class
If you need to control the algorithm used by SSL then you must create a custom binding

```xml
<bindings>
  <customBinding>
    <binding name="strongerSSL" ...>
      <security defaultAlgorithmSuite="TripleDesSha256">
        ...
      </security>
    </binding>
  </customBinding>
</bindings>
```

Other values include

• Basic128, TripleDes, Basic256Sha256Rsa15, and so on

WCF 4.5 Bindings

WCF Security Bindings in .NET 4.5 are Simpler

Instead of configuring like this in WCF 4

```xml
<services>
  <service name="Firebrand.SampleService">
    <endpoint address="" binding="basicHttpBinding">
      <bindings>
        <basicHttpBinding>
          <binding name="kermit">
            <security mode="Transport" />
          </binding>
        </basicHttpBinding>
      </bindings>
    </endpoint>
  </service>
</services>
```

You can now configure like this in WCF 4.5

```xml
<services>
  <service name="Firebrand.SampleService">
    <endpoint address="" binding="basicHttpsBinding">
      <bindings>
      </bindings>
    </endpoint>
  </service>
</services>
```
Appendix C
“Classic” XML and ADO.NET
Developing Windows Azure and Web Services

Updated 11th April 2014

“Classic” XML and ADO.NET
Contents

<table>
<thead>
<tr>
<th>Topic</th>
<th>Slide</th>
</tr>
</thead>
<tbody>
<tr>
<td>System.Xml</td>
<td>3</td>
</tr>
<tr>
<td>DataSets and XML</td>
<td>11</td>
</tr>
<tr>
<td>XPath</td>
<td>15</td>
</tr>
<tr>
<td>Connections</td>
<td>17</td>
</tr>
<tr>
<td>Protecting Data</td>
<td>28</td>
</tr>
<tr>
<td>Factories</td>
<td>34</td>
</tr>
<tr>
<td>Commands</td>
<td>36</td>
</tr>
<tr>
<td>DataReaders</td>
<td>40</td>
</tr>
<tr>
<td>DataAdapters</td>
<td>44</td>
</tr>
<tr>
<td>DataSets</td>
<td>48</td>
</tr>
<tr>
<td>SQL Server</td>
<td>57</td>
</tr>
</tbody>
</table>

Slides 3-27 are most likely to appear in the exam

- **Exam Topic: Manipulate XML data structures**
  - Read, filter, create, modify XML data structures
  - Manipulate XML data by using XMLReader, XMLWriter, XMLDocument, XPath, transform XML by using XSLT transformations

- **Exam Topic: Query and manipulate data by using ADO.NET**
  - Query and manipulate data by using Connection, DataReader, Command, DataAdapter, DataSet
  - Perform synchronous and asynchronous operations
  - Manage transactions (API)
System.Xml (aka “Classic” XML)
XML Processing (.NET 1.0 and later)

For fast, memory-efficient processing of XML
- These use 8KB cache to load each node one at a time
- XmlReader (forward-only, read-only, fast, low-overhead)
- XmlWriter

For DOM-based XML
- These must load the entire file into memory
- XmlDocument: full functionality
- XmlDataDocument: an XmlDocument that can be sync-ed with DataSet
- XPathDocument: read-only but faster XPath queries and transformations

XML Documents and Data

System.Xml
Reading XML Efficiently

Properties
- Name, NodeType, HasValue, Value, HasAttributes, AttributeCount, EOF

Methods
- Read, ReadContentAs, ReadElementContentAs, ReadInnerXml, ReadOuterXml, ReadToDescendant, ReadToNextSibling, etc.
- MoveToAttribute, MoveToContent, MoveToElement, Skip
- MoveToFirstAttribute, MoveToNextAttribute, GetAttribute
- IsStartElement

```csharp
var myReader = XmlReader.Create("SampleXml.xml");
while (myReader.Read()) {
    if (myReader.NodeType == XmlNodeType.Element) {
        Console.WriteLine(myReader.Name);
    } else if (myReader.NodeType == XmlNodeType.Text) {
        Console.WriteLine(myReader.Value);
    }
```
Validating XML with XmlReader

Use XmlReaderSettings class with an XmlReader to perform validation

```csharp
var xrs = new XmlReaderSettings();
xrs.ValidationType = ValidationType.Schema;
xrs.Schemas.Add(null, "books.xsd");
xrs.ValidationEventHandler += xrs_VEH;
var xr = XmlReader.Create("books.xml", xrs);
```

When adding a schema, pass null or empty string to specify the default XML namespace

```csharp
public void xrs_VEH(object sender, ValidationEventArgs e)
{
    // e.Error, e.Message
}
```

XML Namespaces

What’s the difference between Name and LocalName?

- XML can have namespaces with prefixes to differentiate elements and attributes defined by different schemas
- Elements without prefixes belong to the default namespace

```xml
<item xmlns:media="http://schemas.microsoft.com/video/2006/04">
    <title>Article 1</title>
    <description><![CDATA[How to use StackOverflow.com]]></description>
    <link>http://youtube.com/?v=y6_-cLWwEU0</link>
    <media:player url="http://youtube.com/?v=y6_-cLWwEU0" />
    <media:thumbnail url="http://img.youtube.com/vi/y6_-cLWwEU0/default.jpg" width="120" height="90" />
</item>
```

- Use Name to retrieve the prefix:name (e.g. media:player) or LocalName to retrieve just the name without the prefix (player)
Writing XML with the XmlWriter Class

Abstract class that allows you to write XML to a file, console, stream, or other output types

- Configure using an XmlWriterSettings object

```csharp
var settings = new XmlWriterSettings();
settings.Indent = true;
var aWriter = XmlWriter.Create("newfile.xml", settings);
```

Writing Elements

Write methods efficiently generate well-formed XML

- WriteStartDocument, WriteEndDocument
- WriteElementString, WriteStartElement, WriteEndElement, WriteFullEndElement
- WriteAttributeString, WriteComment, and so on

```csharp
myWriter.WriteStartElement("FirstNames");
myWriter.WriteAttributeString("color", "Red");
myWriter.WriteElementString("Name", "Libby");
myWriter.WriteEndElement();
myWriter.WriteStartElement("LastNames");
myWriter.WriteEndElement();
```

```xml
<FirstNames color="Red">
  <Name>Libby</Name>
</FirstNames>
<LastNames />
```
**C.9**

**System.Xml**

**XmlDocument**

```xml
<?xml version="1.0"?>
<books>
  <book>
    <author>Carson</author>
    <price format="dollar">31.95</price>
    <pubdate>05/01/2001</pubdate>
  </book>
  <pubinfo>
    <publisher>MSPress</publisher>
    <state>WA</state>
  </pubinfo>
</books>
```

Process XML Data Using the DOM Model

**C.10**

**System.Xml**

**XSLT Transformations**

The Transform method accepts three input types for the source document: an object that implements the IXPathNavigable interface, an XmlReader object that reads the source document, or a string URI

```csharp
using System.Xml;
using System.Xml.XPath;
using System.Xml.Xsl;

var doc = new XPathDocument("books.xml");
var writer = XmlWriter.Create("books.html");
var transform = new XslCompiledTransform();
var settings = new XsltSettings();
settings.EnableScript = true; // if you use scripting in the XSLT
transform.Load("transform.xsl", settings, null);
transform.Transform(doc, writer);
```

XSLT Transformations
WriteXml method can be used to write the data to an XML file or a stream

```csharp
aDataSet.WriteXml(HttpContext.Current.Server.MapPath(@"App_Data\employee.xml"));
```

```xml
<?xml version="1.0" standalone="yes" ?>
<NewDataSet>
    <Employee>
        <Eid>123456789A</Eid>
        <FirstName>Nancy</FirstName>
        ...
    </Employee>
    <Employee>
        ...
    </Employee>
</NewDataSet>
```

---

GetXml method
- Returns XML as a string

XmlWriteMode enumeration
- IgnoreSchema (default): outputs data only
- WriteSchema: outputs an inline schema as well as data
- DiffGram: outputs original and changed data

For multiple tables in a DataSet
- The result of the WriteXml is XML that lists each table in the order they were defined in the DataSet’s table collection
- To enforce nesting of nodes for related tables, set the Nested property of the DataRelation object to true
**C.13**

**DataSets and XML**

**Using Diffgrams**

Diffgrams maintain row state and versions

```xml
<diffgr:diffgram
  xmlns:msdata="urn:schemas-microsoft-com:xml-msdata"
  xmlns:diffgr="urn:schemas-microsoft-com:xml-diffgram-v1">
  <AWDataSet> ... </AWDataSet>
  <diffgr:before> ... </diffgr:before>
  <diffgr:errors> ... </diffgr:errors>
</diffgr:diffgram>
```

---

**C.14**

**DataSets and XML**

**Reading XML with a DataSet**

**ReadXml method**
- Not all XML files can be read into a DataSet
- May need to explicitly Clear a DataSet before reading into it

**XmlReadMode enumeration**
- Auto (default)
- DiffGram: if XML is in DiffGram format
- Fragment: allows multiple root elements
- ReadSchema: if XML contains schema, it is processed
- IgnoreSchema: if XML contains schema, it is ignored
- InferSchema: if XML contains schema, it is ignored; schema inferred from data; all column data types are strings
- InferTypedSchema: column data types are inferred
XPath Overview

Use / to define a path to elements and attributes
- Name of all author elements
  bookstore/author/name
- All period attributes
  bookstore/author/@period

Use * as a wildcard
- Name of all authors and books
  bookstore/*/name
- Name and nationality of all authors
  bookstore/author/*

XPath Context and Filtering

Context is very important for XPath statements
- Author at root of document
  /author
- Authors anywhere in document
  //author

Use [ ] to define criteria to select nodes
- Names of authors with Russian nationality
  //author[nationality='Russian']/name
- Nationalities of authors not from the classical period
  //author[@period!='Classical']/nationality
SqlConnection Extra Members

- EnlistDistributedTransaction
- PacketSize, ServerVersion, WorkstationId
- StatisticsEnabled, RetrieveStatistics(), ResetStatistics()
- FireInfoMessageOnUserErrors: False (default), True
  - Class (severity) 1-10: raises InfoMessage event
  - Class 11-16: throws SQLException (when FireInfoMessageOnUserErrors is False) or raises InfoMessage event (when FireInfoMessageOnUserErrors is True)
  - Class 17+: throws SQLException and closes connection

RAISERROR

RAISERROR 'Error message', 11, 1234567

RAISERROR

Detecting Information with the Connection Event

InfoMessage event

- Raised when PRINT and RAISERROR SQL statements are used

conn.InfoMessage += conn_InfoMessage;

private void conn_InfoMessage(object sender, SqlInfoMessageEventArgs e)
{
    // e.Errors is a collection of SqlError objects
}
Connections

Configuring Connection Strings

Connection strings provide the necessary information for an application to connect a data source
- Server, database, security information, connection pool size

Many parameters have multiple possible keywords for backwards compatibility e.g. these all mean the same
- Data Source, server, address, addr, Network Address

SQL Server (servername\instancename)

```
Data Source=FLORIDA\MIAMI;Initial Catalog=Northwind;
Integrated Security=SSPI;
database=Northwind;
```

OLEDB for Microsoft Access

```
Provider=Microsoft.Jet.OLEDB.4.0;
Data Source=C:\MyDataFiles\MyDb.mdb;
```

Connections

Security

Windows Authentication
- Secure as no password is transmitted across the network
- Not easy to configure across domains or firewalls
- Use of service accounts is highly recommended to aid pooling

```
"...;Integrated Security=SSPI;"  "...;Trusted_Connection=true;"
```

Mixed (aka SQL) Authentication
- Not as secure because user id and password sent across network
- Works across domains and through firewalls
- Only option when using SQL Databases in Windows Azure

```
"...;User Id=Fred;Password=secret;Persist Security Info=false;"
"...;uid=Fred;pwd=secret;PersistSecurityInfo=false;"
```
Connections
Database Instances—Default and User Instances

To attach a local database file as a user instance

```csharp
User Instance=true;
Data Source=\SQLExpress;
AttachDbFilename=\DataDirectory\pubs.mdf;
Integrated Security=true;
```

- `DataDirectory` means App_Data for a web site
  - Can also use an absolute path

Connections
Encrypting Data Transmission

Transmission between client and database server

- Enable SSL in connection string

  ```
  ...;Encrypt=true;
  ```

- SQL Server must be properly configured with a valid certificate installed

  - Exception will be thrown on Open method call, for example:
    - A connection was successfully established with the server, but then an error occurred during the pre-login handshake.
    (provider: SSL provider, error: 0 - The certificate chain was issued by an authority that is not trusted.)
C. Connections

C.23 Lifetime and Pooling

- **Connection exists between Open and Close**
  - Connections use resources on client and server

- **Connection pools**
  - Reusing existing connections to reduce overhead and increase performance
  - Pooled connections must have identical connection string
  - Pools are separated by process, application domain, connection string, and for each user context when using integrated security
  - Pool is on client-side; server has no idea pooling is being used

---

C.24 Connection Pooling Keywords

<table>
<thead>
<tr>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connection Lifetime or Load</td>
<td>0</td>
<td>If the number of seconds since creation is greater than the value, then the connection is destroyed when returned to the pool</td>
</tr>
<tr>
<td>Balance Timeout</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Connection Reset</td>
<td>True</td>
<td>Ensures the connection points to the original database when returned to the pool (change to false to improve performance)</td>
</tr>
<tr>
<td>Enlist</td>
<td>True</td>
<td>Enlist in the thread's current transaction context</td>
</tr>
<tr>
<td>Max/Min Pool Size</td>
<td>100/0</td>
<td>The maximum/minimum number of connections allowed in the pool</td>
</tr>
<tr>
<td>Pooling</td>
<td>True</td>
<td>Connection pooling is enabled or disabled</td>
</tr>
</tbody>
</table>
Connections

Multiple Active Result Sets (MARS)

- On a standard connection only one result set can be processed at a time and no other operations can occur through the connection until the result set is closed.
- MARS allows a connection to be reused in some situations.
- MARS must be supported by the database.
  - SQL Server 2005 or later supports MARS.
- MARS must be enabled in the connection string.

```
"...;MultipleActiveResultSets=true;"
```

Connections

Protecting Configuration Files

- How to protect `<connectionStrings>` programmatically

```csharp
var s = config.GetSection("connectionStrings") as ConnectionStringsSection;
s.SectionInformation.ProtectSection("RsaProtectedConfigurationProvider");
```

- Use UnprotectSection() to decrypt

- Two providers
  - RsaProtectedConfigurationProvider
  - DataProtectionConfigurationProvider

- Warning!
  - To use the same encrypted configuration file on multiple servers, such as a Web farm, only the RsaProtectedConfigurationProvider enables you to export the keys and import them on another server.
Connections
ConnectionStringBuilder

To dynamically but safely build a connection string

```csharp
var sb = new SqlConnectionStringBuilder();
sb["Data Source"] = @"\SQLEXPRESS"; // or sb.DataSource
sb["database"] = "AdventureWorks";
// or sb["Initial Catalog"]
sb["Integrated Security"] = true;
// this connection string will use the modern keywords
string connectionString = sb.ConnectionString;
```

Can also be used to test a connection string

- An exception is thrown if any values are invalid

```csharp
var sb = new SqlConnectionStringBuilder(conStrToTest);
// or set ConnectionString property
```

Protecting Data
Three Techniques

Encrypt
- Two-way operation (i.e. can be decrypted)
- Best choice for data such as credit card numbers

Hash (integrity check)
- One-way operation (i.e. cannot create original data from hash)
- A checksum that is unique to a piece of data to ensure no modification occurs
- Best choice for data such as passwords

Sign (authentication check)
- A digital signature is a value that is appended to electronic data to prove it was created by someone who possesses a specific private key; the public key is used to verify the signature at the receiver's end
Protecting Data

Three Types of Algorithm

- **Non-Keyed**
  - Simple to code but weak

- **Symmetric Key (aka secret or shared key)**
  - Same key on both sides

- **Asymmetric Keys**
  - Public-private key pair
  - Mathematically linked but cannot derive one from the other

Protecting Data

Symmetric Encryption

- **Good**
  - Fast, large amounts of data

- **Bad**
  - Need a way to share the key

- **OS-Implemented Algorithms (unmanaged code)**
  - DES (common but should be avoided)
  - TripleDES
  - RC2 (official replacement for DES)

- **Managed Algorithms (supports partially-trusted code)**
  - RijndahlManaged, AesManaged
  - Advanced Encryption Standard (AES) is Rijndael with fixed block size and iteration count: best choice
Protecting Data

Asymmetric Encryption

- **Good**
  - More secure than symmetric encryption

- **Bad**
  - Slow, small amounts of data

- **Algorithm**
  - RSACryptoServiceProvider: encrypt (and also sign!)
    - Name comes from initials of three men who invented it

- **How it works**
  - Sender uses receiver's public key to encrypt data
  - Receiver uses their private key to decrypt
  - Often combined with symmetric for best of both worlds, for example, HTTPS/SSL

Protecting Data

Hash and Sign

- **Non-Keyed Hash Algorithms**
  - Secure Hash Algorithm (SHA) with different hash sizes
    - SHA1 (160 bit), SHA256, SHA384, SHA512
    - MD5: Message Digest 5 (128 bit hash)

- **Symmetric Keyed Hash Algorithms**
  - HMACSHA1: Hash-based Message Authentication Code (HMAC)
  - MACTripleDES: 8, 16, 24 byte keys; 8 byte hash size (64 bit)

- **Asymmetric Keyed Hash and Sign Algorithm**
  - Digital Signature Algorithm (DSA)
    - DSA CryptoServiceProvider: hash and sign data
    - DSA cannot encrypt data, only hashes!
    - Do not confuse with RSACryptoServiceProvider
C.33

Protecting Data
Random Number Generators and Salts

RNGCryptoServiceProvider class
- The class can be used to generate a random number for use various types of cryptography and other operations

Example
- To store user passwords in the database in a way that they cannot be extracted, the passwords need to be hashed using a one-way hashing algorithm such as SHA1
- To do so, use the RNGCryptoServiceProvider to create a random salt, append the salt to the password, hash it using SHA1 CryptoServiceProvider class, and store the resulting string in the database along with the salt
- The benefit provided by using a salted password is making a lookup table assisted dictionary attack against the stored values impractical, provided the salt is large enough

C.34

Factories
DbProviderFactories

Pass a string containing invariant name for provider

```csharp
using System.Data.Common;

var factory = DbProviderFactories.GetFactory("System.Data.SqlClient");
```

GetFactoryClasses method
- Returns a DataTable loaded from Machine.config
- 0:Name, 1:Description, 2:InvariantName, 3:AssemblyQualifiedName

Use the factory to create any subsequent objects

```csharp
DbConnection connection = factory.CreateConnection();
DbCommand command = factory.CreateCommand();
DbParameter param = factory.CreateParameter();
```

You can also get a list of available sources of data, for example, SQL Servers

```csharp
DataTable sources = factory.GetDataSourceEnumerator().GetSources();
```
 Commands
DbCommand Members

Methods
• ExecuteNonQuery: returns Int32
• ExecuteReader: returns DbDataReader-derived object
• ExecuteScalar: returns Object
• Prepare, Cancel

Properties
• CommandType: TableDirect, StoredProcedure, Text (default)
• CommandText: name of table, stored procedure, or SQL
• CommandTimeout: default 30 seconds
• Connection, Transaction: objects associated with this command
• Parameters: collection of DbParameters

SqlCommand.ExecuteNonQuery

• Warning! Use ExecuteScalar to return the result as a string if you just need to write the XML to a file or stream rather than immediately process it with a reader

XML
<Customers CustomerID="ALFKI"
  CompanyName="Alfreds Futterkiste" Country="Germany" />

XMLReader xr = cmd.ExecuteXmlReader();
while (xr.Read())
{
  Console.WriteLine(xr.GetAttribute("CompanyName"));
}
**Properties**

- **ParameterName**: string
- **Value**: object
- **DbType**: DbType enumeration
  - Binary, Byte, Boolean, Currency, and so on
- **Size**: in bytes
- **Direction**: ParameterDirection enumeration
  - Input, Output, InputOutput, ReturnValue
- **IsNullable**: true or false
- **SourceColumn**, **SourceVersion**: used to set value when used during DbDataAdapter.Update

---

**ExecuteNonQuery with Stored Procedures**

- **ExecuteNonQuery** returns -1 when executing a stored procedure so you must use a ReturnValue or output parameters

- You can only have one return value but you can have multiple output parameters

```csharp
CREATE PROCEDURE dbo.InsertCategory
    @CategoryName nvarchar(15),
    @Identity int OUT -- or OUTPUT
AS
    INSERT INTO Categories (CategoryName)
    VALUES(@CategoryName)
    SET @Identity = SCOPE_IDENTITY()
    RETURN @@ROWCOUNT
```

```csharp
var param1 = cmd.Parameters.Add("@RowCount", SqlDbType.Int);
param1.Direction = ParameterDirection.ReturnValue;

var param2 = cmd.Parameters.Add("@Identity", SqlDbType.Int, 4);
param2.Direction = ParameterDirection.Output;
```
CommandBehavior values

- **CloseConnection**: close related connection when close reader
- **Default**: equivalent to no parameter
- **SchemaOnly**: returns column information (call GetSchemaTable method of reader to return a DataTable); SqlCommand prefixes call with `SET FMTONLY ON`
- **KeyInfo**: returns column and key information
- **SequentialAccess**: instead of loading entire row into a buffer, loads each column sequentially as a stream; good for BLOBs
- **SingleResult**: optimizes for a single result set
- **SingleRow**: optimizes for a single row; good for commands that pass a value for the primary key

IDataRecord interface

- **Pass ordinal position, i, of column**
  - **GetName(i)**: returns name of column
  - **GetBoolean(i)**, **GetInt32(i)**, **GetString(i)**, and so on: returns column as strongly-typed values
  - **GetValue(i)**: returns column value as Object
  - **GetData(i)**: returns a IDataReader for column value
  - **IsDBNull(i)**: returns True if value is null

- **Others**
  - **FieldCount** property
  - **Item()**, [int], [string]: indexers return column value as Object
  - **GetOrdinal("column")**: returns ordinal position of column
  - **GetValues(object[])**: returns number of items
  - **GetDataTypeName**: returns string, **GetFieldType**: returns Type
**DataReaders**

**IDataReader and DbDataReader members**

- **IDataReader members**
  - RecordsAffected: rows inserted, updated, or deleted
  - Read(): if true, moves to next row
  - NextResult(): if true, moves to next resultset
  - IsClosed, Close(): must close IDataReader before reading any output parameters or return values

- **DbDataReader members**
  - HasRows
  - GetSchemaTable(): returns DataTable with schema info

- **SqlDataReader only: GetSqlXxx(int)**
  - GetSqlMoney, GetSqlXml, GetSqlGuid, etc.

---

**How to Process Multiple SELECTs**

- **Does NOT need MARS**

```c
cmd.CommandText = "SELECT * FROM Customers; SELECT * FROM Products";
SqlDataReader reader = cmd.ExecuteReader();
// do NOT call NextResult() yet!
do{
    while (reader.Read())
    {
        Console.WriteLine(reader[0].ToString());
    }
} while (reader.NextResult());
```
How to Populate Multiple DataTables

You can efficiently populate multiple DataTables in a DataSet with a single Fill call by using table mappings.

```csharp
da.SelectCommand.CommandText = "SELECT * FROM Customers; SELECT * FROM Products";
da.TableMappings.Add("Table", "Customers");
da.TableMappings.Add("Table1", "Products");
da.Fill(ds);
```

Table, Table1, Table2, and so on are logical names that represent the results of the SELECT statements.

SqlCommandBuilder

Used to generate INSERT, UPDATE, and DELETE statements based on a SELECT command.

- Must be associated with a data adapter.

```csharp
var cmd = new SqlCommand();
    cmd.CommandText = "SELECT * FROM Products";
var da = new SqlDataAdapter(cmd);
var bldr = new SqlCommandBuilder(da);
    bldr.ConflictOption = ConflictOption.OverwriteChanges;
da.InsertCommand = bldr.GetInsertCommand();
da.UpdateCommand = bldr.GetUpdateCommand();
da.DeleteCommand = bldr.GetDeleteCommand();
```

Other ConflictOption values:

- CompareAllSearchableValues: WHERE includes all columns
- CompareRowVersion: if timestamp column it is used
Performing Bulk Updates

- The UpdateBatchSize property of the DbDataAdapter object’s allows you to send updates to the DB in batches rather than record by record.

- The UpdateBatchSize property value can be set to:
  - 1 (default), which indicates one row at a time
  - 0, all updated rows in one batch
  - Any other value to indicate number of rows to include in the batch

```csharp
da.UpdateBatchSize = 3;
da.Update(pubsDataSet, "publishers");
```

TableAdapterManager

- Enables hierarchical updates (ADO.NET 3.5+)
  - Created at design-time with a typed DataSet so not in BCL
  - Uses the foreign-key constraints in data tables to determine the correct order to send the Inserts, Updates, and Deletes from a dataset to the database

- Members
  - BackupDataSetBeforeUpdate: use for AutoIncrements that must not miss values
  - UpdateOrder: InsertUpdateDelete (default)
  - UpdateAll(): implicit transaction
DataSets

Properties That Improve Efficiency

▪ EnforceConstraints
  • If false, UniqueConstraints and ForeignKeyConstraints are temporarily ignored
  • Useful to improve performance when filling a DataSet

▪ RemotingFormat
  • Xml (default) or Binary (usually smaller, but 20kb overhead)

▪ SchemaSerializationMode
  • IncludeSchema (default) or ExcludeSchema

DataSets

DataColumn.Expression

▪ When you create an expression, use the ColumnName property to refer to columns
  ```
col.Expression = "UnitPrice * Quantity";
```

▪ You can use an aggregate expression if you have a DataRelation between tables
  ```
col.Expression = "Count(Child(Product))";
```

▪ IIf (inline if)
  ```
col.Expression = "IIf(ListPrice > 100, 'Expensive', 'Cheap')";
```
**DataSets**

**DataRow.DataRowState**

- **Detached**
  - Newly created row not yet added to Rows collection

- **Added**
  - Newly created row now member of Rows collection

- **Unchanged**
  - Row that has not been modified; only has current version

- **Modified**
  - Row that has been modified; has current and original versions

- **Deleted**
  - Row that has been marked for deletion; only has original version

---

**DataSets**

**DataRowVersion**

- **DataRows** can have up to four versions
  - Current
  - Original
  - Proposed: only exists until EndEdit is called on an edited row
  - Default: only one copy of this version exists for each table

- **Read versions with overloaded indexer**

  ```csharp
  s = row["FirstName", DataRowVersion.Original];
  ```

- **VersionNotFoundException** is raised if you attempt to retrieve a version that does not exist
  - Use HasVersion method to test for existence of a version

  ```csharp
  if(row.HasVersion(DataRowVersion.Original))
  ```
Modifying Data in a DataTable

Select a row

```
DataRow[] SelectedRows = aTable.Select("ID='CONTO'";  
DataRow SelectedRow = SelectedRows[0];
```

Modify a row

```
SelectedRow.BeginEdit();  // optional
SelectedRow["CompanyName"] = "Contoso";
SelectedRow.EndEdit();  // Proposed becomes Current
```

Mark a row for deletion (tracks this change)

```
SelectedRow.Delete();
```

Remove a row completely (change not tracked!)

```
aTable.Rows.Remove(SelectedRow);
```

AcceptChanges and RejectChanges

AcceptChanges

- Can be issued on the DataRow, DataTable and, DataSet objects
- Data changes are accepted
- DataRow state is set to Unchanged

RejectChanges

- Can be issued on the DataRow, DataTable and, DataSet objects
- Data changes are rejected
- DataRow is rolled back to the point in time when the last AcceptChanges method was called

Can also explicitly change with SetAdded and SetModified
**DataSets**
**Getting Changes for a DataSet or DataTable**

- The **GetChanges** method generates a new DataSet or DataTable with only inserted, modified or deleted rows.
- Can be used to:
  - Send the changes to the data source when only a few rows in the original DataSet were changed.
  - Transport only changed data.

**DataSets**
**Using Merge to Combine DataSet Data**

- The **Merge** method merges one DataSet into another.
- If the DataSet schemas match, the data merged:
  - Is added if the primary key does not match an existing row.
  - Replaces existing data if the key matches and the **PreserveChanges** property is set to false (default).
  - Ignored if key matches and the **PreserveChanges** method property is set to true.
- If the schemas do not match, the **MissingSchemaAction** property can be set to:
  - Add: creates missing tables and columns.
  - AddWithPrimaryKey: also creates missing primary keys.
  - Error: throws exception.
  - Ignore: skips data for missing tables and columns.
Add a ForeignKeyConstraint to Constraints collection of the foreign table

Properties of ForeignKeyConstraint
- Table: Orders, Columns: CustomerID
- RelatedTable: Customer, RelatedColumns: ID

DeleteRule, UpdateRule
- Cascade (default): child rows deleted or updated (if necessary)
- None (throw exception), SetDefault, SetNull

AcceptRejectRule
- None (default)
- Cascade: AcceptChanges or RejectChanges on related child rows
Avoiding SQL Injection Attacks

SQL injection is an attack in which malicious code is inserted into strings that are later passed to an instance of SQL Server for parsing and execution.

- For example, this bad code reads a value posted from a web form and concatenates it into a SQL statement:
  ```csharp
  var ShipCity = Request.Form("ShipCity");
  var sql = "select * from OrdersTable where ShipCity = '" + ShipCity + '";"
  ```

- A malicious user could enter the following for ShipCity:
  ```csharp
  Redmond'; drop table OrdersTable--
  ```

- Reject the following characters: ' ; -- /* */ xp_

- Better: use parameterized statements or stored procedures

SQL Injection

SQL Server
StoredProcedure

Stored procedures should use parameters as their input:

```sql
CREATE PROCEDURE HumanResources.uspGetEmployees
    @LastName nvarchar(50) = NULL, --default value
    @FirstName nvarchar(50),
    @HowMany int OUTPUT /* or OUT */
AS
SELECT COUNT(*) FROM ...
SET @HowMany = SELECT COUNT(*) FROM ...
SELECT FirstName, LastName, Department FROM HumanResources.vEmployeeDepartmentHistory WHERE FirstName = @FirstName AND LastName = @LastName;
```

- Some organizations require use of stored procedures, others such as Facebook don’t use them at all


How to: Create a Stored Procedure (SQL Server Management Studio)
Partitioning a database improves performance and simplifies maintenance

- Hardware partitioning
  - RAID
- Horizontal partitioning (SQL Server 7.0 and later)
  - Each table has same columns, but subset of rows
  - Excessive UNION queries, used to merge the tables logically at query time, can affect performance
- Vertical partitioning (SQL Server 2005 and later)
  - Each logical row in a split table matches the same logical row in the other tables as identified by a UNIQUE KEY column that is identical in all of the partitioned tables

---

**Equivalent .NET Types**

<table>
<thead>
<tr>
<th>SQL Server</th>
<th>Best .NET Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>money</td>
<td>Decimal</td>
</tr>
<tr>
<td>bit</td>
<td>Boolean</td>
</tr>
<tr>
<td>binary / image</td>
<td>Byte[]</td>
</tr>
<tr>
<td>numeric</td>
<td>Decimal</td>
</tr>
<tr>
<td>float</td>
<td>Double</td>
</tr>
<tr>
<td>uniqueidentifier</td>
<td>Guid</td>
</tr>
<tr>
<td>varchar, nvarchar, char, nchar, text, ntext</td>
<td>String, Char[]</td>
</tr>
<tr>
<td>datetime</td>
<td>DateTime</td>
</tr>
<tr>
<td>date</td>
<td>DateTime</td>
</tr>
<tr>
<td>time</td>
<td>TimeSpan</td>
</tr>
<tr>
<td>timestamp</td>
<td>Byte[]</td>
</tr>
<tr>
<td>bigint</td>
<td>Int64</td>
</tr>
<tr>
<td>smallint</td>
<td>Int16</td>
</tr>
<tr>
<td>tinyint</td>
<td>Byte</td>
</tr>
</tbody>
</table>
C.61

SQL Server
Custom CLR Types

SQL Server 2005 and later supports the CLR and so can use any .NET type as a column or parameter type
• Must use SqlCommand
• Must set SqlParameter.SqlDbType to SqlDbType.Udt
• Must set SqlParameter.UdtTypeName to the type name

```csharp
paramLocation.SqlDbType = SqlDbType.Udt;
paramLocation.UdtTypeName = "Point";
```

• Type name can be fully-qualified (as registered in database, NOT the .NET fully-qualified name)

```csharp
paramLocation.UdtTypeName = "AdventureWorks.dbo.Point";
```

C.62

SQL Server
Spatial Datatypes

SQL Server 2008 and later
• geometry: planar (Euclidean) data
  • SqlDbType.Udt
  • UdtTypeName: "GEOMETRY"
• geography: ellipsoidal data such as GPS coordinates
  • SqlDbType.Udt
  • UdtTypeName: "GEOGRAPHY"
SQL Server 2008 and later

- A table type can be used as parameter to a stored procedure

```sql
CREATE TYPE dbo.Reviews AS TABLE
    (ReviewID int, ReviewText nvarchar(50))

CREATE PROCEDURE AddReviews
    (@reviews dbo.Reviews READONLY)
```

- In .NET, pass a DataTable, IEnumerable<SqlDataRecord>, or DbDataReader

```csharp
var cmd = new SqlCommand("AddReviews");
cmd.CommandType = CommandType.StoredProcedure;
cmd.Parameters.Add("reviews", SqlDbType.Structured);
cmd.Parameters["reviews"].Value = dataTableOfReviews;
```

**FILESTREAM in SQL Server 2008**

- Can also use GetSqlBytes() to store image in SqlBytes variable instead of reading from SqlFileStream

```sql
CREATE TABLE employees ( ...,
    Photo VARBINARY(MAX) FILESTREAM NULL, ...
)
```

```csharp
command.CommandText = "select Photo.PathName()," +
    "GET_FILESTREAM TRANSACTION_CONTEXT() from employees";
using (SqlDataReader reader = command.ExecuteReader()) {
    string path = reader.GetString(0);
    SqlFileStream fileStream = new SqlFileStream(path,
        (byte[])reader.GetValue(1), FileAccess.Read,
        FileOptions.SequentialScan, 0);
    for (long index = 0; index < fileStream.Length; index++){
        Console.Write(fileStream.ReadByte());
    }
    fileStream.Close();
```
How to read SQL Server xml columns

If you want to work with the content of the column as XML you must use an XmlReader

```csharp
while (salesReaderData.Read()) // SqlDataReader
{
    SqlXml salesXML = salesReaderData.GetSqlXml(0);
    XmlReader salesReaderXml = salesXML.CreateReader();
    salesReaderXml.MoveToContent();
    while (salesReaderXml.Read())
    {
        if (salesReaderXml.NodeType == XmlNodeType.Element)
        {
            string elementLocalName = salesReaderXml.LocalName;
            salesReaderXml.Read();
            Console.WriteLine(elementLocalName + " ": " +
            salesReaderXml.Value);
        }
    }
}
```

using System.Data.SqlTypes;

SQL Server Native Client

- Is a stand-alone data access API used for both OLE DB and ODBC, that was introduced in SQL Server 2005
- Provides new functionality above and beyond that supplied by the Windows Data Access Components (Windows DAC, formerly Microsoft Data Access Components, or MDAC)
- Can be used to create new COM-based applications or enhance existing COM-based applications that need to take advantage of features introduced in SQL Server 2005, such as multiple active result sets (MARS), user-defined data types (UDT), query notifications, snapshot isolation, and XML data type support
- If you are developing a managed application, you should use the .NET Data Provider for SQL Server, not the native client

SQL Server Native Client Programming
Data access tracing is available not only in ADO.NET, but also in MDAC 2.8 SP2, Microsoft Windows Data Access Components, SQL Server Native Client.

A control GUID file specifies which provider(s) to trace:

- `ctrl.guid.adonet`: ADO.NET only
- `ctrl.guid.snac`: SQL Server 2005
- `ctrl.guid.snac10`: SQL Server 2008
- `ctrl.guid.mdac`: Windows Data Access Components (formerly Microsoft Data Access Components) on Windows 7 only
- `ctrl.guid.wdac`: Windows Data Access Components on Windows 7 only

Data Access Tracing in SQL Server 2008
[Link](http://msdn.microsoft.com/en-us/library/cc765421(v=sql.100).aspx)

Control GUID Files
[Link](http://msdn.microsoft.com/en-us/library/cc765421(v=sql.100).aspx#B__Control_GUID_Files)
Appendix D
LINQ
Developing Windows Azure and Web Services

Updated 11th April 2014

LINQ
Contents

<table>
<thead>
<tr>
<th>Topic</th>
<th>Slide</th>
</tr>
</thead>
<tbody>
<tr>
<td>var</td>
<td>3</td>
</tr>
<tr>
<td>Generics</td>
<td>4</td>
</tr>
<tr>
<td>Initializers</td>
<td>6</td>
</tr>
<tr>
<td>Delegates</td>
<td>9</td>
</tr>
<tr>
<td>Lambda Expressions</td>
<td>11</td>
</tr>
<tr>
<td>LINQ</td>
<td>15</td>
</tr>
<tr>
<td>Extension Methods</td>
<td>22</td>
</tr>
<tr>
<td>Projection</td>
<td>33</td>
</tr>
<tr>
<td>Joining and Grouping</td>
<td>39</td>
</tr>
<tr>
<td>LINQ to XML</td>
<td>43</td>
</tr>
<tr>
<td>Tools for Learning</td>
<td>48</td>
</tr>
</tbody>
</table>

Exam Topic: Query data by using LINQ to Entities
- Query data by using LINQ operators (for example, project, skip, aggregate, filter, and join)
- Log queries
- Implement query boundaries (IQueryable vs. IEnumerable)

Exam Topic: Manipulate XML data structures
- LINQ to XML
D.3  

Inferred and Anonymous Types

• var infers type of local variables at compile time
  ```csharp
  var name = "Mark"; var cost = 2.99M; var width = GetWidth();
  ```

  • The compiler must be able to infer the type so you must assign an initial value, which can be returned from a method call

• Anonymous types can be inferred from an object initializer statement
  ```csharp
  var p = new { Name = "Bob", Age = 45 };
  ```

  • Warning! Instances of anonymous types are immutable in C#

D.4  

Generics  
What Are They?

• Define a template for a strongly-typed class
  • Actual type is created at compile time
  • Improves performance and reduces runtime errors
  • Commonly used with collections

• Example with constraints
  ```csharp
  public class Gen<TKey, TValue>
  where TKey : IComparable
  where TValue : Person, IComparable, new()
  {
    public TKey Key;
    public TValue Value;
  }

  var ga = new Gen<int, Employee>();
  ```
Generics

Constraints

When you define a generic class or method, you can apply restrictions to the kinds of types that can be used.

<table>
<thead>
<tr>
<th>Constraint</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>where T : struct</td>
<td>The type argument must be a value type</td>
</tr>
<tr>
<td>where T : class</td>
<td>The type argument must be a reference type</td>
</tr>
<tr>
<td>where T : new()</td>
<td>The type argument must have a public parameterless constructor; must come last</td>
</tr>
<tr>
<td>where T : &lt;base class name&gt;</td>
<td>The type argument must be or derive from the specified base class</td>
</tr>
<tr>
<td>where T : &lt;interface name&gt;</td>
<td>The type argument must be or implement the specified interface; multiple can be specified; can also be generic</td>
</tr>
<tr>
<td>where T : U</td>
<td>The type argument supplied for T must be or derive from the argument supplied for U</td>
</tr>
</tbody>
</table>

Constraints on Type Parameters (C# Programming Guide)

Generics

Generic Methods

Any type (including non-generic types) can have generic methods.

The generic applies to the type in the method signature.

```csharp
public class NonGen
{
    public void M1<T>(T Value)
    {
        // use Value
    }
}
```

• Specify the type when you call the method.

```csharp
var n = new NonGen();
n.M1<int>(23);
n.M1<string>("Fred");
```
D.7

Initializers
Object Initializers

Given this type

```csharp
public class Person
{
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public int Age { get; set; }
}
```

Initializing with C# 2.0 and earlier

```csharp
Person person = new Person();
person.FirstName = "John";
person.LastName = "Smith";
person.Age = 32;
```

Initializing with C# 3.0 and later

```csharp
var person = new Person
{ FirstName = "John", LastName = "Smith", Age = 32 };
```

D.8

Initializers
Array and Collection Initializers

Initialize an array of a simple type

```csharp
var names = new string[] { "Scott", "Bill", "Susanne" };
```

Initialize a collection of a complex type

```csharp
var people = new List<Person> {
    new Person
    { FirstName = "Scott", LastName = "Smith", Age = 32 },
    new Person
    { FirstName = "Bill", LastName = "Gates", Age = 50 },
    new Person
    { FirstName = "Susanne", LastName = "Smith", Age = 32 }
};
```

• The type must implement IEnumerable and have an appropriately-typed Add method
What Is A Delegate?

A delegate is a “type-safe function pointer”
- Like a reference it contains the memory address of something, but it points to a method rather than a data structure
- A delegate must match the signature of the method you want to call

```csharp
// method I want to call indirectly
int M1(string s)
{
    return s.Length;
}

// define with same signature
delegate int Del(string t);

// call method directly
int x = M1("Hello");

// call method indirectly
var d = new Del(M1);
int y = d("Goodbye");
```

Why are Delegates Useful?

- Treat methods as data
  - For example, create a queue of methods to call
- Anonymous delegates
  - Simplify code by removing need for defining a private method
    ```csharp
    Button1.Click += delegate { Debug.Write("Clicked"); }; 
    ```
- Can be invoked asynchronously using BeginInvoke
- Lambda expressions (used in LINQ)
  - Lambda expressions can be used in place of a delegate instance
- Loose-binding of types; cleaner type design
  - Foundation of events
A lambda expression is simply a *nameless* function
- Can be used wherever a delegate is valid

![Lambda expression diagram]

**Note:** Func is a generic delegate defined by Microsoft

---

A lambda expression syntax
- Inferring input types from a delegate
  ```csharp
delegate bool MyDelegate(int a, string b);
MyDelegate d = (x, y) => x == y.Length; // cannot use var with lambdas
```
- Explicitly defining input types
  ```csharp
  MyDelegate d = (int x, string y) => x == y.Length;
  ```
- Must use parentheses with zero input parameters; parentheses are only optional with one parameter
  ```csharp
  AnotherDelegate d2 = () => SomeMethod();
  ```
Lambda Expressions

Generic Delegates Often Used With Lambdas

- Func(TResult)
  - For lambda expressions with no inputs

- Func(T, TResult)
  - For lambda expressions with one input parameter

```csharp
Func<int, bool> myFunc = x => x == 5;
bool result1 = myFunc(4); // returns false
bool result2 = myFunc(5); // returns true
```

- Func(T1, T2, TResult)
- Func(T1, T2, T3, TResult) and so on

- Predicate(T): one input and always returns a Boolean

```csharp
Predicate<int> myFunc = x => x == 5;
```

Lambda Expressions

Lambda Statements and Multi-Line Expressions

- Lambda statements are nameless methods that return void

- Statement lambda syntax

```csharp
(parameters) => { statements; }
```
  - Statement lambdas cannot be used in expression trees and therefore cannot be used in LINQ queries

- Generic delegates for use with lambda statements
  - Action(T), Action(T1, T2), and so on

- Lambda expressions can also have multiple statements, but must explicitly return a value

```csharp
(input parameters) => { statements; return value; }
```
LINQ
What Is It?

Most databases understand SQL...

• …but to C# 2.0 and VB 8.0, an SQL statement is just a string
• LINQ integrates query syntax to a .NET language

LINQ is made up of three parts

• Providers for data sources (required)
  • LINQ to Objects, LINQ to SQL, LINQ to Entities, LINQ to XML, LINQ to SharePoint, LINQ to Amazon, and so on
• Extensions to the base class libraries (required)
• Extensions to the languages and compilers (optional)
  • C# keywords: from, select, orderby, and so on
  • VB keywords: From, Select, Order By, and so on

LINQ Provider Limitations

Theoretically, once you learn LINQ, you can query any LINQ provider...

• …but some LINQ providers have limitations

LINQ to Entities limitations

• This LINQ provider must eventually convert the expression tree created by your LINQ statements into Transact-SQL statements, so not all LINQ statements are fully supported, or might be implemented in ways that you do not like
  • Aggregate, SkipWhile, ToString, Zip, and others not supported

LINQ to Objects supports all features

• So use ToSomething<T> methods to get data from LINQ to Entities and then use LINQ to Objects on the result
LINQ
Enumerable and Queryable classes (System.Linq)

LINQ requires types to implement interfaces to support its features: IEnumerable(T) or IQueryable(T)

- If a type does not, IEnumerable has some extension methods: OfType(T), Cast(T) that can convert to the generic versions

LINQ uses extension methods defined by Enumerable and Queryable classes in the System.Linq namespace

- Importing the namespace allows the extension methods to be used on any type that implements IEnumerable(T) or IQueryable(T)

```csharp
using System.Linq;
```

LINQ IEnumerable and IQueryable interfaces

 IEnumerable means LINQ to Objects

- All data must be materialized locally before extension methods are applied

 IQueryable means LINQ to Entities (or LINQ to SQL)

- Data retrieval is deferred
- An expression tree is created and only when the query is enumerated (with foreach) or one of the ToXxx methods is called will the expression tree be converted into an SQL statement and executed to retrieve the appropriate data
- Use ToString (with DbContext) or ToTraceString (with ObjectContext) to see the T-SQL that will be executed
- Use a ToSomething method to retrieve data, then you can use LINQ to Objects without limitations
**LINQ**

**How to Create Extension Methods**

Extension methods allow you to add methods to a type without inheritance or recompilation

- Create a static class with a static method that uses the `this` keyword before a parameter to specify the type to extend

```csharp
namespace MyExtensions {
    public static class StringExtensions {
        public static bool IsValidEmailAddress(this string s) {
            // Implementation...
        }
    }
}
```

- Import the namespace containing the class

```csharp
using MyExtensions;
```

- Use the extension method on the type you extended

```csharp
if (email != null) {
    // Use IsValidEmailAddress method...
}
```

---

**LINQ**

**To create extension methods for LINQ**

- Create a static class that extends `IEnumerable<T>`
  - Returns a scalar (double)

```csharp
namespace System.Linq {
    public static class MyLinqExtensions {
        public static double MyAggregate<T>(this IEnumerable<T> input) {
            // Implementation...
        }
    }
}
```

- Returns a sequence (`IEnumerable<T>`) and therefore can be chained

```csharp
public static IEnumerable<T> MyProcessor<T>(this IEnumerable<T> input) {
    // Implementation...
}
```
### LINQ
Language-Level Support (Syntactic Sugar)

Instead of using extension methods and lambda expressions, C# and Visual Basic provide simplified syntax for queries like this

```csharp
var query = from name in names
    where name.StartsWith("A")
    orderby name
    select name; // select is required
```

The compiler turns the syntactic sugar above into calls to the extension methods with lambdas

```csharp
var query = names.Where(name => name.StartsWith("A")).OrderBy(name => name);
// Select is optional
.Select(name => name)
```

### LINQ Extension Methods
Where

Where takes a Func<TInput,TReturn> generic delegate as input (so we can use a lambda expression instead)

- Expression must have a single input parameter (of whatever type T in IEnumerable(T) is) and must return a boolean

```csharp
names.Where(n => n.StartsWith("A"))
```

- n would be a string

```csharp
string[] names = ...
... names.Where(n => n.StartsWith("A"));
```

- p would be a Person

```csharp
Person[] people = ...
... people.Where(p => p.Age > 18);
```
**LINQ Extension Methods**

**Where and Select Method Index Values**

Where (and Select) take Func<TInput, int, TReturn> delegate as input so you can filter based on index.

```csharp
var names = new string[]
    { "Fred", "George", "Mary", "Sally", "Emily", "Harry" }
var query = names.Where((name, index) => index % 2 == 0);
var results = query.ToArray();
list.ItemsSource = results; // WPF ListBox
```

Syntactic sugar does not seem to have an equivalent.

**LINQ Extension Methods**

**OrderBy**

OrderBy takes a Func<TInput, TKeySelector> generic delegate as input.

- Lambda expression must have a single input parameter (of whatever type T in IEnumerable(T) is) and can return any type.
- For names string array, we might want to order by the number of characters in each string entry.

```csharp
... names.OrderBy(name => name.Length);
```

- For names string array, we might want to order by the entire entry (which looks strange but is necessary due to the syntax!)

```csharp
... names.OrderBy(name => name);
```
**D. 25**

**LINQ Extension Methods**

**OrderBy and GroupBy**

† Usually you want to order first, then group

```csharp
var names = new string[]
{ "Fred", "George", "Gary", "Emily" };
var query = names.OrderBy(name => name)
 .GroupBy(name => name[0]);
```

† If you group first, you need to specify how to order the group’s key, not the original items

```csharp
var names = new string[]
{ "Fred", "George", "Gary", "Emily" }; var query = names.GroupBy(name => name[0])
 .OrderBy(group => group.Key);
```

**D. 26**

**LINQ Extension Methods**

**Chaining, Deferred Execution, and Materialization**

† Most extension methods return `IEnumerable(T)` so that they can be chained

```csharp
... names.Where(name => name.StartsWith("A"))
 .OrderBy(name => name);
```

• The extension methods will be processed in order
• Methods that return a sequence of values do not consume the target data until the query is enumerated (deferred execution)
• The query is not executed until enumerated over and will re-execute over the original data each time so will detect changes
• Materialize a copy with `ToArray`, `ToList`, and so on
• Methods that return a singleton value execute and consume the target data immediately
• Do not assume the entries in a sequence are ordered unless you explicitly specify
LINQ Extension Methods

Enumerable Static Methods

- `Empty<T>`
  ```csharp
  IEnumerable<Person> empty = Enumerable.Empty<Person>();
  ```

- `Range`
  ```csharp
  IEnumerable<int> numbers = Enumerable.Range(4, 3);
  /* 4
   * 5
   * 6 */
  ```

- `Repeat`
  ```csharp
  IEnumerable<string> madness = Enumerable.Repeat(
    "All work and no play makes Jack a dull boy.", 99);
  /*
   * All work and no play makes Jack a dull boy.
   * All work and no play makes Jack a dull boy.
   * All work and no play makes Jack a dull boy.
  */
  ```

LINQ Extension Methods

Non-Deferred, Scalar Return Value Methods

- `Aggregate()`
  ```csharp
  var db = new NorthwindEntities();
  var query = db.Products;
  decimal minPrice = query.Min(p => p.ListPrice);
  ```

- `All()`, `Any()`
  ```csharp
  if(names.Any(name => name.StartsWith("A")))
  ```

- `SequenceEqual()`
  ```csharp
  var query = db.Products;
  decimal minPrice = query.Min(p => p.ListPrice);
  ```

  ```csharp
  if(names.Any(name => name.StartsWith("A")))
  ```

  ```csharp
  var query = db.Products;
  ```

  ```csharp
  decimal minPrice = query.Min(p => p.ListPrice);
  ```

  ```csharp
  if(names.Any(name => name.StartsWith("A")))
  ```
LINQ Extension Methods

Non-Deferred, Single Item Return Value Methods

- First, FirstOrDefault, Last, LastOrDefault, ElementAt, ElementAtOrDefault, DefaultIfEmpty(def)
  - First, last or at index, or default if sequence is empty
  - N.B. Default for type, e.g. default(int) would be 0

```csharp
Person p = people.First(); // might throw exception
Person p = people.FirstOrDefault(); // might return null
```

- Single, SingleOrDefault
  - Returns a specific member of a sequence, or default value, or throws exception if more than one item in sequence

```csharp
Person q1 = people.Where(p => p.ID == 123).Single();
Person q2 = people.SingleOrDefault(p => p.ID == 123);
```

LINQ Extension Methods

Deferred, Multiple Item Return Value Methods

- Where
  - Filters the sequence by specific criteria

```csharp
IQueryable: OrderBy, OrderByDescending, Reverse
IOrderedEnumerable: ThenBy, ThenByDescending
  - Ascending and descending chained sorts, or reverse the order
```

- Skip, SkipWhile
  - Skips n members, or while lambda expression returns true

- Take, TakeWhile
  - Takes n members, or while lambda expression returns true

- Distinct, Except, Intersect, Concat, Union, Zip
  - Sequence where members are distinct, differ, match, all, or zipped 1-1, 2-2, 3-3, and so on
**LINQ Extension Methods**

**AsSomething Conversions**

- **AsEnumerable<T>()**
  - Convert IEnumerable to IEnumerable<T>
  - “Execute” a query without creating a local collection

- **AsQueryable<T>()** Convert IQueryable to IQueryable<T>

- **AsParallel()**
  - PLINQ is designed to exploit opportunities for parallelization, however, not all queries benefit
  - It partitions the data source into segments, and then executes the query on separate threads on multiple processors
  - The overhead can be more expensive than the speedup so PLINQ may decide to execute some or all of the query sequentially

```csharp
var q = from cust in customers.AsParallel()
        .WithExecutionMode(ParallelExecutionMode.ForceParallelism)
```

**ToList and ToArray Conversions**

- **ToList and ToArray:** return flat collection of results

- **ToDictionary**
  - One-to-one mapping of keys to objects
  - Requires a lambda to define property to use for the key

```csharp
Dictionary<string, Product> products =
    db.Products.ToDictionary(p => p.ProductName);
```

- **ToLookup**
  - One-to-many mapping of keys to collections
  - Requires a lambda to define property to use for the key

```csharp
ILookup<string, Product> products =
    db.Products.ToLookup(p => p.Category.CategoryName);
IEnumerable<Product> bikes = products["Bike"]; // we could use var but it is better to document the actual type
D.33

Projection

Primitive Results

◆ p is a Product

```csharp
var db = new NorthwindEntities();

// syntactic sugar
var query = from p in db.Products
            select p;

List<Product> results = query.ToList();
```

◆ p.ProductName is a string

```csharp
// syntactic sugar
var query = from p in db.Products
             select p.ProductName;

List<string> results = query.ToList();
```

D.34

Projection

Projecting into Types

◆ A type that defines a subset of product information

```csharp
public class ProductSubset
{
    public string Name;
    public decimal Price;
}
```

◆ Project into instances of this type using either query syntax or Select extension method

```csharp
// syntactic sugar
var query = from p in db.Products
            select new ProductSubset
            {
                Name = p.ProductName,
                Price = p.ListPrice
            };

// extensions and lambdas
var query = db.Products
            .Select(p => new ProductSubset
            {
                Name = p.ProductName,
                Price = p.ListPrice
            });
```

◆ Materialize results

```csharp
List<ProductSubset> results = query.ToList();
```
Projecting into Anonymous Types

**Project into instances of an anonymous type using either query syntax or Select extension method**

- Note: you can assign a new name for the property e.g. Name or reuse its existing name e.g. ListPrice

```csharp
// syntactic sugar
var query = from p in db.Products
             select new {
                Name = p.ProductName,
                p.ListPrice
            };  

// extensions and lambdas
var query = db.Products
            .Select(p => new {
                Name = p.ProductName,
                p.ListPrice
            });
```

**Materialize results and store in inferred variable**

```csharp
// must use var
var results = query.ToList();
```

---

SelectMany Example 1

**SelectMany projects each element of a sequence to an IEnumerable<T> and flattens the resulting sequences into one sequence**

```csharp
var nameList = new List<string> {  
    "Matt", "Adam", "John", "Peter",  
    "Owen", "Steve", "Richard", "Chris"  
};

var names1 = nameList.Where(n => n.Length == 4)  
    .Select(n => n);  

var names2 = nameList.Where(n => n.Length == 4)  
    .SelectMany(n => n);
```
**Projection SelectMany Example 2**

We want to create a single sequence of words from a sequence of sentences

```csharp
var sentences = new List<string> { 
    "Bob is quite excited.",
    "Jim is very upset."
};
```

**Using SelectMany**

```csharp
// extensions and lambdas
var words = sentences
    .SelectMany(s => s.TrimEnd('.').Split(' '));
```

**Using “from chaining”**

```csharp
// syntactic sugar
var words = from s in sentences
             from w in s.TrimEnd('.').Split(' ') 
             select w;
```

**Projection SelectMany Example 3**

We want to get a flat list of products from categories

```csharp
var db = new NorthwindEntities();
```

**Using SelectMany**

```csharp
var query = db.Categories.SelectMany(c => c.Products);
```

**Using LINQ ‘from’ chaining**

```csharp
// syntactic sugar
var query = from c in db.Categories from p in c.Products select p;
```

Select would give us this:
Joining and Grouping

Joining with Query Syntax

- **Joining by using where...==**

```csharp
// syntactic sugar
var query = from p in db.Products
             from c in db.Categories
             where p.CategoryID == c.CategoryID
             ...
```

- **Joining by using join...on...equals**

```csharp
// syntactic sugar
var query = from p in db.Products
             join c in db.Categories
             on p.CategoryID equals c.CategoryID
             ...
```

- Both are equivalent to using the Join extension method (see next slide)

Joining and Grouping

Joining with Join extension method

- **Join between Categories and Products (1-many)**
  - The first lambda chooses property on Category (c) to join on
  - The second lambda chooses property on Product (p) to join on
  - The third lambda expression projects the results, merging properties from each Category entity (cat) and its matching Product entity (prod)

```csharp
var query = db.Categories.Join(db.Products,
                                c => c.CategoryID, p => p.CategoryID,
                                (cat, prod) => new
                                {
                                   CategoryID = cat.CategoryID,
                                   CategoryName = cat.CategoryName,
                                   ProductName = prod.ProductName
                                });
```

- One “row” returned for each product (77 in Northwind)
Joining and Grouping

Joining with GroupJoin extension method

GroupJoin between Categories and Products (1-many)

- The first lambda chooses property on Category (c) to join on
- The second lambda chooses property on Product (p) to join on
- The third lambda expression projects the results, merging properties from each Category entity (cat) and its matching Product entities (products)

```csharp
var query = db.Categories.GroupJoin(db.Products, 
c => c.CategoryID, p => p.CategoryID, 
(cat, products) => new 
{ 
    CategoryID = cat.CategoryID, 
    CategoryName = cat.CategoryName, 
    NumberOfProducts = products.Count() 
});
```

- One “row” returned for each category (8 in Northwind)

Joining and Grouping

Grouping with Query Syntax

Groups return List(IGrouping<TKey, TElement))

```csharp
var query = from p in db.Products 
group p by p.Color into colourgroup 
select colourgroup; 
List<IGrouping<string, Product>> results = query.ToList(); 
foreach (IGrouping<string, Product> group in results) 
{ 
    listBox1.Items.Add(group.Key); // Red, Blue, etc. 
    foreach (Product prod in group) 
    { 
        listBox1.Items.Add(prod.ProductName); 
    } 
}
```
LINQ to XML
Generating an XML File from LINQ-able Entities

“products” could be an entity set or collection

```csharp
using System.Xml.Linq;

var xml = new XElement("Products",
    from p in products
    select new XElement("Product",
        new XElement("ProductID", p.ProductID),
        new XElement("Name", p.Name),
        new XElement("Color", p.Color),
        new XElement("ListPrice", p.ListPrice),
        new XElement("Size", p.Size)));

xml.Save(productFileName);
```

Use `XAttribute` for attributes

```
<Products>
  <Product>
    <ProductID>1</ProductID>
    <Name>Chai</Name>
    ...
  </Product>
  <Product>
    ...
  </Product>
</Products>
```

LINQ to XML
Generating a Collection from an XML File

Convert each child XML element into an entity

```csharp
using System.Xml.Linq;

var doc = XDocument.Load(productFileName);
var query = from product in doc.Descendants("Product")
    select new Product
    {
        ProductID = (int)product.Element("ProductID").Value,
        Name = product.Element("Name").Value,
        Color = product.Element("Color").Value,
        ListPrice = product.Element("ListPrice").Value,
        Size = product.Element("Size").Value
    };

List<Product> products = query.ToList<Product>();
```
Imagine that you need to convert this XML into a collection of Car objects:

```xml
<cars>
  <car name="Toyota Coupe">
    <profile name="Vendor" value="Toyota"/>
    <profile name="Model" value="Celica"/>
    <profile name="Doors" value="2"/>
    <support name="Racing" value="yes"/>
    <support name="Towing" value="no"/>
  </car>
  <car name="Honda Accord Aerodec">
    <profile name="Vendor" value="Honda"/>
    <profile name="Model" value="Accord"/>
    <profile name="Doors" value="4"/>
    <support name="Racing" value="no"/>
  </car>
</cars>
```

```csharp
public class Car
{
    public string Name;
    public string Vendor;
    public string Model;
    public int Doors;
    public bool Racing;
}
```

The `let` keyword is used in LINQ syntactic sugar to define local variables. Here's an example:

```csharp
var xd = XDocument.Load("cars.xml");
var query = from car in xd.Root.Elements("car")
            select new
                {
                    Name = car.Attribute("name").Value,
                    Vendor = profiles.Single(prof => prof.Name == "Vendor").Value,
                    Model = profiles.Single(prof => prof.Name == "Model").Value,
                    Doors = int.Parse(profiles.Single(prof => prof.Name == "Doors").Value,
                    Racing = supports.Single(sup => sup.Name == "Racing").Value == "yes"
                };
List<Car> cars = query.ToListCar<>();
```

[The Linq "let" keyword](http://www.codethinked.com/the-linq-quot3bletquot3b-keyword)
Tools for Learning
LINQPad 4 and 101 LINQ Samples

**LINQPad**
- Interactively query databases using LINQ
- 500 examples

LINQPad
http://www.linqpad.net

101 LINQ Samples – C#  
http://code.msdn.microsoft.com/101-LINQ-Samples-3fb9811b

Lambda Expressions (C#)

LINQ Query Expressions (C#)
You are unlikely to get questions about features of EF4 and ObjectContext in the exam although Visual Studio can generate the old ObjectContext and EntityObject classes or the inner ObjectContext can be accessed by casting a DbContext as an IObjectContextAdapter like this:

```csharp
using System.Data.Entity.Infrastructure;

var db = new NorthwindEntities(); // DbContext
var oc = ((IObjectContextAdapter)db).ObjectContext;
```
Entity Data Models
Vocabulary Overview

<table>
<thead>
<tr>
<th>AWModel.edmx</th>
<th>AWModel.edmx.cs</th>
</tr>
</thead>
<tbody>
<tr>
<td>“Database”</td>
<td>&lt;EntityContainer Name=&quot;AdventureWorksEntities&quot;&gt; class AdventureWorksEntities : ObjectContext (or DbContext)</td>
</tr>
<tr>
<td>“Table”</td>
<td>&lt;EntitySet Name=&quot;Contacts&quot;&gt; DbSet&lt;Contact&gt; Contacts (or DbSet&lt;Contact&gt; Contacts)</td>
</tr>
<tr>
<td>“Row”</td>
<td>&lt;EntityName Name=&quot;Contact&quot;&gt; class Contact : EntityObject (or class Contact)</td>
</tr>
<tr>
<td>“Column”</td>
<td>&lt;Property Name=&quot;ContactID&quot;&gt; [EdmScalarPropertyAttribute(…)] Int32 ContactID …</td>
</tr>
<tr>
<td>“Relationship”</td>
<td>&lt;NavigationProperty Name=&quot;Orders&quot; … &lt;NavigationProperty Name=&quot;Contact&quot; … &lt;Association Name=&quot;FK_Orders_Contact_ContactID&quot;&gt; [EdmRelationshipNavigationPropertyAttribute(…)] EntityCollection&lt;Order&gt; Orders [EdmRelationshipNavigationPropertyAttribute(…)] Contact Contact</td>
</tr>
</tbody>
</table>

VS2010 .edmx designer-generated code creates classes that use ObjectContext
VS2012 .edmx designer-generated code creates classes that use DbContext

Entity Data Models
Entity Properties

Scalar property in class

[EdmScalarPropertyAttribute(
   EntityKeyProperty=false, IsNullable=false)]
public global::System.String CategoryName {

Navigation property in EntityObject class

[EdmRelationshipNavigationPropertyAttribute(
   "NorthwindModel", "FK_Products_Categories", "Products")]
public EntityCollection<Product> Products {
Entity Data Models
Defining an Independent Association

Entities are related through associations

- Must specify the entities that are involved in the relationship and the possible number of entities at each end, which is known as the multiplicity (one (1), zero or one (0..1), or many (*))

```
<Association Name="CustomerOrders">
  <End Type="ExampleModel.Customer" Role="Customer" Multiplicity="1" />
  <OnDelete Action="Cascade" />
</End>
<End Type="ExampleModel.Order" Role="Order" Multiplicity="*" />
</Association>
```

- “Independent Associations” like the one above do not use foreign key constraints

Entity Data Models
Defining a Foreign Key Association

With foreign keys exposed, the relationship between the entities is managed with a ReferentialConstraint element.

A corresponding AssociationSetMapping element is not necessary to map this association to the data source.

```
<Association Name="CustomerOrders">
  <End Type="ExampleModel.Customer" Role="Customer" Multiplicity="1" />
  <OnDelete Action="Cascade" />
</End>
<End Type="ExampleModel.Order" Role="Order" Multiplicity="*" />
<ReferentialConstraint>
  <Principal Role="Customer">
    <PropertyRef Name="Id" />
  </Principal>
  <Dependent Role="Order">
    <PropertyRef Name="CustomerId" />
  </Dependent>
</ReferentialConstraint>
</Association>
```
E.7

ObjectContext uses ObjectSet(T) for the entity sets

```csharp
public class AWContext : ObjectContext { ... public ObjectSet<Product> Products { ... }
```

ObjectSet(T) inherits from ObjectQuery(T) and implements some common interfaces

```csharp
public class ObjectSet<TEntity> : ObjectQuery<TEntity>, IObjectSet<TEntity>, IQueryable<TEntity>, IEnumerable<TEntity>, IQueryable, IEnumerable
where TEntity : class
```

Cast your LINQ query to access ObjectQuery features such as ToTraceString to see the T-SQL (or whatever)

```csharp
string TSQL = ((ObjectQuery)query).ToTraceString();
```

Related links:

E.8

EntityCollection(T), EntityReference(T)

Navigation properties in designer-generated code

```csharp
public class Customer : EntityObject { ... public EntityCollection<Order> Orders { ... }
```

```csharp
public class Order : EntityObject { ... public EntityReference<Customer> CustomerReference { ... }
```

Useful members

- IsLoaded: returns true if collection or reference is loaded
- Load(MergeOption): Loads related object(s)
- CreateSourceQuery(): Returns an object query that returns the same object(s) that exist in the current collection or reference
- EntityCollection(T).Count: the number of entities in collection
- EntityReference(T).Value: the actual entity

Related links:
### POCO

#### Requirements for ObjectContext

<table>
<thead>
<tr>
<th>Feature</th>
<th>Requirements</th>
</tr>
</thead>
</table>
| To use POCO with EF                  | 1. Class name must match  
                                           2. Property names and data types must match |
| To create either type of proxy       | 1. Class must be public and NOT sealed or abstract  
                                           2. Class must have a public or protected no-argument constructor  
                                           3. Class must NOT implement IEntityWithChangeTracker or IEntityWithRelationships  
                                           4. Call ObjectContext.CreateObject<T> or ObjectSet<T>.CreateObject |
| To create a lazy-loading proxy       | 1. Navigation properties must have a public, virtual, and non-sealed get accessor |
| To create a change-tracking proxy    | 1. Properties must have public, non-sealed, and virtual get and set accessors  
                                           2. Navigation property for “many” end of a relationship must implement ICollection<T> |

### POCO

#### Lazy-Loading without a Proxy

🌟 If the class does NOT support lazy-loading, you will have to explicitly call LoadProperty before reading any navigation properties

```csharp
// or oc.LoadProperty(order, "Products");
oc.LoadProperty(order, o => o.Products);
var prods = order.Products;
```

🌟 You cannot use Load because your navigation property would not be of type EntityReference<T> or EntityCollection<T> in a POCO
POCO Change Tracking

- POCO supports two methods of change tracking
  - Proxy creation (aka “instant”) change tracking
  - Snapshot change tracking

- Instantiate a POCO using the `CreateObject` method(s)

  - If it does not support proxy creation then it takes a snapshot when the entity is attached/added to the `ObjectContext`

```csharp
Customer c = oc.CreateObject<Customer>();
```

- For snapshot change tracking you must call `DetectChanges` before saving to the data source

```csharp
oc.DetectChanges();
oc.SaveChanges();
```

Retrieving Data

Efficient Retrieval of a Single Entity

- Executing a query will always hit underlying database

```csharp
var query = from c in oc.Contacts where c.ContactID == 3 select c;
Contact contact = query.Single();
// do something with contact
```

- Use `TryGetObjectByKey` to check the `ObjectContext` cache first, then the database only if necessary

```csharp
object contact = null;
var key = new EntityKey(
"AEntities.Contacts", "ContactID", 3);
if(oc.TryGetObjectByKey(key, out contact)) {
// do something with contact
```

ObjectContext.TryGetObjectByKey Method
You can create an instance of EntityKey by using the class constructor:

```csharp
var key = new EntityKey(
    "AWEntities.Contacts", "ContactID", 3);
```

Or you can use the CreateEntityKey method ofObjectContext to generate a key for a specific object:

```csharp
Contact c = ...;
var key = oc.CreateEntityKey("Contacts", c);
```

---

**ExecuteFunction**
- Executes a stored procedure or function that is mapped in the conceptual model

**ExecuteStoreCommand**
- Executes an arbitrary command directly against the data source using the existing connection

**ExecuteStoreQuery**
- Executes a query directly against the data source that returns a sequence of typed results

**Translate**
- Translates a DbDataReader that contains rows of entity data to objects of the requested entity type
Retrieving Data

CreateQuery\(<T>\)( ) method

纪律

 presumed ObjectQuery\(<T>\) by using the specified
 query string

```csharp
var query = oc.CreateQuery<Contact>(
    "SELECT VALUE c FROM AWEntities.Contacts
     AS c WHERE c.FirstName = @fn",
    new ObjectParameter("fn", "John"));
```

Alternatively, create a new ObjectQuery\(<T>\)

```csharp
var query = new ObjectQuery<Contact>(
    "SELECT VALUE c FROM AWEntities.Contacts
     AS c WHERE c.FirstName = @fn", oc);
query.Parameters.Add(
    new ObjectParameter("fn", "John"));
```

Modifying Data

How to Create and Add Entities

纪律

 The EDM defines a static method for every entity to
 make it easy to create new instances

- Forces you to pass a list of all non-nullable property values
  instead of having to set each property individually

```csharp
Contact c = Contact.CreateContact("John", "Smith", ...);
```

You must mark the entity as added using AddObject

- `ObjectContext.AddObject("entityset", entity)`
- `ObjectSet< T >.AddObject(entity)`

For related entities

- Call Add method on navigation property e.g. `c.Orders.Add(o)`
- Set navigation property e.g. `o.Customer = c`
Modifying Data

**MergeOption (only applies when loading)**

- **AppendOnly (default)**
  - Objects that do not exist in the object context are attached
  - Existing objects are *unaffected*

- **OverwriteChanges**
  - Objects that do not exist in the object context are attached
  - Existing objects are *overwritten*

- **PreserveChanges**
  - Objects that do not exist in the object context are attached
  - Existing objects that are Unchanged are *overwritten*
  - Existing objects that are Modified are *preserved*

- **NoTracking:** entities are Detached

---

Modifying Data

**ObjectContext methods**

- **AddObject(string entitySet, object)**
  - Adds an object to the object context (marked as Added)

- **Attach(IEntityWithKey) and Detach(object)**
  - Attaches an object or object graph to the object context when the object has an entity key

- **AttachTo(string entitySet, object)**
  - Call AttachTo on the ObjectContext to attach the object to a specific entity set in the object context or if the object has a null (Nothing in Visual Basic) EntityKey value

- **ApplyCurrentValues(string entitySet, object) and ApplyOriginalValues(string entitySet, object)**
  - Copies the scalar values from the supplied object into the object in the ObjectContext that has the same key
E.19 Modifying Data
ObjectStateEntry

Maintains state and key information for objects and relationships and change tracking for object properties

```csharp
ObjectStateEntry stateEntry = context.ObjectStateManager.GetObjectStateEntry(((IEntityWithKey)order).EntityKey);

CurrentValueRecord rec1 = stateEntry.CurrentValues;
string oldPO = (string)rec1.GetValue(rec1.GetOrdinal("PurchaseOrderNumber"));

order.PurchaseOrderNumber = "12345";
string newPO = (string)rec1.GetValue(rec1.GetOrdinal("PurchaseOrderNumber"));

IEnumerable<string> modifiedFields = stateEntry.GetModifiedProperties();

foreach (string s in modifiedFields)
    Console.WriteLine("Modified field name: {0}\n" + "Old Value: {1}\n New Value: {2}" , s, oldPO, newPO);
```

E.20 Modifying Data
ObjectContext.SaveChanges

SaveChanges()

- Persists all updates to the data source and resets change tracking in the object context (i.e. it calls AcceptAllChanges() automatically)
- Returns number of objects in an Added, Modified, or Deleted state when SaveChanges was called

SaveChanges(SaveOption)

- None, AcceptAllChangesAfterSave (default), DetectChangesBeforeSave
- Use to control the automatic call to AcceptAllChanges() or enable an automatic call to DetectChanges()
- Flag so can be combined

SaveChanges(bool) is obsolete
Modifying Data

OptimisticConcurrencyException

- When invoking SaveChanges, catch this exception
- Invoke Refresh on context to resolve conflict
  - RefreshMode: StoreWins, ClientWins
  - Pass entity or IEnumerable
- Invoke SaveChanges again

```csharp
try {
    context.SaveChanges();
} catch (OptimisticConcurrencyException ex) {
    context.Refresh(RefreshMode.ClientWins, orders);
    context.SaveChanges();
}
```

Performance Considerations

<table>
<thead>
<tr>
<th>Operation</th>
<th>Frequency</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Loading metadata</td>
<td>Once per AppDomain</td>
<td>You can manually open and close connections</td>
</tr>
<tr>
<td>Opening connection</td>
<td>As needed</td>
<td></td>
</tr>
<tr>
<td>Generating views</td>
<td>Once per AppDomain</td>
<td>Can pre-generate</td>
</tr>
<tr>
<td>Preparing the query</td>
<td>Once for each unique query</td>
<td>Can pre-compile</td>
</tr>
<tr>
<td>Executing the query</td>
<td>Once for each query</td>
<td></td>
</tr>
<tr>
<td>Loading and validating types</td>
<td>Once for each ObjectContext instance</td>
<td>Types are validated against the model</td>
</tr>
<tr>
<td>Tracking</td>
<td>Once for each object returned*</td>
<td>MergeOption.NoTracking disables this step</td>
</tr>
<tr>
<td>Materializing</td>
<td>Once for each object returned*</td>
<td></td>
</tr>
</tbody>
</table>

*Not required when using EntityDataReader
Both LINQ to Entities and LINQ to SQL have a CompiledQuery class with a Compile method

- ... but the concept of compilation doesn't really make sense for LINQ to Object queries
- In LINQ to Objects, a Where method takes a Func<T, bool> as opposed to an Expression<Func<T, bool>>
- In LINQ to Objects, the C# compiler compiles the lambda expression down to an anonymous method and generates the IL at compile time and passes a delegate to that Where method
- In other LINQ flavors, like LINQ to Entities, the lambda is not compiled to IL; Instead, the compiler builds up an expression tree object out of the lambda expression and passes the expression tree to LINQ methods

Is it possible to compile a query for LINQ to Objects?
http://stackoverflow.com/questions/2649874/is-it-possible-to-compile-a-query-for-linq-to-objects

EF 4.1
What is Microsoft ADO.NET Entity Framework 4.1?

- aka “Magic Unicorn Edition” for VS2010 and later
- EF 4.1 introduces two new features
  - The DbContext API is a simplified abstraction overObjectContext and a number of other types
  - Code First is a new development pattern that provides an alternative to the Database First and Model First patterns
- Code First is focused around defining your model using .NET classes
  - These classes can then be mapped to an existing database or be used to generate a database schema
  - Additional configuration can be supplied using Data Annotations or via a fluent API

EF 4.1 Released
Appendix F
Firebrand Sample Services
Developing Windows Azure and Web Services

Updated 11th April 2014

Firebrand Sample Services
Contents

<table>
<thead>
<tr>
<th>Topic</th>
<th>Slide</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overview</td>
<td>3</td>
</tr>
<tr>
<td>The Service</td>
<td>5</td>
</tr>
<tr>
<td>The Hosts - Console</td>
<td>8</td>
</tr>
<tr>
<td>The Clients</td>
<td>12</td>
</tr>
<tr>
<td>Exposing Data</td>
<td>21</td>
</tr>
<tr>
<td>The Hosts - IIS</td>
<td>24</td>
</tr>
<tr>
<td>Faults</td>
<td>27</td>
</tr>
<tr>
<td>Duplex</td>
<td>30</td>
</tr>
<tr>
<td>REST</td>
<td>34</td>
</tr>
<tr>
<td>Discovery</td>
<td>35</td>
</tr>
<tr>
<td>Router</td>
<td>40</td>
</tr>
<tr>
<td>Sessions</td>
<td>46</td>
</tr>
</tbody>
</table>
Overview

Clients and Services

- **ClientUsingWSDL**
  - SampleServiceClient

- **ClientUsingChannelFactory / ManualContract**
  - Dynamic Proxy

- **ClientUsingDuplex**
  - CalculatorClient
    - Callback

- **IISRouter on port 803**
  - RoutingService
    - “Hello”
    - “Bye”

- **SampleService**
  - IISHost on port 802

- **SampleService**
  - ConsoleHost on port 801

Overview

Start with a Blank Solution

Add a new **Blank Solution** named Firebrand

This solution will eventually contain these projects

- **Services**: S1.DataContracts, S2.ServiceContracts, S3.Implementation
- **Hosts**: H1.ConsoleHost, H2.IISHost, H3.IISRouter

Download complete solution
The Service

Define the Data Contract(s)

- Add a new Class Library
  - Name: S1.DataContracts
  - Add a reference to System.Runtime.Serialization
  - Rename Class1.cs to Widget.cs
  - Change the namespace to Firebrand

```csharp
namespace Firebrand {
    using System.Runtime.Serialization;
    [DataContract]
    public class Widget {
        [DataMember]
        public string Name { get; set; }
        [DataMember]
        public bool Active { get; set; }
    }
}
```

The Service

Define the Service Contract(s)

- Add a new Class Library
  - Name: S2.ServiceContracts
  - Add references to the S1.DataContracts project and System.ServiceModel assembly
  - Rename Class1.cs to ISampleService.cs

```csharp
namespace Firebrand {
    using System.ServiceModel;
    [ServiceContract]
    public interface ISampleService {
        [OperationContract]
        Widget ModifyWidget(Widget w);
        [OperationContract]
        int AddNumbers(int a, int b);
    }
}
```
The Service
Implement the Service

Add a new Class Library
- Name: S3.Implementation
- Add references to S1.DataContracts and S2.ServiceContracts
- Rename Class1.cs to SampleService.cs

```csharp
namespace Firebrand {
    public class SampleService : ISampleService {
        public Widget ModifyWidget(Widget w) {
            if (w != null) {
                if (w.Active) w.Name += "A";
            }
            return w;
        }
        public int AddNumbers(int a, int b) {
            return a + b;
        }
    }
}
```

The Hosts
Create a Console Host (1/2)

Add a new Console Application
- Name: H1.ConsoleHost, change namespace to Firebrand
- Add references to System.ServiceModel, S1.DataContracts, S2.ServiceContracts, S3.Implementation

```csharp
using System;
using System.Net;
using System.ServiceModel;
using System.ServiceModel.Description;

var host = new ServiceHost(typeof(SampleService), new Uri(
    string.Format("http://{0}:801/Sample", Dns.GetHostName())));
host.Description.Behaviors.Add(
    new ServiceMetadataBehavior { HttpEnabled = true });
host.Description.Behaviors.Find<ServiceDebugBehavior>()
    .IncludeExceptionDetailInFaults = true;
host.Open();
Console.WriteLine("{0} is listening on these endpoints:",
    host.Description.Name);
Console.WriteLine();
```
The Hosts
Create a Console Host (2/2)

Enumerate the endpoints that have been configured, including automatic (aka default) endpoints

- Wait for the user to press ENTER, then close the host

```csharp
foreach (var ep in host.Description.Endpoints)
{
    Console.WriteLine(" Endpoint: " + ep.Name);
    Console.WriteLine(" Address: " + ep.Address);
    Console.WriteLine(" Binding: " + ep.Binding);
    Console.WriteLine(" Contract: " + ep.Contract.ContractType);
    Console.WriteLine();
}
Console.WriteLine("Press ENTER to close service hosts.");
Console.ReadLine();
Console.WriteLine("Closing hosts... please wait.");
host.Close();
Console.WriteLine("Closed.");
```

The Hosts
Start the Console Host

- Ctrl+F5 to start without attaching the debugger

If you see this exception, restart Visual Studio as Administrator

```xml
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions xmlns:xsd="http://www.w3.org/2001/XMLSchema"
     xmlns:wss="http://www.w3.org/2004/01/wss-sec" xmlns:soap12="http://www.w3.org/2003/05/soap-envelope"
    <wsdl:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
                 xmlns="http://localhost:801/Sample/WS-Security/">
```
**The Hosts**

**Test the Console Host**

Run the WCF Test Client

- File-Add Service...
- Enter URL: http://localhost:801/Sample
- Double-click Config File and review it (or right-click to edit it)
- Double-click AddNumbers, fill parameters, click Invoke
- Click XML tab to see SOAP messages

![WCF Test Client](image)

**Create Explicit WS-* and Custom Binding Endpoints**

In H1.ConsoleHost, in App.config

- Define a pair of binding configurations and endpoints to use them

```
<system.serviceModel>
  <bindings>
    <wsHttpBinding>
      <binding name="FBWSNoSecurity"/>
      <security mode="None"/>
    </wsHttpBinding>
    <customBinding>
      <binding name="FBBinaryOverHttp">
        <binaryMessageEncoding/>
        <httpTransport/>
      </customBinding>
  </bindings>
  <services>
    <service name="Firebrand.SampleService">
      <endpoint address="" binding="customBinding" bindingConfiguration="FBBinaryOverHttp" contract="Firebrand.ISampleService"/>
      <endpoint address="ws" binding="wsHttpBinding" bindingConfiguration="FBWSNoSecurity" contract="Firebrand.ISampleService"/>
    </service>
  </services>
</system.serviceModel>
```
The Clients
Create a Client with a Service Reference

Add a new **WPF Application** (clients can be anything)
- Name: C1.ClientUsingWSDL
- Ensure **H1.ConsoleHost** is running
- **Add Service Reference** and type `http://localhost:801/Sample`
  - Expand the service to download the WSDL, click the interface
- **Namespace**: `SampleServiceReference`, click OK

![Add Service Reference](image)

The Clients
Code the Client to Use the Proxy (Add Numbers)

MainWindow: add two text boxes named **A** and **B**, a label named **Answer**, and a button named **AddNumbers**

```csharp
using C1.ClientUsingWSDL.SampleServiceReference;

try {
    var proxy = new SampleServiceClient("WSHttpBinding_ISampleService");
    int a = int.Parse(A.Text);
    int b = int.Parse(B.Text);
    Answer.Content = proxy.AddNumbers(a, b); // or use async & await
} 
catch (Exception ex) {
    Answer.Content = await proxy.AddNumbersAsync(a, b);
} 

private async void AddNumbers_Click(object sender, RoutedEventArgs e) {
    Answer.Content = ex.Message;
    if you use await you must also apply async to method
} 
```
The Clients

**Code the Client to Use the Proxy (Modify Widget)**

- Add a check box named `Active`, a text box named `NameTextBox`, a label named `ModifiedName`, and a button named `ModifyWidget`

```csharp
try
{
    var proxy = new SampleServiceClient("CustomBinding_ISampleService");
    var w = new Widget
    {
        Name = NameTextBox.Text,
        Active = Active.IsChecked.Value
    };
    w = await proxy.ModifyWidgetAsync(w);
    ModifiedName.Content = "Modified name: " + w.Name;
    proxy.Close();
}
catch (Exception ex)
{
    ModifiedName.Content = "Exception: " + ex.Message;
}
```

---

The Clients

**Reuse Data Contracts**

- The Widget class is currently re-created on the client by the service reference
  - Underneath `SampleServiceReference` review the `Reference.cs` file (you will need to Show All Files)

- To reuse the original data contract
  - Add a reference to `S1.DataContracts`
  - Right-click `SampleServiceReference` and choose Update Service Reference

- Now the Widget class is not re-created
  - Import the Firebrand namespace to use it

```csharp
using Firebrand;
```
The Clients
Read from <client> endpoints

*When you have multiple client endpoints the proxy constructor must have the name of the endpoint set*

- Add a reference to System.Configuration
- Add a ComboBox named Endpoints
- Handle the Window_Loaded event

```csharp
using System.ServiceModel.Configuration;
using System.Configuration;

var client = (ClientSection)ConfigurationManager.GetSection("system.serviceModel/client");
Endpoints.Items.Clear();
foreach (ChannelEndpointElement ep in client.Endpoints)
    Endpoints.Items.Add(ep.Name);
if (Endpoints.Items.Count > 0)
    Endpoints.SelectedIndex = 0;
var proxy = new SampleServiceClient(Endpoints.Text);
```

The Clients
Create a Client with a Channel Factory

*Add a new WPF Application*

- Name: C2.ClientUsingChannelFactory
- Add references to System.ServiceModel, S1.DataContracts and S2.ServiceContracts projects
- Draw a check box, text box, label, and button

```csharp
var address = new EndpointAddress("http://localhost:801/Sample/ws");
var binding = new WSHttpBinding
    { Security = new WSHttpSecurity { Mode = SecurityMode.None } };
var factory = new ChannelFactory<ISampleService>(binding, address);
var proxy = factory.CreateChannel();
var w = new Widget
    { Name = NameTextBox.Text,
      Active = Active.IsChecked.Value
    };
w = proxy.ModifyWidget(w);
ModifiedName.Content = "Modified: " + w.Name;
((ICommunicationObject)proxy).Close();
```
The Clients
Create a Client with a Combined Interface

Add an interface file: `ISampleServiceWithComms.cs`

```csharp
public interface ISampleServiceWithComms : ISampleService, ICommunicationObject {
}
```

Change your channel factory code to use the new interface instead of `ISampleService`
- When creating the factory
  ```csharp
  var factory = new ChannelFactory<ISampleServiceWithComms>(binding, address);
  ```
- When closing the proxy
  ```csharp
  proxy.Close();
  ```

The Clients
Create a Client with a Locally-Defined Contract

Add a new WPF Application

```csharp
using System.ServiceModel;
    [ServiceContract( Name = "ISampleService")]
    public interface ISampleServiceSubset : ICommunicationObject {
        [OperationContract]
        int AddNumbers(int a, int b);
    }
    // do not need to include all operations
}
```

```csharp
<client>
  <endpoint name="ep" address="http://localhost:801/Sample/ws"
         binding="wsHttpBinding" bindingConfiguration="FBWSNoSecurity"
</client>

var factory = new ChannelFactory<ISampleServiceSubset>("ep");
var proxy = factory.CreateChannel();
MessageBox.Show(proxy.AddNumbers(2, 3).ToString());
proxy.Close();
```
Exposing Data
Using Entity Framework (Data Contracts)

S1. DataContracts
- Add ADO.NET Entity Data Model, “Generate from Database”
- Connect to (localdb)\v11.0 and Northwind database
- Select all tables and click Finish
- Review NorthwindModel.Context.cs and Category.cs

Optional but strongly recommended
- Click background of EDMX design window
- In Properties, change “Lazy Loading Enabled” to False

Exposing Data
Using Entity Framework (Service Contract and Impl.)

S2. ServiceContracts, ISampleService.cs

```csharp
[OperationContract]
Product[] GetProducts();
[OperationContract]
Product GetProduct(int ProductID);
```

S3. Implementation, SampleService.cs

```csharp
using System.Linq;

public Product[] GetProducts() {
    using (var db = new NorthwindEntities()) {
        db.Configuration.LazyLoadingEnabled = false;
        return dbProducts.ToArray();
    }
}

public Product GetProduct(int ProductID) {
    using (var db = new NorthwindEntities()) {
        db.Configuration.LazyLoadingEnabled = false;
        return db.Products.Single(p => p.ProductID == ProductID);
    }
}
Exposing Data
Using Entity Framework (Host)

H1.ConsoleHost, App.config
• Copy and paste the connectionStrings section from S1.DataContracts into the host's App.config file

```xml
<connectionStrings>
  <add name="NorthwindEntities" connectionString="metadata=res://*/NorthwindModel.csl|res://*/NorthwindModel.ssdl|res://*/NorthwindModel.msl;provider=System.Data.SqlClient;provider connection string=&quot;data source=(localdb)\v11.0;initial catalog=Northwind;integrated security=True;MultipleActiveResultSets=True;App=EntityFramework&quot;; providerName="System.Data.EntityClient"/>
</connectionStrings>
```

S3.Implementation
• Add NuGet package for EntityFramework (best to use the same version that your S1.DataContracts project uses)

Exposing Data
Using Entity Framework (Updating References)

C1.ClientUsingWSDL
• Right-click SampleServiceReference, and then click Update Service Reference
• Add a button “Get Products” and a DataGrid named grid

```csharp
private void GetProducts_Click(object sender, RoutedEventArgs e) {
    try {
        var proxy = new SampleServiceClient(Endpoints.Text);
        var products = proxy.GetProducts();
        grid.ItemsSource = products;
        proxy.Close();
    }catch (FaultException ex) {
        MessageBox.Show(ex.Message);
    }
}
```
C.1. Client Using Channel Factory

- Add a button “Get Products” and a DataGrid named grid

```csharp
private void GetProducts_Click(object sender, RoutedEventArgs e) {
    try {
        var address = new EndpointAddress("http://localhost:801/Sample/ws");
        var binding = new WSHttpBinding
            { Security = new WSHttpSecurity { Mode = SecurityMode.None }};
        var factory = new ChannelFactory<ISampleServiceWithComms>(binding, address);
        var proxy = factory.CreateChannel();
        var products = proxy.GetProducts();
        grid.ItemsSource = products;
        proxy.Close();
    }
    catch(FaultException ex) {
        MessageBox.Show(ex.Message);
    }
}
```

F.26

The Hosts - IIS

Create an IIS Host

- Add a new WCF Service Application
  - Project Name: H2.IISHost
  - Delete the IService1.cs and Service1.svc.cs files
  - Add reference to S3.Implementation project
  - Rename Service1.svc to Sample.svc and then edit it
    ```xml
    <%@ ServiceHost Service="Firebrand.SampleService" %>
    ```

- Build and test the service
  - Right-click the .svc file, choose View in Browser, click WSDL link

- Change the port number assigned by Cassini to 802
  - Project-Properties, Web
The Hosts - IIS
Test the IIS Host

Run the WCF Test Client
- File-Add Service...
- Enter URL: http://localhost:802/Sample.svc
- Test as before

Warning! When testing a Web Project solution in Visual Studio 2010 you are using the ASP.NET Development Server (aka Cassini) which only supports HTTP. To test other bindings you must either deploy to IIS 7.x with WAS or create a non-Web host like the ConsoleHost project.

The Hosts - IIS
Using Service Activations

We do not need a physical .svc file to host a service

```
<serviceHostingEnvironment ...>
<serviceActivations>
  <add relativeAddress="VirtualService.svc"
       service="Firebrand.SampleService" />
</serviceActivations>
```

You will now be able to use your “fake” service

http://localhost:802/VirtualService.svc

You can also specify a factory to customize how the service host is created

```
<add relativeAddress="VirtualService.svc"
      service="Firebrand.SampleService"
      factory="Firebrand.SampleServiceFactory" />
```
Faults
Defining a Fault Contract

**S1. DataContracts**

```csharp
[DataContract]
public class ExtraError {
    [DataMember] public int Level;
    [DataMember] public string Stuff;
}
```

**S2. ServiceContracts**

```csharp
[ServiceContract]
public interface ISampleService {
    [OperationContract]
    [FaultContract(typeof(ExtraError))]
    int AddNumbers(int a, int b);
}
```

Faults
Implementing a Fault Contract

**S3. Implementation, in the AddNumbers method**

```csharp
if (a == 0) {
    var reasons = new List<FaultReasonText>() {
        new FaultReasonText("Hello World", "en-US"),
        new FaultReasonText("Bonjour Monde", "fr-FR")
    };
    var reason = new FaultReason(reasons);
    var extra = new ExtraError() {
        Level = 11, Stuff = "This is extra.";
    };
    throw new FaultException<ExtraError>(extra, reason);
}
return a + b;
```

- Note: when using SOAP 1.1 only the first translation will be sent
- Use WHttpBinding or later to use SOAP 1.2 which does support multiple translations
**Faults**

**Catching a Fault Contract**

C1.ClientUsingServiceReference, AddNumbersButton

```csharp
catch (FaultException<ExtraError> fex) {
    MessageBox.Show(string.Format(
        "Localized: {0}, Level: {1}, Stuff: {2}"
        // fex.Message is thread localized reason
        fex.Message, fex.Detail.Level, fex.Detail.Stuff),
        "FaultException<ExtraError>");
    // all localized reasons
    fex.Reason.Translations
}
```

```csharp
catch (FaultException<ExceptionDetail> fed) {
    if includeExceptionDetailsInFaults is true
    MessageBox.Show(fed.Detail.StackTrace,
        "FaultException<ExceptionDetail>");
}
```

```csharp
catch (CommunicationException ce) {
    MessageBox.Show(ce.ToString(),
        "CommunicationException");
}
```

**Duplex Service Contract**

S2.ServiceContract, add two new interfaces

- Interface for service-side to implement

```csharp
[ServiceContract(CallbackContract = typeof(ICalcAnswer))]
public interface ICalcOneWay {
    [OperationContract(IsOneWay = true)]
    void Multiply(int a, int b);
}
```

- Interface for client-side to implement

```csharp
public interface ICalcAnswer {
    [OperationContract(IsOneWay = true)]
    void SendAnswer(int answer);
}
```

- No need for [ServiceContract] on this interface
**Service Implementation and Console Host .config**

**S3. Implementation**

```csharp
public class Calculator : ICalcOneWay {
    public void Multiply(int a, int b) {
        ICalcAnswer cb = OperationContext.Current.GetCallbackChannel<ICalcAnswer>();
        cb.SendAnswer(a * b);
    }
}
```

**H1. ConsoleHost, App.config**

- Endpoint will use the base address specified on the next slide

```xml
<service name="Firebrand.Calculator">
    <endpoint address="" binding="wsDualHttpBinding" contract="Firebrand.ICalcOneWay" />
</service>
```

**Console Host code**

**H1. ConsoleHost, Program.cs, Main method**

- Insert the following before the “Press ENTER...” line

```csharp
ServiceHost hostCalc = new ServiceHost(typeof(Calculator),
    new Uri("http://localhost:801/Calc"));
hostCalc.Description.Behaviors.Add(new ServiceMetadataBehavior() { HttpGetEnabled = true });
hostCalc.Open();
Console.WriteLine("Starting {0} service...", host.Description.Name);
foreach(var ep in hostCalc.Description.Endpoints) {
    Console.WriteLine(" Endpoint: " + ep.Name);
    Console.WriteLine(" Address: " + ep.Address);
    Console.WriteLine(" Binding: " + ep.Binding);
    Console.WriteLine(" Contract: " + ep.Contract.ContractType);
}
```
Duplex
Create the Client

Add a new **WPF Application**
- Project Name: `C3.DuplexClient`
- **Add Service Reference** and enter `http://10263a-svr1:801/Calc`
  - Expand the service to download the WSDL, click the interface
  - Namespace: `ServiceReference`, click OK

```csharp
using System.ServiceModel;
using C3.DuplexClient.ServiceReference;

public partial class MainWindow : Window, ICalcOneWayCallback {
    private CalcOneWayClient proxy;
    public void SendAnswer(int answer) {
        MessageBox.Show(answer.ToString());
    }

    // button1_Click
    proxy = new CalcOneWayClient(new InstanceContext(this));
    proxy.Multiply(3, 7);
}
```

REST
Create Explicit RESTful Binding Endpoint

```xml
<endpointBehaviors>
    <behavior name="RESTful">
        <webHttp helpEnabled="true"
            automaticFormatSelectionEnabled="true" />
    </behavior>
</endpointBehaviors>

<services>
    <service name="Firebrand.SampleService">
        <endpoint address="REST" binding="webHttpBinding"
            behaviorConfiguration="RESTful"
            contract="Firebrand.ISampleService" />
    </service>
</services>

// to allow a browser to test the operation
[webGet] // default is POST
int AddNumbers(int a, int b);

http://localhost:801/Sample/REST/AddNumbers?a=2&b=3
<int xmlns="http://...">5</int>
```

Add reference to `System.ServiceModel.Web` and import namespace

Change Console app to use full .NET Framework 4
**Discovery**
**Configure the Service to Make Announcements**

**H1.ConsoleHost, App.config**

- Triggers announcements automatically when the service host opens and closes

```xml
<serviceBehaviors>
  <behavior name="MakeAnnouncements">
    <serviceDiscovery>
      <announcementEndpoints>
        <endpoint kind="udpAnnouncementEndpoint" />
      </announcementEndpoints>
    </serviceDiscovery>
  </behavior>
</serviceBehaviors>

<services>
  <service name="Firebrand.SampleService"
    behaviorConfiguration="MakeAnnouncements" />
```

**Discovery**
**Configure the Service to Respond to Discoveries**

**H1.ConsoleHost, Program.cs**

- Add reference to `System.ServiceModel.Discovery`
- using `System.ServiceModel.Discovery`
- This will allow the service to respond to “Probe” and “Resolve” requests (we could use .config instead)
- Add the following before opening the host

```csharp
host.AddServiceEndpoint(new UdpDiscoveryEndpoint());
```
Configure the Client

Add a new Windows Forms Application

- Project Name: C4.ClientUsingDiscovery
- Switch to target the full .NET Framework 4
- Add two labels and list boxes, and one button and label

```csharp
using System.ServiceModel;
using System.ServiceModel.Discovery;
using Firebrand;
using System.Collections.ObjectModel;

// fields in Form1 class
private AnnouncementService service;
private ServiceHost host;
```

Form1_Load (use Tab, Tab to generate handlers)

```csharp
service.OnlineAnnouncementReceived += new ...
service.OfflineAnnouncementReceived += new ...
host = new ServiceHost(service);
host.AddServiceEndpoint(new UdpAnnouncementEndpoint());
host.Open();
```

Online (and offline) event handlers

```csharp
// service_OnlineAnnouncementReceived
listBox2.Items.Add("Hello from " + e.EndpointDiscoveryMetadata.Address);

// service_OfflineAnnouncementReceived
listBox2.Items.Add("Bye from " + e.EndpointDiscoveryMetadata.Address);
```

Test by starting and stopping H1.ConsoleHost
### Discovery

**Configure the Client (to Discover Services)**

```csharp
button1_Click

```

```csharp
DiscoveryClient client = new DiscoveryClient(  
    new UdpDiscoveryEndpoint());
var matches = client.Find(new FindCriteria(  
    typeof(ISampleService))).Endpoints;
client.Close();
if(matches.Count == 0) {
    listBox1.Items.Add("No services match.");
} else {
    foreach(var item in matches) {
        listBox1.Items.Add(item.Address);
    }
}
```

```csharp
Cursor lastCursor = this.Cursor;
this.Cursor = Cursors.WaitCursor;
label2.Text = "Searching...";
label2.Refresh();
this.Cursor = lastCursor;
```

### Router

**Create a Host for Routing Service**

**Add a new WCF Service Application**

- **Project Name:** H3.IISRouter
- **Delete IService1.cs, Service1.svc.cs**
- **Rename Service1.svc to Router.svc and then edit it**

```xml
```

- **Add reference to System.ServiceModel.Routing**
- **Set Port Number to be fixed as 803**

**Web.config**

```xml
<serviceBehaviors>
    <behavior name="FirebrandRouting">
        <routing filterTableName="FirebrandTable"/>
    </behavior>
</serviceBehaviors>
```
Router
Configure the Filters and Backup Lists

```xml
<routing>
  <filters>
    <filter name="FirebrandFilter" filterType="MatchAll" />
  </filters>
  <filterTables>
    <filterTable name="FirebrandTable">
      <add filterName="FirebrandFilter" endpointName="Service801" backupList="FirebrandList" />
    </filterTable>
  </filterTables>
</routing>
```

Visual Studio 2010 warns that the name attribute is not allowed but it does work

Router
Configure the Endpoints

```xml
<services>
           behaviorConfiguration="FirebrandRouting">
    <endpoint address="" name="requestReplyEndpoint"
               binding="basicHttpBinding"
    <endpoint address="Oneway" name="onewayEndpoint"
               binding="basicHttpBinding"
               contract="System.ServiceModel.Routing.ISimplexDatagramRouter" />
  </service>
</services>

<client>
  <endpoint address="http://localhost:801/Sample/ws"
             binding="wsHttpBinding"
             contract="*" name="Service801" />
  <endpoint address="http://localhost:802/Sample.svc"
             binding="basicHttpBinding"
             contract="*" name="Service802" />
</client>
```
Create an Alternative Service Implementation

S3. Implementation
- Copy class SampleService
- Rename copy to SampleServiceB
- Change the string “Suffix” to “SuffixB”

H1. ConsoleHost, Program.cs and App.config
- Change SampleService to SampleServiceB

```csharp
ServiceHost host = new ServiceHost(typeof(SampleServiceB),
<service name="Firebrand.SampleServiceB" ...>
```
- The console host will act as the primary service on port 801, but can now fail over automatically to use the secondary service on port 802 hosted by IIS

Modify the Client to use the Router

C1. ClientUsingWSDL
- Right-click ServiceReference and choose Configure Service Reference...
- Change address to: http://localhost:801/Sample.svc
- Click OK

App.config
- Edit the client endpoint to use Router.svc for address instead of talking directly to Sample.svc

```xml
<client>
  <endpoint address="http://localhost:803/Router.svc"
    binding= ...
```
**Router**

**Test the Router**

- **Build the Solution**
- **Ensure the services are running**
  - H1.ConsoleHost: start without debugger (Ctrl+F5)
  - H2.IISHost: right-click **Sample.svc**, View in Browser
  - H3.IISRouter: right-click **Router.svc**, View in Browser

- **Start C1.ClientUsingWSDL**
  - Click Modify Widget: name will be suffixed with “SuffixB”
  - Stop the H1.ConsoleHost
  - Click Modify Widget: name will be suffixed with “Suffix”

---

**Sessions**

**Define Operations That Use Sessions**

- **S2.ServiceContracts**

  ```csharp
  [OperationContract]
  int ReadCounter();
  [OperationContract]
  int ReadStaticCounter();
  [OperationContract]
  string GetMessage();
  [OperationContract]
  void SetMessage(string message);
  ```

- **S3.Implementation**

  ```csharp
  private int Counter;
  private static int StaticCounter;
  ```

  - Add the following to the AddNumbers method

  ```csharp
  Interlocked.Increment(ref Counter);
  Interlocked.Increment(ref StaticCounter);
  ```
Implement Operations That Use Sessions

**S3. Implementation**

```csharp
public int ReadCounter() {
    return Counter;
}
public int ReadStaticCounter()
    return StaticCounter;
}

private static string Message;
public string GetMessage() {
    return Message;
}
public void SetMessage(string msg) {
    Message = msg;
}
```

Call Operations That Use Sessions

**C1. ClientUsingWSDL**

- Update ServiceReference
- Display current state in a label
  ```csharp
  StateLabel.Text = "State: " + proxy.State;
  ```
- Display session ID in a label
  ```csharp
  string id = proxy.InnerChannel.SessionId;
  SessionLabel.Text = "Session: " +
    (string.IsNullOrEmpty(id) ? "none" : id);
  ```

Test by switching between Basic and WS bindings

- Basic does not support sessions but WS does
- Every time you call AddNumbers the static counter will always increment (even for multiple clients) but the instance counter will only increment for bindings that support sessions
Appendix G
What’s New in Visual Studio 2013 and Updated Exam
Developing Windows Azure and Web Services

Updated 11th April 2014

What’s New in Visual Studio 2013 and Updated Exam
Changes for EF and WCF

Exam Topics: Query and manipulate data by using the Entity Framework
- Perform asynchronous operations using Entity Framework
- Map a stored procedure

Exam Topics: Query and manipulate data by using Data Provider for Entity Framework
- Programmatically configure a Data Provider

Exam Topics: Query data by using LINQ to Entities
- Log queries and database commands
- Implement async query

Exam Topics: Create an Entity Framework data model
- Describe a data model by using Custom Code First Conventions

Exam Topics: Configure WCF services by using configuration settings
- Configure bindings (including WebSocket bindings)
- Configure message compressions and encoding

Exam Topics: Secure a WCF service
- Design and implement multiple authentication modes

Most of these topics have already been covered in the main slides
What’s New in Visual Studio 2013 and Updated Exam Changes for Web API

Exam Topics: Design a Web API
- Design and implement routes

Exam Topics: Implement a Web API
- Implement attribute routing
- Implement SignalR
- Test Web API web services

Exam Topics: Secure a Web API
- Design, implement, and extend authorization and authentication filters to control access to the application
- Implement Cross Origin Request Sharing (CORS)
- Implement SSO by using OAuth 2.0
- Configure multiple authentication modes on a single endpoint

Exam Topics: Host and manage Web API
- Self-host a Web API in your own process (a Windows service) including Open Web Interface for .NET (OWIN)

Exam Topics: Consume Web API web services
- Request batching

What’s New in Visual Studio 2013 and Updated Exam Changes for Deployment

Exam Topics: Choose a deployment strategy for a Windows Azure web application
- Deploy applications using Azure Web Site

Exam Topics: Configure a web application for deployment
- Configure WCF endpoints (including HTTPS protocol mapping), bindings, and behaviors
- Enable and monitor ASP.NET App Suspend

Exam Topics: Create, configure, and publish a web package
- Configure deployment

Exam Topics: Share assemblies between multiple applications and servers
- Configure assembly binding redirects (for example, from MVC4 to MVC5)

<table>
<thead>
<tr>
<th>Topic</th>
<th>Slide</th>
</tr>
</thead>
<tbody>
<tr>
<td>WCF New Features</td>
<td>5</td>
</tr>
<tr>
<td>Web API Attribute Routing</td>
<td>9</td>
</tr>
<tr>
<td>Web API Other New Features</td>
<td>16</td>
</tr>
<tr>
<td>ASP.NET New Features</td>
<td>21</td>
</tr>
</tbody>
</table>
**WCF New Features**

**Miscellaneous Improvements**

- **Improved Intellisense Support**
- **Support for UDP Endpoint (udpBinding)**
  - Typically TCP is used in non-time critical (may be non-real-time) applications but UDP is very useful for real-time applications
  - TCP performs data-packets checking for the order of the message to be send, it is slower than UDP
  - In UDP, packet delivery is not guaranteed and loss or out-of-arrival of packets is acceptable

---

**WCF New Features**

**HTTPS Protocol Mapping**

- **In WCF 4.0**
  - Default endpoint defines protocol mapping for basicHttpBinding
  - Although we had SSL support in previous versions of WCF, but it was necessary to declare an explicit endpoint

- **In WCF 4.5**
  - If IIS is enabled for SSL and if the service does not have any explicit endpoint defined for the specific binding, then WCF can be hosted on IIS with HTTPS enabled i.e. basicHttpsBinding
  - To experience the automatic Https protocol mapping on IIS, you need to create an Application Pool with Identity as LocalSystem
**WebSocket Binding**

The support of WebSocket in WCF 4.5 is achieved through NetHttpBinding and NetHttpsBinding.

- WebSocket is the new protocol for bidirectional communication over port 80 and 443.
- NetHttpBinding with WebSocket can replace the use of WsDualHttpBinding, since WebSocket provides a duplex channel which also supports sessions.
- This is better than WsDualHttpBinding which uses two channels, and require the use of WS-ReliableMessaging for session management.

```xml
<endpoint address="http://localhost:8083"
          binding="netHttpBinding"
          contract="Contracts.IDuplexContract"/>
```

**Authentication**

WCF 4.5 enabled you to inherit the authentication types from IIS, so you only need to declare them once.

- Set the clientCredentialType attribute to InheritedFromHost.

```xml
<service name="MyService">
  <endpoint address="" binding="basicHttpBinding"
             contract="Mycontract" bindingConfiguration="secured"/>
</service>
```

**New Features**

Web API Attribute Routing

What Is It?

Attribute routing gives you more control over the URIs in your web API

- The earlier style of routing, called convention-based routing, is still fully supported
- You can combine both techniques in the same project

One advantage of convention-based routing is that templates are defined in a single place, and the routing rules are applied consistently across all controllers

- Unfortunately, convention-based routing makes it hard to support certain URI patterns that are common in RESTful APIs

Attribute Routing in Web API 2

Create a REST API with Attribute Routing in Web API 2

Web API Attribute Routing

Why Is It Useful?

Resources often contain child resources: Customers have orders, books have authors, and so forth

- It’s natural to create URIs that reflect these relations
  /customers/1/orders
- This type of URI is difficult to create using convention-based routing

With attribute routing, it’s trivial to define a route for this URI

[Route("customers/{customerId}/orders")]
public IEnumerable<Order> GetOrdersByCustomer(int customerId)
Web API Attribute Routing

How to Enable Attribute Routing

Call `MapHttpAttributeRoutes` during configuration

```csharp
using System.Web.Http;

public static class WebApiConfig
{
    public static void Register(HttpConfiguration config)
    {
        // Web API configuration and services
        // Web API routes
        config.MapHttpAttributeRoutes();

        config.Routes.MapHttpRoute(
            name: "DefaultApi",
            routeTemplate: "api/{controller}/{id}",
            defaults: new { id = RouteParameter.Optional }
        );
    }
}
```

Web API Attribute Routing

Route Prefixes

You can set a common prefix for an entire controller

```csharp
[RoutePrefix("api/books")]
public class BooksController : ApiController
{
    // GET api/books
    [Route("")]
    public IEnumerable<Book> Get() { ... }

    // GET api/books/5
    [Route("{id:int}\")]
    public Book Get(int id) { ... }

    // POST api/books
    [Route("" )]
    public HttpResponseMessage Post(Book book) { ... }

    // GET /api/authors/1/books
    [Route("~/api/authors/{authorId:int}/books")]
    public IEnumerable<Book> GetByAuthor(int authorId) { ... }
}
```
## Web API Attribute Routing
### Route Constraints

The general syntax is "\{parameter:constraint\}"

- Here, the first route will only be selected if the "id" segment of the URI is an integer
- Otherwise, the second route will be chosen

```csharp
[Route("users/{id:int}\)]
public User GetUserById(int id)
```

```csharp
[Route("users/{name}\)]
public User GetUserByName(string name)
```

Some, such as min, take arguments in parentheses

```csharp
[Route("users/{id:int:min(1)}\)]
public User GetUserById(int id)
```

### Custom Route Constraints

#### Implement the IHttpRouteConstraint interface

```csharp
public class NonZeroConstraint : IHttpRouteConstraint
{
    public bool Match(HttpRequestMessage request, IHttpRoute route,
                        string parameterName, IDictionary<string, object> values,
                        HttpRouteDirection routeDirection)
    {
        // Implementation
    }
}
```

#### Register the route constraint

```csharp
var constraintResolver = new DefaultInlineConstraintResolver();
constraintResolver.ConstraintMap.Add("nonzero", typeof(NonZeroConstraint));
config.MapHttpAttributeRoutes(constraintResolver);
```

#### Apply the route constraint

```csharp
[Route("\{id:nonzero\}\)]
public HttpResponseMessage GetNonZero(int id)
```
Web API Attribute Routing

Route Name and Order

Setting a route name

```
[Route("api/books/{id}", Name="GetBookById")]
public BookDto GetBook(int id)
```

Setting a route order (default is zero)

```
[RoutePrefix("orders")]
public class OrdersController : ApiController
{
    [Route("{id:int}")] // constrained parameter (2nd)
    public HttpResponseMessage Get(int id) { ... }
    [Route("details")] // literal (1st)
    public HttpResponseMessage GetDetails() { ... }
    [Route("pending", RouteOrder = 1)] // (would be 2nd, but now 5th)
    public HttpResponseMessage GetPending() { ... }
    [Route("{customerName}")] // unconstrained parameter (3rd)
    public HttpResponseMessage GetByCustomer(string customerName) { ... }
    [Route("{date:datetime}")] // wildcard (4th)
    public HttpResponseMessage Get(DateTime date) { ... }
}
```

Web API Other New Features

Request Batching

Using a useful way of minimizing the number of messages that are passed between the client and the server

- To enable batching in general, use custom message handlers (DefaultHttpBatchHandler, DefaultODataBatchHandler) which you can register per-route to handle the batch requests

```
```

On the client side you can use the existing Web API client library to submit a batch request

Web API Batching
http://aspnetwebstack.codeplex.com/wikipage?title=Web+API+Request+Batching
Web API Other New Features
SSO with OAuth 2.0

To authenticate a simple http request in a Web API service you have to send a token in the http authorization header

When Microsoft released Web API it did not provide any tool or api to handle authentication

- This gap was immediately filled by open source providers such as Dominic Baeir see Thinktecture.IdentityModel
- Microsoft now has the JWT Token handler that should be used to validate JWT tokens on the web api application
- You can include it by importing the NuGet Package: JSON Web Token Handler for Microsoft .Net framework 4.5

How to Authenticate Web API using OAuth 2.0
http://blogs.microsoft.co.il/appisec/2013/10/15/how-to-authenticate-web-api-using-oauth-20/

Web API Other New Features
The Open Web Interface for .NET (OWIN)

OWIN defines a standard interface between .NET web servers and web applications

- The goal of the OWIN interface is to decouple server and application
- The Katana project represents the set of OWIN components that, while still open source, are built and released by Microsoft

There are three hosting options for Katana

- IIS/ASP.NET, Custom Host, OwinHost.exe

Katana includes two server implementations

- Microsoft.Owin.Host.SystemWeb
- Microsoft.Owin.Host.HttpListener

An Overview of Project Katana
http://www.asp.net/aspnet/overview/owin-and-katana/an-overview-of-project-katana
Web API Other New Features
Use OWIN to Self-Host ASP.NET Web API 2

🌟 Add the Web API and OWIN Packages

```bash
Install-Package Microsoft.AspNet.WebApi.OwinSelfHost
```

🌟 Configure Web API for Self-Host

```csharp
namespace OwinSelfhostSample
{
    public class Startup
    {
        public void Configuration(IAppBuilder appBuilder)
        {
            HttpConfiguration config = new HttpConfiguration();
            config.Routes.MapHttpRoute(
                name: "DefaultApi",
                routeTemplate: "api/{controller}/{id}",
                defaults: new { id = RouteParameter.Optional }
            );
            appBuilder.UseWebApi(config);
        }
    }
}
```

Use OWIN to Self-Host ASP.NET Web API 2
http://www.asp.net/web-api/overview/hosting-aspnet-web-api/use-owin-to-self-host-web-api

Web API Other New Features
Use OWIN to Self-Host ASP.NET Web API 2

🌟 Add a Web API controller
• Something like the ValuesController

🌟 Start the OWIN Host and Make a Request

```csharp
static void Main()
{
    string baseAddress = "http://localhost:9000/";
    using (WebApp.Start<Startup>(url: baseAddress))
    {
        var client = new HttpClient();
        var response = client.GetAsync(baseAddress + "api/values").Result;
        Console.WriteLine(response);
        Console.WriteLine(response.Content.ReadAsStringAsync().Result);
    }
    Console.ReadLine();
}
```

using Microsoft.Owin.Hosting;
using System;
using System.Net.Http;
ASP.NET New Features

App Suspend

Radically changes the economic model for hosting large numbers of ASP.NET sites on a server
- Built on top of a new feature in Windows Server 2012 R2 called IIS Idle Worker Process Page-out
- It’s the cloud version of the app suspend (AKA tombstoning) that you see in Windows Phone and Windows Store apps

![Graph showing 90% reduction in startup time with app suspend enabled]

ASP.NET App Suspend – responsive shared .NET web hosting

App Suspend is a new setting in IIS configuration, available on each application pool
- You can validate that an app was suspended in the event viewer, in the Application event log
  - Search for event 2310

Enable and monitor ASP.NET App Suspend on Windows Server 2012 R2