Microsoft® Official Course

Module 8

Accessing Remote Data

Microsoft®

Module Overview

- Accessing Data Across the Web
- Accessing Data in the Cloud

- In this module, you will learn how to use
  - the request and response classes in the **System.Net** namespace to directly manipulate remote data sources.
  - Windows Communication Foundation (WCF) Data Services to expose and consume an EDM over the web.
    - EDM = entity data model

Lesson 1: Accessing Data Across the Web

- Overview of Web Connectivity in the .NET Framework
- Defining a Data Contract
- Creating a Request and Processing a Response
- Authenticating a Web Request
- Sending and Receiving Data
- Demonstration: Consuming a Web Service

Accessing Data Across the Web

- Data is often exposed over the web through web services or other application programming interfaces (APIs).

- To be able to consume such data sources in your application, you need a way to send and receive messages so that you can establish a connection and ultimately send and receive data

  - the request and response pattern:
    you request something from a remote data source and then it sends a response.

  - the .NET Framework provides several request and response classes (HTTP, FTP, file) that you can use to connect to a variety of data sources.

Overview of Web Connectivity in the .NET Framework

- The .NET Framework provides the infrastructure to enable you to integrate your applications with remote data sources (FTP site, ASP.NET, or WCF Web Service).

- When consuming a remote data source in your application, the .NET Framework uses requests and responses to pass messages between the two (or more) systems. This involves the following steps:
  1. Initiate a connection to the remote data source. This might include passing a security token or user credentials so that the remote data source can authenticate you.
  2. Send a request message to the remote data source. This message may also include any data that the remote data source requires to satisfy the request, such as the identifier for the sales record you want to retrieve.
  3. Wait for the remote data source to process your request and issue a response. As a developer, you have no control over how long it might take to receive a response from a web service.
  4. Process the response, and extract any data that is included in the response.

- Not all communications have to include both a request and response message, some are one-way operations.
  - For example: if your application wants to let a web service know that it has finished processing a task.

- The .NET Framework provides the **System.Net** namespace, which contains several request and response classes that enable you to target different data sources.

Overview of Web Connectivity in the .NET Framework

Class	Description
WebRequest	An abstract class that provides the base infrastructure for any request to a Uniform Resource Identifier (URI).
WebResponse	An abstract class that provides the base infrastructure to process any response from a URI.
HttpWebRequest	A derivative of the **WebRequest** class that provides functionality for any HTTP web request.
HttpWebResponse	A derivative of the **WebResponse** class that provides functionality to process any HTTP web response.
FtpWebRequest	A derivative of the **WebRequest** class that provides functionality for any FTP request.
FtpWebResponse	A derivative of the **WebResponse** class that provides functionality to process any FTP response.
FileWebRequest	A derivative of the **WebRequest** class that provides functionality for requesting files.
FileWebResponse	A derivative of the **WebResponse** class that provides functionality to process a file response.

Defining a Data Contract

If you want to expose a custom object from a web service, you must provide metadata that describes the structure of the object.

- The serialization process uses this metadata to convert your object into a transportable format, such as XML or JavaScript Object Notation (JSON).
- This metadata provides instructions to the serializer that enable you to control which types and members are serialized.

- The .NET Framework provides the **System.Runtime.Serialization** namespace, which includes the **DataContract** and **DataMember** attributes.
  - You can use these attributes to define serializable types and members
  - Use the **DataContract** and **DataMember** attributes to expose types from a web service

```
[DataContract()]
public class SalesPerson
{
   [DataMember()]
   public string FirstName { get; set; }

   [DataMember()]
   public string LastName { get; set; }

   [DataMember()]
   public string Area { get; set; }

   [DataMember()]
   public string EmailAddress { get; set; }
}
```

Creating a Request and Processing a Response

- The following steps describe how to send an HTTP request to a web service and process the response by using the **HttpWebRequest** and **HttpWebResponse** classes:
  - Get a URI

```
var uri = "http://sales.fourthcoffee.com/SalesService.svc/GetSalesPerson";
```

    - an HTTP URI that addresses the GetSalesPerson method in the Fourth Coffee Sales Service.
    - A URI (Uniform Resource Identifier) is a string that refers to a resource. The most common are URLs [source ...]
  - Create a request object

```
var request = WebRequest.Create(uri) as HttpWebRequest;
```

    - Create a request object to configure the request and access the response.
    - Always use the static **Create** method that the **WebRequest** base class exposes and then cast to the type of request you require.
  - Get a response object from the request object
    - create a response object by invoking the **GetResponse** method on the **WebRequest** object, and you then cast to the type of response you require.

```
var response = request.GetResponse() as HttpWebResponse;
```

  - Read the properties in the response object

```
var status = response.StatusCode;
// Returns OK if a response is received.
```

    - Access and process the response by using the various members that the **WebResponse** object provides.
- If the remote data source uses a different protocol, such as FTP, you can apply the same pattern but use the **FtpWebRequest** and **FtpWebResponse** classes instead.

Handling Network Exceptions

- When consuming any remote resources, whether an FTP site or an HTTP web service, you cannot guarantee that the resource is online and listening for your request.

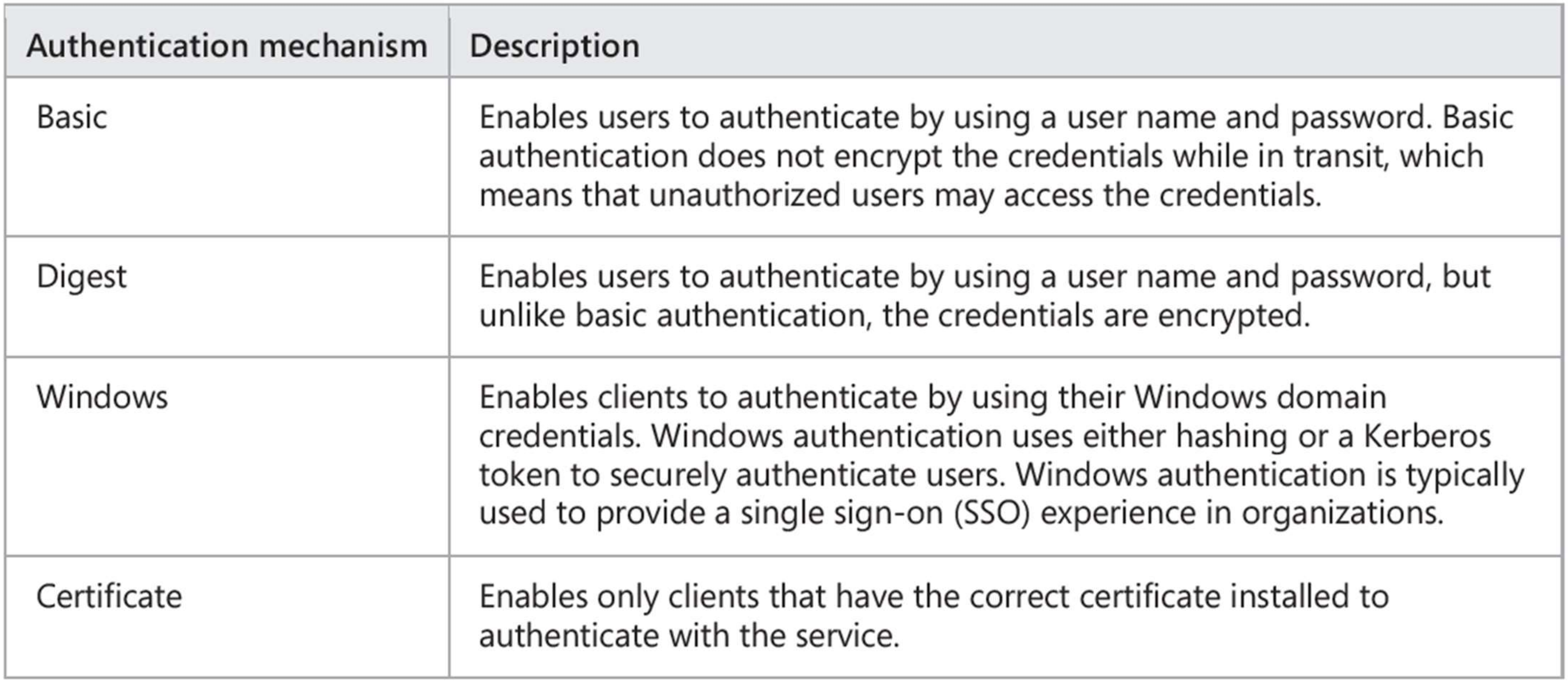
- If you try to send a request to a web service by using the **HttpWebRequest** class and the web service is not online, the **GetResponse** method call will throw a **WebException** exception with the following message:
  - *WebException – The remote server returned an error: (404) Not Found.*

- Similarly, if you try to access a secure web service with the wrong credentials or security token, the **GetResponse** method call will throw a **WebException** exception with the following message:
  - *WebException – The remote server returned an error: 401 unauthorized*

- As a minimum, you should enclose any logic that communicates with a remote data source in a **try/catch** statement, with the **catch** block handling exceptions of type **WebException**.

Type of Authentication

- The following table describes some of the common authentication techniques that you can use to secure remote data sources



Authentication mechanism	Description
Basic	Enables users to authenticate by using a user name and password. Basic authentication does not encrypt the credentials while in transit, which means that unauthorized users may access the credentials.
Digest	Enables users to authenticate by using a user name and password, but unlike basic authentication, the credentials are encrypted.
Windows	Enables clients to authenticate by using their Windows domain credentials. Windows authentication uses either hashing or a Kerberos token to securely authenticate users. Windows authentication is typically used to provide a single sign-on (SSO) experience in organizations.
Certificate	Enables only clients that have the correct certificate installed to authenticate with the service.

Authenticating a Web Request by Using Credentials

- When communicating with a remote data source that requires a user name and password, you use the **Credentials** property - in any class derived from **WebRequest** class (or implements **ICredentials**) - to pass credentials to data source.

- Create the **request** object (make sure to add: using System.Net;)

```
var uri = "http://Sales.FourthCoffee.com/SalesService.svc/GetSalesPerson";
var request = WebRequest.Create(uri) as HttpWebRequest;
```

  - Use the **NetworkCredential** class (to pass custom credentials)

```
var username = "jespera";
var password = "Pa$$w0rd";
request.Credentials = new NetworkCredential(username, password);
```

  - Use the **CredentialCache** class (to pass the credentials that the user logged on with)

```
//DefaultCredentials property gets the current user's credentials
request.Credentials = CredentialCache.DefaultCredentials;
```

  - Use the **X509Certificate2** class (to pass an X509 certificate)
    - When the remote data source receives the request, it must extract and process the X509 certificate.

```
var certificate = FourthCoffeeCertificateServices.GetCertificate();
request.ClientCertificates.Add(certificate);
```

Sending and Receiving Data

- Each derivative of the **WebRequest** and **WebResponse** classes enables you to send and receive data by using the specific protocol that the class implements.
    - E.g., **HttpWebRequest** class enables you to send requests to a web service by using the HTTP protocol.

- **WebRequest** class includes the following members used to send data to a remote data source:
    - **ContentType** property enables you to set the type of data that the request will send.
        - E.g., if you are sending JSON data by using the **HttpWebRequest** class, you use the **application/json** content type.
    - **Method** property enables you to set the type of method that the **WebRequest** object will use to send the request.
        - E.g., if you are uploading a file by using the **FtpWebRequest** class, you use the **STOR** method.
    - **ContentLength** property enables you to set the number of bytes that the request will send.
    - **GetRequestStream** method enables you to access and write data to the underlying data stream in the request object.

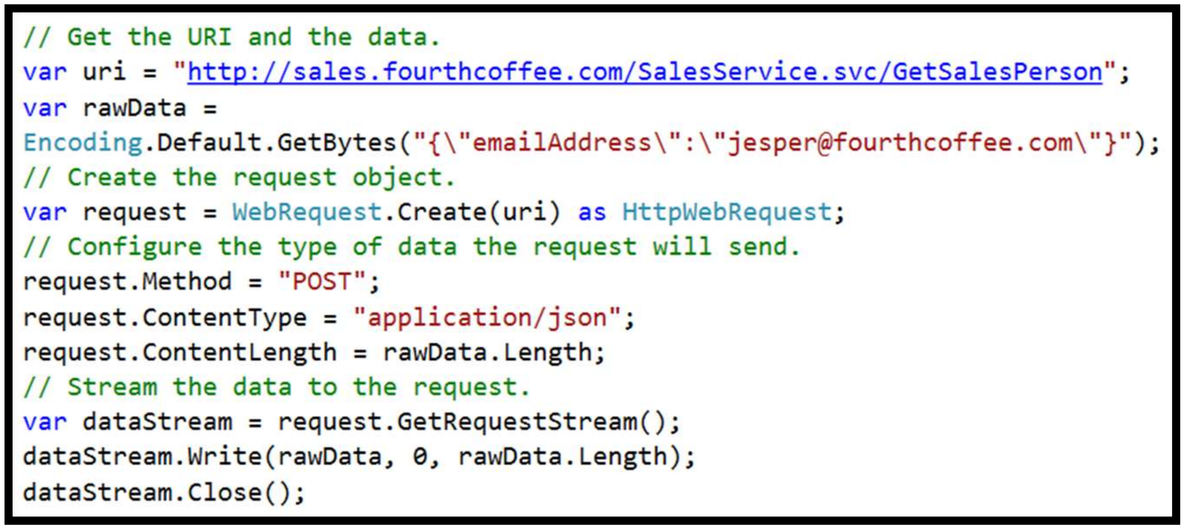
- **WebResponse** base class provides
    - **GetResponseStream** method which enables you to access and stream data from the response.

the **GetRequestStream** and **GetResponseStream** methods can be used to send and receive data.

Sending Data to a Remote Data Source

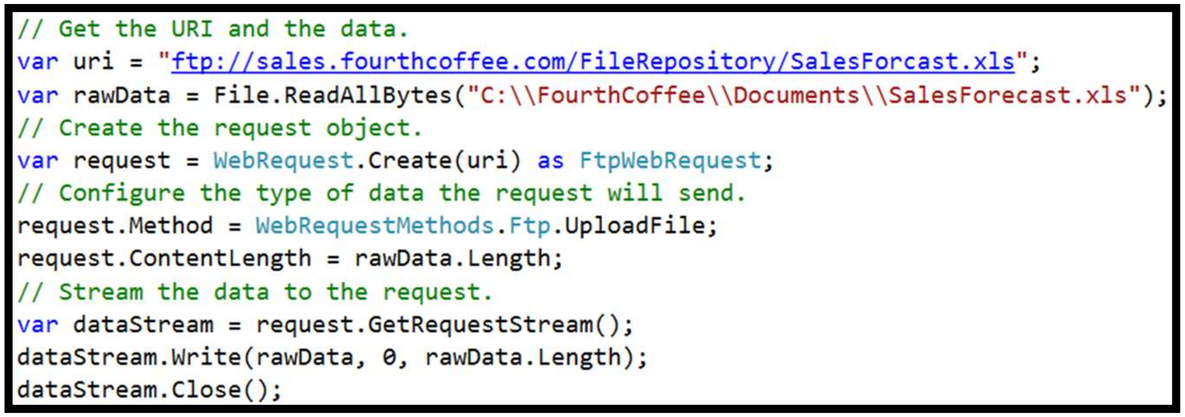
- Whether sending data to a web service or uploading a file to an FTP site, the process for creating the request remains the same:
  1. Get the URI to the remote data source and the data you want to send.
  2. Create the request object.
  3. Configure the request object, which includes setting the request method and the length of the data that the request will send.
  4. Stream the data to the request object.

- Example: use a POST operation to send a JSON string to a web service by using **HttpWebRequest**



```
// Get the URI and the data.
var uri = "http://sales.fourthcoffee.com/SalesService.svc/GetSalesPerson";
var rawData =
Encoding.Default.GetBytes("{\"emailAddress\":\"jesper@fourthcoffee.com\"}");
// Create the request object.
var request = WebRequest.Create(uri) as HttpWebRequest;
// Configure the type of data the request will send.
request.Method = "POST";
request.ContentType = "application/json";
request.ContentLength = rawData.Length;
// Stream the data to the request.
var dataStream = request.GetRequestStream();
dataStream.Write(rawData, 0, rawData.Length);
dataStream.Close();
```

- Example: upload a file to an FTP site by using **FtpWebRequest**



```
// Get the URI and the data.
var uri = "ftp://sales.fourthcoffee.com/FileRepository/SalesForcast.xls";
var rawData = File.ReadAllBytes("C:\\FourthCoffee\\Documents\\SalesForecast.xls");
// Create the request object.
var request = WebRequest.Create(uri) as FtpWebRequest;
// Configure the type of data the request will send.
request.Method = WebRequestMethods.Ftp.UploadFile;
request.ContentLength = rawData.Length;
// Stream the data to the request.
var dataStream = request.GetRequestStream();
dataStream.Write(rawData, 0, rawData.Length);
dataStream.Close();
```

Receiving Data from a Remote Data Source

- To get data that a response might contain, you use the **GetResponseStream** method of the response object.
  - The **GetResponseStream** method returns a stream, which you can read to get the data.

- Example: use the **GetResponseStream** method to access the response data:

```
var request = WebRequest.Create(uri) as HttpWebRequest;
...
var response = request.GetResponse() as HttpWebResponse;
var stream = new StreamReader(response.GetResponseStream());
// Code to process the stream.
stream.Close();
```

- When you have acquired the response stream, you must then verify that the data is in the correct format.
  - For example, if the response that is returned is in the JSON format, you can use the **DataContractJsonSerializer** class to serialize the raw JSON data into an object that you can consume in your code.
  - Alternatively, if the data is in the XML format, you can use the **SoapFormatter** class to deserialize the data or Language-Integrated Query (LINQ) to XML to manually parse the XML.

Lesson 2: Accessing Data in the Cloud

- What Is WCF Data Services?
- Defining a WCF Data Service
- Exposing a Data Model by Using WCF Data Services
- Exposing Web Methods by Using WCF Data Services
- Referencing a WCF Data Source
- Retrieving and Updating Data in a WCF Data Service
- Demonstration: Retrieving and Modifying Grade Data in the Cloud Lab

- The lab exercises use WCF Data Services.

What Is WCF (Win. Comm Foundation) Data Services?

- Representational State Transfer (REST) is an architectural model rather than a prescribed way of building web services
  - REST has become a popular model for implementing web services that need to access data
    - other models, such as those based on the WebRequest/WebResponse model are more suited to invoking remote methods.
  - REST describes a stateless, hierarchical scheme for representing resources and business objects over a network. Resources are accessed through URIs that identify the data to retrieve
  - REST provides a hierarchical model for organizing these URIs.
  - The actual scheme and structure of these URIs is the choice of the organization that is implementing the web service. For example, Fourth Coffee might choose

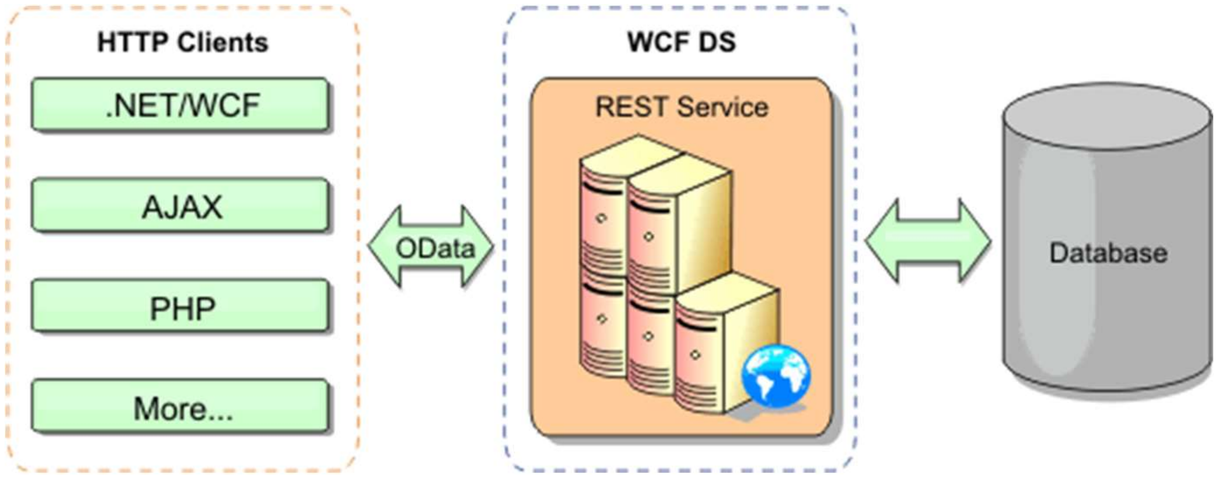
http://FourthCoffee.com/SalesService.svc/SalesPersons	//data for all of its sales people
http://FourthCoffee.com/SalesService.svc/SalesPersons/99	//data for a specific sales person (by an identifier)
http://FourthCoffee.com/SalesService.svc/SalesPersons?top=10	//it fetches the first 10 sales people only

- WCF Data Services enables you to create & access data services over the web
  - uses URIs to address data and simple, well-known formats to represent data, e.g. XML & Atom.
  - enables you to access resources over a REST application programming interface (API) and supports the standard HTTP verbs:

GET	(to retrieve data),
PUT, POST, and DELETE.	(insert, update, and delete operation)

EXTRA – REST (Representational State Transfer)

- The REST data service is deprecated (source)

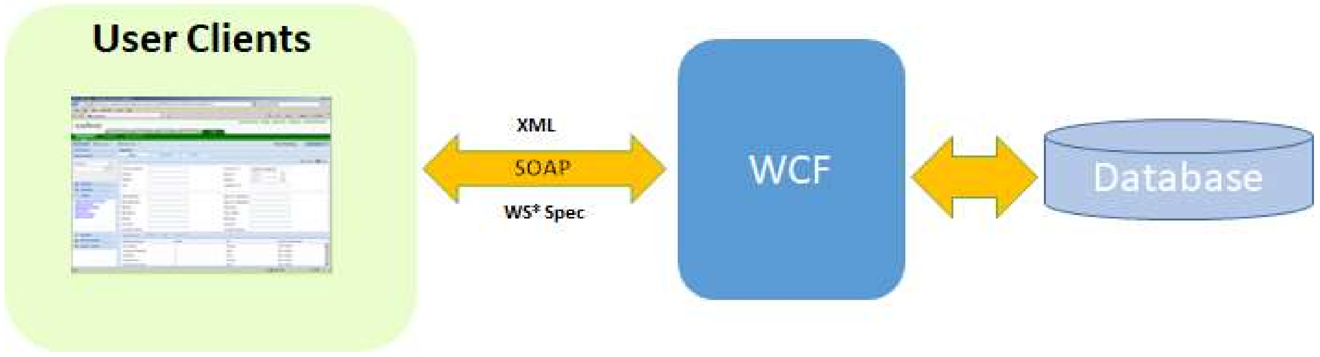


- **Guiding Principles of REST (https://restfulapi.net/)**
  - **Client–server** – By separating the user interface concerns from the data storage concerns, we improve the portability of the user interface across multiple platforms and improve scalability by simplifying the server components.
  - **Stateless** – Each request from client to server must contain all of the information necessary to understand the request, and cannot take advantage of any stored context on the server. Session state is therefore kept entirely on the client.
  - **Cacheable** –the data within a response to a request be implicitly or explicitly labeled as cacheable or non-cacheable. If a response is cacheable, then a client cache is given the right to reuse that response data for later, equivalent requests.
  - **Uniform interface** – By applying the software engineering principle of generality to the component interface, the overall system architecture is simplified and the visibility of interactions is improved. In order to obtain a uniform interface, multiple architectural constraints are needed to guide the behavior of components. REST is defined by four interface constraints: identification of resources; manipulation of resources through representations; self-descriptive messages; and, hypermedia as the engine of application state.
  - **Layered system** – The layered system style allows an architecture to be composed of hierarchical layers by constraining component behavior such that each component cannot “see” beyond the immediate layer with which they are interacting.
  - **Code on demand (optional)** – REST allows client functionality to be extended by downloading and executing code in the form of applets or scripts. This simplifies clients by reducing the number of features required to be pre-implemented.

EXTRA – WCF (Windows Communication Foundation)

- Windows Communication Foundation (WCF) is a framework for building service-oriented applications (source)
  - Using WCF, you can send data as asynchronous messages from one service endpoint to another.
  - A service endpoint can be part of a continuously available service hosted by IIS, or it can be a service hosted in an application.
  - An endpoint can be a client of a service that requests data from a service endpoint. The messages can be as simple as a single character or word sent as XML, or as complex as a stream of binary data.

- A few sample scenarios include:
  - A secure service to process business transactions.
  - A service that supplies current data to others, such as a traffic report or other monitoring service.
  - A chat service that allows two people to communicate or exchange data in real time.
  - A dashboard application that polls one or more services for data and presents it in a logical presentation.
  - Exposing a workflow implemented using Windows Workflow Foundation as a WCF service.
  - A Silverlight application to poll a service for the latest data feeds.



EXTRA – more info …

- https://docs.microsoft.com/en-us/dotnet/framework/wcf/getting-started-tutorial
  - Tutorial: Define a service contract
  - You create a WCF contract with a user-defined interface. This contract defines the functionality that the service exposes.
  - Tutorial: Implement a service contract
  - After you define a contract, you must implement it with a service class.
  - Tutorial: Host and run a basic service
  - Configure an endpoint for the service and host the service in a console application. For a service to become active, you must configure it and host it within a run-time environment. This run-time environment creates the service and controls its context and lifetime.

Defining a WCF Data Service

- By using WCF Data Services, you can expose data from relational data sources such as SQL Server through an EDM conceptual schema that is created by using the ADO.NET Entity Framework, and you can enable a client application to query and maintain data by using this schema.
  - WCF Data Services can also expose non-relational data, but this requires building customized classes

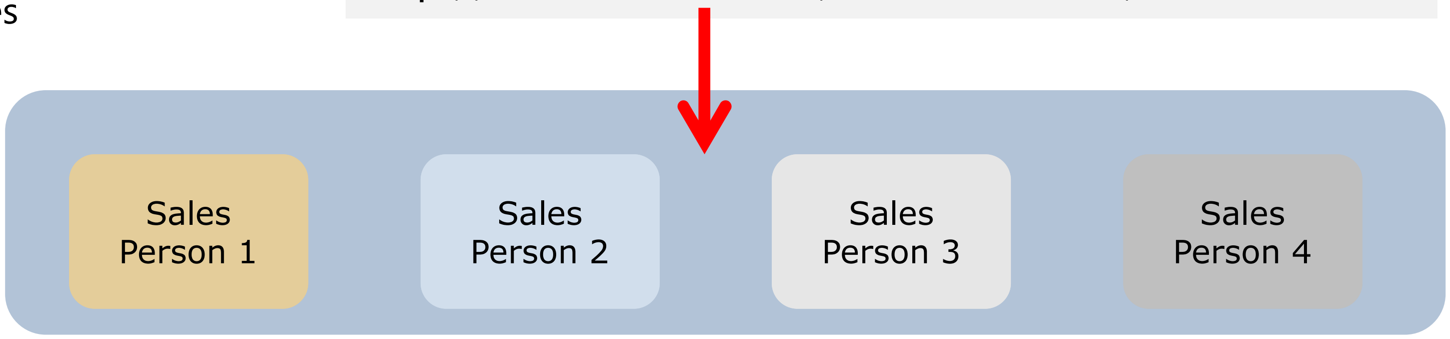
- WCF Data Services is based on the **System.Data.Services.DataService** generic class
  - the type parameter that is a collection containing at least one property that implements the **IQueryable** interface, such as the **DbContext** class

```
public class FourthCoffeeDataService : DataService<FourthCoffee>
{
   ...
}
```

  - **DataService** type implements the basic functionality to expose the entities in this collection as a series of REST resources.

- URIs are mapped to entity sets by a data service:
  - SalesPersons container for SalesPerson instances

```
http://FourthCoffee.com/SalesService.svc/SalesPersons
```



Exposing a Data Model by Using WCF Data Services

- The primary purpose of a WCF Data Service is to provide access to data
- By default, when you define a WCF Data Service, no entities are exposed
- You specify a policy that enables or disables access to resources in the **InitializeService** method of your data service.
  - Configure the access rules on the WCF Data Service by using the **SetEntitySetAccessRule** method
  - the * value means all entities. Instead, you can replace this with the name of the entity if you want to set access rules for a particular entity.
  - The **EntitySetRights.All** value grants unrestricted access to these resources.

```
public class FourthCoffeeDataService : DataService<FourthCoffee>
{
   public static void InitializeService( DataServiceConfiguration config)
   {
      config.SetEntitySetAccessRule("*", EntitySetRights.All);
   }
}
```

Exposing Web Methods by Using WCF Data Services

- The primary purpose of a WCF Data Service is to provide access to data
- You can also expose methods through a WCF Data Service.
  - You need to explicitly configure access to a method by using the **SetServiceOperationAccessRule** method.
    - specify the name of the operation and the appropriate access rights
  - Expose operations by using the **WebGet** and **WebInvoke** attributes
    - WCF Data Services operations that are accessed by using a GET request should be annotated with the **WebGet** attribute.
    - Operations that are accessed by using PUT, POST, or DELETE requests should be annotated with **WebInvoke** attribute
  - A data service operation can take parameters and returns one of the following data types:
    - IEnumerable<T> or IQueryable<T> (where T represents an entity type in the service).
      - You can invoke this operation by using the URI: **http://<hostName>/FourthCoffee/FourthCoffeeDataService.svc/SalesPersonByArea?area='snacks'**
    - T (where T represents an entity type in the service)
    - A primitive value
    - void. Not all operations have to return a value.

```
public class FourthCoffeeDataService : DataService<FourthCoffee>
{
    public static void InitializeService(DataServiceConfiguration config)
    {
      config.SetServiceOperationAccessRule("SalesPersonByEmail",
          ServiceOperationRights.ReadMultiple);
    }
    [WebGet]
    public IQueryable<SalesPerson> SalesPersonByArea(string area)
    { ... }

   [WebGet]
   [SingleResult]
   public SalesPerson SalesPersonByEmail(string emailAddress)
   { ...  }
}
```

Exposing Web Methods by Using WCF Data Services

```
{
    ...
    config.SetServiceOperationAccessRule("SalesPersonByArea",
        ServiceOperationRights.ReadMultiple);
}
...
[WebGet]
public IQueryable<SalesPerson> SalesPersonByArea(string area)
{
    if (!String.IsNullOrEmpty(area))
    {
        return from p in this.CurrentDataSource.SalesPerson
               where String.Equals(p.Area, area)
               select p;
    }
    else
    {
        throw new ArgumentException("Area must be specified", "area");
    }
}
```

```
public class FourthCoffeeDataService : DataService<FourthCoffee>
{
    public static void InitializeService(
        DataServiceConfiguration config)
    {
        ...
        config.SetServiceOperationAccessRule("SalesPersonByEmail",
            ServiceOperationRights.ReadMultiple);
    }
    ...
    [WebGet]
    [SingleResult]
    public SalesPerson SalesPersonByEmail(string emailAddress)
    {
        return (from p in this.CurrentDataSource.SalesPerson
                where String.Equals(p.Area, area)
                select p).SingleOrDefault();
    }
}
```

```
public class FourthCoffeeDataService : DataService<FourthCoffee>
{
    public static void InitializeService(
        DataServiceConfiguration config)
    {
        ...
        config.SetServiceOperationAccessRule("SalesPersonCount",
            ServiceOperationRights.ReadSingle);
    }
    ...
    [WebGet]
    [SingleResult]
    public int SalesPersonCount()
    {
       return (from p in this.CurrentDataSource.SalesPerson
               select p).Count();
    }
}
```

Referencing a WCF Data Source

Client libraries:

- A class derived from the **DataServiceContext** class
  - exposes one or more **DataServiceQuery** objects as properties
  - the name of this class is usually the same as the name of the DbContext object that is used by the EDM on which the WCF Data Service is based
    - For example, the FourthCoffeeDataService WCF Data Service uses a DbContext object called **FourthCoffeeEntities** to connect to the underlying EDM, so the name of the DataServiceContext type is also FourthCoffeeEntities.
  - this class performs a similar role to the **DbContext** class in the Entity Framework:
    - A client application connects to the data source through a **DataServiceContext** object and fetches the data for the entities that the data service exposes by using the **DataServiceQuery** properties.
    - Each **DataServiceQuery** property is a generic collection object that presents data from one of the underlying entities that provides the data for the WCF Data Service.
- Expose entities that the **DataServiceQuery** collection contains
    - A client application can perform **LINQ** queries against the **DataServiceQuery** collection properties, and the client library constructs the appropriate HTTP request to fetch the corresponding data.
    - The WCF Data Service fetches the matching data and populates the **DataServiceQuery** collection. The client application can then iterate through this collection and retrieve the data for each item.

Create a client library by using:

- The **Add Service Reference** dialog box in Visual Studio OR
  - sends a metadata query to the specified URL, and it uses the response to generate the appropriate **DataServiceContext** class that contains the **DataServiceQuery** properties and the classes for each of the entities that the WCF Data Service exposes.
- The **WCF Data Service** client utility (**DataSvcUtil**) from the command line

Retrieving and Updating Data in a WCF Data Service

- One can use the WCF Data Service context to load and modify entities in the same way you would with a local EDM, with the exception of the **UpdateObject** method, which you must use when modifying existing entities.

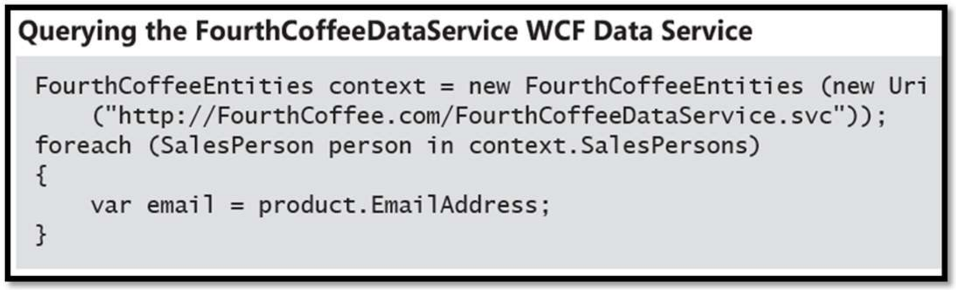
- Retrieve data from a WCF Data Service by using the client library:
  1. Create an instance of the type that is derived from the **DataServiceContext** class in the client library, and then connect to the WCF Data Service.
     - The constructor for this class expects a **Uri** object that contains the address of the service.
  2. Retrieve data by querying the appropriate **DataServiceQuery** collection in the object above.
     - When you query a DataServiceQuery collection, the client library constructs an HTTP request that specifies the resource and any criteria that is required. The query is transmitted to the WCF Data Service, and the data is returned and used to populate the DataServiceQuery object.
  3. Iterate through the items in the **DataServiceQuery** collection and process the returned objects.

Querying the FourthCoffeeDataService WCF Data Service

```
FourthCoffeeEntities context = new FourthCoffeeEntities (new Uri
    ("http://FourthCoffee.com/FourthCoffeeDataService.svc"));
foreach (SalesPerson person in context.SalesPersons)
{
    var email = product.EmailAddress;
}
```

Retrieving and Updating Data in a WCF Data Service

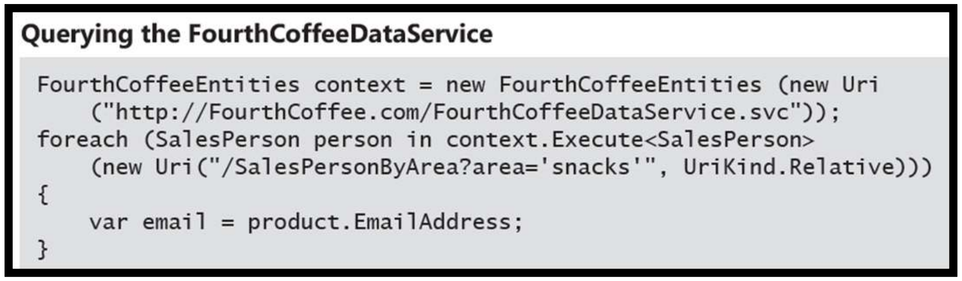
- Retrieve entities:
  - Use the properties that are exposed by the context
    - Get all email addresses …



Querying the FourthCoffeeDataService WCF Data Service

```
FourthCoffeeEntities context = new FourthCoffeeEntities (new Uri
    ("http://FourthCoffee.com/FourthCoffeeDataService.svc"));
foreach (SalesPerson person in context.SalesPersons)
{
    var email = product.EmailAddress;
}
```

  - Invoke custom service operations
    - modify data by invoking any custom service operations that you may have exposed in the WCF Data Service.
    - invoke service operations by using the **Execute** method of the **DataServiceContext** class
      - the **Execute** method returns is an enumerable collection
      - If the service operation returns a single, scalar value, you should extract it using a method such as **First**



Querying the FourthCoffeeDataService

```
FourthCoffeeEntities context = new FourthCoffeeEntities (new Uri
    ("http://FourthCoffee.com/FourthCoffeeDataService.svc"));
foreach (SalesPerson person in context.Execute<SalesPerson>
    (new Uri("/SalesPersonByArea?area='snacks'", UriKind.Relative)))
{
    var email = product.EmailAddress;
}
```

  - Use eager or explicit loading to get related entities
    - Seen below

Retrieving and Updating Data in a WCF Data Service

- Modify entities - methods provided by the **DataServiceContext** class
  - Use the **AddTo*XXXX*** method to add a new entity
    - for example, for the **SalesPerson** entity, you get an **AddToSalesPersons** method

```
FourthCoffeeEntities context = new FourthCoffeeEntities (new Uri
    ("http://FourthCoffee.com/FourthCoffeeDataService.svc"));
...
var newSalesPerson = new SalesPerson
{
   Area = "tea",
   EmailAddress = "roy@fourthcoffee.com",
   FirstName = "Roy",
   LastName = "Antebi"
 };
context.AddToSalesPersons(newSalesPerson);
context.SaveChanges();
```

  - Use the **DeleteObject** method to remove an existing entity

```
FourthCoffeeEntities context = new FourthCoffeeEntities (new Uri
    ("http://FourthCoffee.com/FourthCoffeeDataService.svc"));
...
var salesPerson = (from p in context.SalesPersons
                   where p.EmailAddress.Equals("roy@fourthcoffee.com")
                   select p).Single();
context.DeleteObject(salesPerson);
context.SaveChanges();
```

  - Use the **UpdateObject** method to modify an existing entity

```
FourthCoffeeEntities context = new FourthCoffeeEntities (new Uri
    ("http://FourthCoffee.com/FourthCoffeeDataService.svc"));
...
var salesPerson = (from p in context.SalesPersons
                   where p.EmailAddress.Equals("roy@fourthcoffee.com")
                   select p).Single();
salesperson.Area = "soft drinks";
context.UpdateObject(salesPerson);
context.SaveChanges();
```

Retrieving and Updating Data in a WCF Data Service

- Use eager or explicit loading to get related entities
  - by default <u>only the entity you requested is returned in the response</u>.
    - For example, the **SalesPerson** entity in the **FourthCoffeeEntities** object is related to the **Sales** entity. When you request a **SalesPerson** entity, the response will not include the related **Sales** entity.
  - the eager loading strategy that the **Expand** method implements causes the data for the specified related entities to be retrieved as <u>part of the same request that fetches the primary data for the query</u>.
    - useful if you know that you will always need this related data,
    - it can be wasteful of bandwidth for the cases where you do not actually use these entities.

```
FourthCoffeeEntities context = new FourthCoffeeEntities (new Uri
    ("http://FourthCoffee.com/FourthCoffeeDataService.svc"));
var salesPersons = (from s in context.SalesPersons.Expand("Sales")
                    select s).ToList();
```

  - the explicit loading strategy (the **LoadProperty** method) <u>sends an additional query to the WCF Data Service that is requesting the related data for a specific object</u>.
    - it has the advantage that it does not waste bandwidth by automatically fetching data that is not used.
    - Use the **LoadProperty** method of the **DataServiceContext** object each time you require data that is related to a particular entity
      - fetch sales associated with each SalesPerson obj.

```
FourthCoffeeEntities context = new FourthCoffeeEntities (new Uri
    ("http://FourthCoffee.com/FourthCoffeeDataService.svc"));
foreach (var salesPerson in context.SalesPersons)
{
    ...
    context.LoadProperty(salesPerson, "Sales");
    foreach (var sale in salesPerson.Sales)
    {
        ...
    }
}
```

Module Review and Takeaways

- **Question** Which of the following correctly describes how to access data that is provided in an HTTP response?
  - (   )Option 1: Invoke the GetResponseStream static method on the HttpWebResponse class.
  - (   )Option 2: Read the ContentLength instance property on the HttpWebResponse object.
  - (   )Option 3: Invoke the GetRequestStream instance method on the HttpWebResponse object.
  - (   )Option 4: Invoke the GetResponseStream instance method on the HttpWebResponse object.
  - (   )Option 5: Invoke the GetResponseStream instance method on the HttpWebRequest object.
- **Question** When you create a WCF Data Service to provide remote access to an EDM, how do you specify which entity sets the data service should make available to client applications?
  - (   )Option 1: Do nothing. All entity sets in the EDM are automatically available to client applications.
  - (   )Option 2: In the InitializeService method of the data service, use the SetEntityAccessRule method of the DataServiceConfiguration object to specify which entity sets should be made available to client applications.
  - (   )Option 3: Create a certificate for each client that can connect to the service. Configure the service to only allow authenticated clients to connect and retrieve data.
  - (   )Option 4: Define a data contract for each entity set.
  - (   )Option 5: Configure the service to enable HTTP GET requests for each entity set.