Module Overview

Module Overview

- Accessing Data Across the Web
- Accessing Data by Using OData Connected Services

Overview

Systems often consist of many components and services; some might be hosted within your organization's infrastructure, whereas others could be hosted in data centers anywhere in the world. The ability for applications to be able to interact with such services is a common requirement in modern applications.

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. You will also learn how to use Windows® Communication Foundation (WCF) Data Services to expose and consume an entity data model (EDM) over the web.

Objectives

After completing this module, you will be able to:

- Send data to and receive data from web services and other remote data sources.
- Access data by using WCF Data Services.

Lesson 1: Accessing Data Across the Web

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

Lesson Overview

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.

In this lesson, you will learn how to consume remote data sources such as web services and File Transfer Protocol (FTP) sites, which will include how to create a request, supply credentials for authentication, package data into the request, and process data that is returned in the response.

OBJECTIVES

After completing this lesson, you will be able to:

- Describe how the Microsoft® .NET Framework uses requests and responses.
- Expose types from web services by using data contracts.
- Create a request and process a response.
- Provide credentials or a security token to enable the remote data source to perform authentication.
- Send and receive data.

Overview of Web Connectivity in the .NET Framework

Overview of Web Connectivity in the .NET Framework

- Use the request and response pattern
- Use the classes in the **System.Net** namespace:
  - **WebRequest** (abstract base class)
  - **WebResponse** (abstract base class)
  - **HttpWebRequest**
  - **HttpWebResponse**
  - **FtpWebRequest**
  - **FtpWebResponse**
  - **FileWebRequest**
  - **FileWebResponse**

The .NET Framework provides the infrastructure to enable you to integrate your applications with remote data sources. The remote data source could be anything from an FTP site to an 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.

Note: Not all communications have to include both a request and response message. Depending on the nature of the application, it might only be applicable to send one message. For example, if your application wants to let a web service know that it has finished processing a task, you only need to send a request. This is known as a one-way operation.

Web Connectivity in the .NET Framework

The .NET Framework provides the **System.Net** namespace, which contains several request and response classes that enable you to target different data sources. The following table describes some of these request and response classes.

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.

Depending on whether you want to send a request to a web service by using HTTP or you want to download a file from an FTP site, the .NET Framework provides the necessary classes for you to consume these remote data sources in your applications.

Additional Reading: For more information about the System.Net namespace, refer to the System.Net Namespace page at https://aka.ms/moc-20483c-m8-pg1.

Defining a Data Contract

Defining a Data Contract

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; }
}
```

A remote data source can expose any type of data. For example, a web service can expose binary streams, scalar values, or custom objects. The choice of the type of data that you expose is determined by the requirements of your application, but how you expose it is controlled by the data contracts that you define in your web services.

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.

Data Contracts in the .NET Framework

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.

The following code example shows how to define a serializable type by using the **DataContract** and **DataMember** attributes.

Defining a Data Contract

```
[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; }
}
```

Additional Reading: For more information about the **DataContract** and **DataMember** attributes, refer to the DataContractAttribute Class page at https://aka.ms/moc-20483c-m8-pg2.

Creating a Request and Processing a Response

Creating a Request and Processing a Response

- Get a URI

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

- Create a request object

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

- Get a response object from the request object

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

- Read the properties in the response object

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

The protocol that your remote data source uses determines the request and response classes you must use. Irrespective of the classes you use, you can apply the same pattern to send a request and receive a response.

The HttpWebRequest and HttpWebResponse Classes

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:

1. Get a URI to the web service to which you want to send a request. The following code example shows an HTTP URI that addresses the **GetSalesPerson** method in the Fourth Coffee Sales Service.
   `var uri = "http://sales.fourthcoffee.com/SalesService.svc/GetSalesPerson";`
2. Create a request object to configure the request and access the response. The following code example shows how to create an **HttpWebRequest** object.
   `var request = WebRequest.Create(uri) as HttpWebRequest;`

   Note: Regardless of the type of request object you require, you always use the static Create method that the WebRequest base class exposes and then cast to the type of request you require.
3. Get the response from the request object. The following code example shows how to get the response from an **HttpWebRequest** object.
   `var response = request.GetResponse() as HttpWebResponse;`

   Note: Similar to creating a request object, you create a response object by invoking the GetResponse method on the WebRequest object, and you then cast to the type of response you require.
4. Access and process the response by using the various members that the **WebResponse** object provides. The following code example shows how to use and view the status of the response by using the **StatusCode** property.
   `var status = response.StatusCode; // Returns OK if a response is received.`

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

If you do not handle these exceptions in your code, they will cause your application to fail, offering a poor user experience. Therefore, 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**.

Authenticating a Web Request

Authenticating a Web Request

- Create the request object

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

- Use the **NetworkCredential** class

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

- Use the **CredentialCache** class

```
request.Credentials = CredentialCache.DefaultCredentials;
```

- Use the **X509Certificate2** class

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

Remote data sources are often protected to prevent unauthorized users from using the service and gaining access to data. Exposing an unprotected data source over the web can lead to unauthorized users sending requests and increasing the load on the data source. There are many ways in which you can secure remote data sources. One approach is to authenticate each user who attempts to connect to the remote data source.

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.

The nature of the service and where it is hosted are likely to influence an organization's choice of authentication mechanism. For example, a service that is exposed over an organization's intranet might use Windows authentication so that users can authenticate by using their Active Directory® Domain Services (AD DS) credentials. Using Windows authentication in this scenario will provide the users with an SSO experience and not require them to remember credentials for each service they consume.

The .NET Framework provides a number of classes that you can use to authenticate with a secure remote data source.

Authenticating Users by Using Credentials

When communicating with a remote data source that requires a user name and password, you use the **Credentials** property that is exposed by any class that is derived from the **WebRequest** base class to pass the credentials to the data source. You can set the **Credentials** property to any object that implements the **ICredentials** interface:

- The **NetworkCredential** class implements the **ICredentials** interface and enables you to encapsulate a user name and password. The following code example shows how to instantiate a **NetworkCredential** object and pass values for the user name and password to the class constructor.

```
var uri = "http://Sales.FourthCoffee.com/SalesService.svc/GetSalesPerson";
var request = WebRequest.Create(uri) as HttpWebRequest;
var username = "jespera";
var password = "Pa$$w0rd";
request.Credentials = new NetworkCredential(username, password);
```

- The **CredentialCache** class provides a number of members that enable you to get credentials in the form of an **ICredentials** object. These members include the **DefaultCredentials** property, which returns an **ICredentials** object containing the credentials that the user is currently logged on with. The following code example shows how to use the **DefaultCredentials** property to get the current user's credentials.

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

Authenticating users by using basic or Windows authentication is common, but it does require users to remember and provide credentials. Alternatively, you may choose to authenticate clients by using an X509 certificate.

Additional Reading: For more information about the NetworkCredential class, refer to the NetworkCredential Class page at https://aka.ms/moc-20483c-m8-pg3.

Authenticating Users by Using an X509 Certificate

You can use an X509 certificate as a security token to authenticate users. Instead of users having to specify credentials when they access the remote data source, they will automatically gain access as long as the request that is sent includes the correct X509 certificate. To the users, this means they need the correct X509 certificate in their certificate store on the computer where the client application is running.

The following code example shows how you can create an **HttpWebRequest** object and add an X509 certificate to the request object by using the **ClientCertificates** property.

HttpWebRequest and x509 Certificates

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

When the remote data source receives the request, it must extract and process the X509 certificate. If the X509 certificate is invalid, the remote data source can return a suitable response, such as a "The remote server returned an error: 401 unauthorized" message.

Note: The **GetCertificate** method of the web service that is shown in the previous example returns an **X509Certficate2** object. This type provides programmatic access to the properties of the X509 certificate.

Sending and Receiving Data

Sending and Receiving Data

- Send data

```
var uri =
  "http://sales.fourthcoffee.com/SalesService.svc/GetSalesPerson";
var rawData = Encoding.Default.GetBytes(
  "{\"emailAddress\":\"jespera@fourthcoffee.com\"}");
var request = WebRequest.Create(uri) as HttpWebRequest;
request.Method = "POST";
request.ContentType = "application/json";
request.ContentLength = rawData.Length;
var dataStream = request.GetRequestStream();
dataStream.Write(rawData, 0, rawData.Length);
dataStream.Close();
```

- Process the response

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

You use requests and responses to send data to or retrieve data from a remote data source. For example, you may want to add a new record to a web service or retrieve a file from an FTP site.

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

The **WebRequest** base class includes the following members that you can use to send data to a remote data source:

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

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

Sending Data to a Remote Data Source

Whether you are 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.

The following code example shows how to use a POST operation to send a JSON string to a web service by using the **HttpWebRequest** class.

Sending Data to a Web Service

```
// 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();
```

The following code example shows how to upload a file to an FTP site by using the **FtpWebRequest** class.

Uploading a File to an FTP Site

```
// 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.

The following code example shows how to use the **GetResponseStream** method to access the response data.

Reading Data from the Response

```
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.

Additional Reading: For more information about LINQ to XML, refer to the LINQ to XML page at https://aka.ms/moc-20483c-m8-pg4.

Demonstration: Consuming a Web Service

Demonstration: Consuming a Web Service

In this demonstration, you will use the **HttpWebRequest** and **HttpWebResponse** classes to consume a web service over HTTP

Preparation steps
In this demonstration, you will use the **HttpWebRequest** and **HttpWebResponse** classes to consume a web service over HTTP.
The application will use the **System.Net** classes to get sales people records from the Fourth Coffee Sale Service. You will send a request that will contain the email address of the sale person record you want to get in the JSON format, process the response, and then display the record deta
Demonstration Steps
You will find the steps in the **Demonstration: Consuming a Web Service**" section on the following page: https://github.com/MicrosoftLearning/20483-Programming-in-C-Sharp/blob/master/Instructions/20483C_MOD08_DEMO.md

Lesson 2: Accessing Data by Using OData Connected Services

Demonstration: Consuming a Web Service

In this demonstration, you will use the **HttpWebRequest** and **HttpWebResponse** classes to consume a web service over HTTP

Lesson Overview
WCF Data Services follows the Representational State Transfer (REST) architectural model and uses open web standards such as the Open Data Protocol (OData) to expose and consume data over the web. By following these standards, you can build solutions based on WCF Data Services that a wide variety of client applications can easily access, regardless of the technology that is used to implement the client application.

In this lesson, you will learn how to create a WCF Data Service and how to define which entities and operations you want to expose. You will also learn how to create a client library and how to consume the entities and operations that you have exposed.

OBJECTIVES

After completing this lesson, you will be able to:

- Describe the purpose of a WCF Data Service.
- Define a WCF Data Service.
- Expose a data model by using a WCF Data Service.
- Expose web methods by using a WCF Data Service.
- Create a client library and reference a WCF Data Service.
- Retrieve and manipulate entities that a WCF Data Service exposes.

What Is WCF Data Services?

What Is WCF Data Services?

WCF Data Services:

- Enables you to create highly flexible data services
- Uses the REST model for data access

```
http://FourthCoffee.com/SalesService.svc/SalesPersons
http://FourthCoffee.com/SalesService.svc/SalesPersons/99
http://FourthCoffee.com/SalesService.svc/SalesPersons?top=10
```

- Enables you to query and modify data by using URIs with standard HTTP verbs: GET, PUT, POST, and DELETE

WCF Data Services enables you to create and access data services over the web. You expose your data as resources that applications can access by using a URI. These resources are exposed as sets of entities that are related by associations, the same concepts as in an EDM. However, you can expose data from many types of storage, including databases and Common Language Runtime (CLR) classes.

WCF Data Services uses URIs to address data and simple, well-known formats to represent that data, such as XML and Atom. This results in data being served as a REST-style resource collection.

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 described in the previous lesson, 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. For example, Fourth Coffee might choose to make the data for all of its sales people available through the following URI:

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

The data for a specific sales person could be fetched by specifying the identifier (such as a sales person number) for that sales person, like this:

http://FourthCoffee.com/SalesService.svc/SalesPersons/99

Similarly, the details of products that it sells might be available through the following URI:

http://FourthCoffee.com/SalesService.svc/Products

The details for a specific product could be retrieved by including the product ID in the URI:

http://FourthCoffee.com/SalesService.svc/Products/25

Note: The exact URI scheme that a web service uses to expose data is a decision of the organization that implements the web service, but the examples that are shown here illustrate a common pattern.

The REST model performs HTTP GET queries to retrieve data, but the REST model also supports insert, update, and delete operations by using HTTP PUT, POST, and DELETE requests.

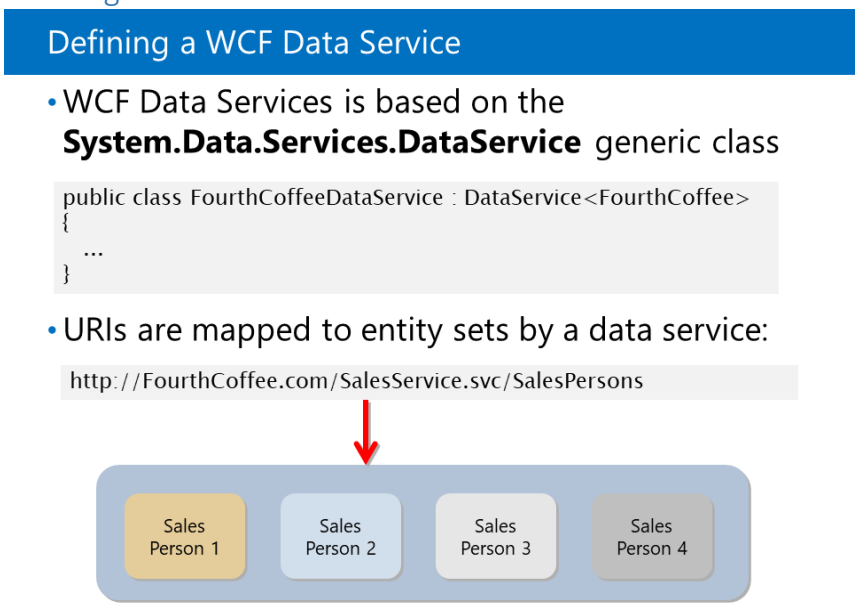
The REST model enables a web service to extend URIs to support filtering, sorting, and paging of data. For example, the following URI sends a request that fetches the first 10 sales people only:

http://FourthCoffee.com/SalesService.svc/SalesPersons?top=10

The list of filters and other functionality depends on how the web service is implemented, but features such as these are available with WCF Data Services. Additionally, WCF Data Services enables you to extend your web services by writing service operations as methods that perform business logic at the server. These methods are then accessible as URIs in a similar manner to resources. You can also define interceptors, which are called when you query, insert, update, or delete data and can validate or modify the data, enforce security, or reject the change.

Additional Reading: For more information about WCF Data Services, refer to the WCF Data Services page at http://go.microsoft.com/fwlink/?LinkID=267815.

Defining a WCF Data Service



By using WCF Data Services, you can expose data from relational data sources such as Microsoft 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.

Note: WCF Data Services can also expose non-relational data, but this requires building customized classes. WCF Data Services operates most naturally with the model that the ADO.NET Entity Framework presents.

A WCF Data Service is based on the **System.Data.Services.DataService** generic class. This class expects a type parameter that is a collection containing at least one property that implements the **IQueryable** interface, such as the **DbContext** class for an entity set that is defined by using the Entity Framework. The **DataService** type implements the basic functionality to expose the entities in this collection as a series of REST resources.

The following code example shows the definition of a WCF Data Service based on a **DbContext** class called FourthCoffee that is generated by using the ADO.NET Entity Framework.

Defining a Data Service

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

You can implement methods in a WCF Data Service that specify the entities to expose from the underlying EDM and that configure the size of datasets that the data service presents. You can also override methods that are inherited from the **DataService** class to customize the way in which the service operates.

By default, WCF Data Services uses a simple addressing scheme that exposes the entity sets that are defined within the specified EDM. When you consume a WCF Data Service, you address these entity resources as an entity set that contains instances of an entity type. For example, suppose that the following URI (shown in the previous topic) returns all of the **SalesPerson** entities that were defined in the EDM that was used to construct a WCF Data Service:

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

The "/SalesPersons" element of the URI points to the SalesPersons entity set, which is the container for SalesPerson instances.

Additional Reading: For more information about defining a WCF Data Service, refer to the Exposing Your Data as a Service (WCF Data Services) page at http://go.microsoft.com/fwlink/?LinkID=267816.

Exposing a Data Model by Using WCF Data Services

Exposing a Data Model by Using WCF Data Services

Configure the access rules on the WCF Data Service by using the **SetEntitySetAccessRule** method

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

For security reasons, WCF Data Services does not automatically expose any resources, such as entity collections, that the EDM implements. You specify a policy that enables or disables access to resources in the **InitializeService** method of your data service. This method takes a **DataServiceConfiguration** object, which has a **SetEntitySetAccessRule** property that you use to define the access policy.

The following code example shows how to allow access to all resources that the WCF Data Service exposes.

Data Service Access to all Entities

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

The parameters to the **SetEntitySetAccessRule** method are the name of a resource and the access rights to grant over that resource. You can specify a resource explicitly, or you can use wildcard characters. The ***** value that is shown in the code example is a shorthand way of specifying all resources that the WCF Data Service publishes. The **EntitySetRights.All** value grants unrestricted access to these resources.

Note: Enumerations and partial classes that are implemented in an EDM do not propagate through WCF Data Services. If you want to expose enumerations and extensions to partial classes, you should include these types in a separate assembly that you can reference in the service and client application directly.

Additional Reading: For more information about defining access rules in a WCF Data Service, refer to the Configuring the Data Service (WCF Data Services) page at http://go.microsoft.com/fwlink/?LinkID=267817.

Exposing Web Methods by Using WCF Data Services

Exposing Web Methods by Using WCF Data Services

Expose operations by using the **WebGet** and **WebInvoke** attributes

```
public class FourthCoffeeDataService : DataService<FourthCoffee>
{
    public static void InitializeService(DataServiceConfiguration config)
    {
        config.SetServiceOperationAccessRule("SalesPersonByEmail",
            ServiceOperationRights.ReadMultiple);
    }

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

The primary purpose of a WCF Data Service is to provide access to data. However, you can also implement custom operations that manipulate data. A WCF Data Service operation is simply a method of the data service that is visible to the REST infrastructure and that can be accessed by sending an HTTP GET, PUT, POST, or DELETE request.
WCF Data Services operations that are accessed by using a GET request should be annotated with the **WebGet** attribute. These operations typically return data, although they may run some business logic that does not return a value. Operations that are accessed by using PUT, POST, or DELETE requests should be annotated with the **WebInvoke** attribute. These operations typically modify the data that the service uses in some way.
Similar to entity sets, you must explicitly enable access to operations that a WCF Data Service exposes. You can do this by calling the **SetServiceOperationAccessRule** method of the **DataServiceConfiguration** object when you initialize the service. You specify the name of the operation and the appropriate access rights.
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). If an operation returns an enumerable collection based on an entity type that the service recognizes, a client application can perform queries by specifying HTTP URIs in the manner shown in the previous topics in this lesson. Implementing an operation that returns an enumerable collection in this way gives you detailed control over the contents of the collection. It is the responsibility of your code to generate this collection, possibly based on information that the client application provides. The following code example shows an operation that retrieves sales people per area.

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

You can invoke this operation by using the following URI:
http://<*hostName*>/FourthCoffee/FourthCoffeeDataService.svc/SalesPersonByArea?area='snacks'

- **T** (where **T** represents an entity type in the service). An operation can return a single instance of an entity. The following code example shows an operation that retrieves a sales person record that has a specific email address. Notice that you should also annotate an operation that returns a scalar value with the **SingleResult** attribute.

```
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();
}
}
```

- A primitive value. The following code example shows an operation that retrieves a count of all of the sales people and returns the count value as an **int**.

```
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();
}
}
```

- **void**. Not all operations have to return a value.

Additional Reading: For more information about service operations, refer to the Service Operations (WCF Data Services) page at http://go.microsoft.com/fwlink/?LinkID=267818.

Referencing a WCF Data Source

Referencing a WCF Data Source

- Client libraries:
  - Are derived from the **DataServiceContext** class
  - Expose entities that the **DataServiceQuery** collection contains
- Create a client library by using:
  - The **Add Service Reference** function in Visual Studio
  - The **DataSvcUtil** command line utility

The client library for a WCF Data Service consists of a class that is derived from the **DataServiceContext** type that 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 that is generated for the client library is also **FourthCoffeeEntities**.

The **DataServiceContext** 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.

The client library also provides definitions of the types that each **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.

You can generate the client library for a WCF Data Service by using the **Add Service Reference** dialog box in Visual Studio or by using the WCF Data Service client utility, DataSvcUtil, from the command line.

Adding a Service Reference

You can use the **Add Service Reference** dialog box in a client application. This dialog box enables you to specify the URL of the WCF Data Service to connect to. The dialog box 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 returned metadata is stored in the client project as an .edmx file. This is not the same as an .edmx file that is generated by using the ADO.NET Entity Data Model Designer (it has a different format), but you can view this metadata file by using the XML editor or any text editor.

To add a data service reference, perform the following steps:

1. If the data service is not part of the solution and is not already running, start the data service and note its URI.
2. In Solution Explorer, right-click the client project, and then select **Add Service Reference**.
3. If the data service is part of the current solution, click **Discover**.
4. Alternatively, if the data service is not part of the current solution, in the **Address** box, type the base URL of the data service, and then click **Go**.
5. Click **OK**.

After you have referenced the WCF Data Service, you can then consume the entities and service operations that it exposes.

Additional Reading: For more information about creating a service reference by using the command line, refer to the WCF Data Service Client Utility (DataSvcUtil.exe) page at http://go.microsoft.com/fwlink/?LinkID=267819.

Retrieving and Updating Data in a WCF Data Service

Retrieving and Updating Data in a WCF Data Service

- Retrieve entities:
  - Use the properties that are exposed by the context
  - Invoke custom service operations
  - Use eager or explicit loading to get related entities
- Modify entities:
  - Use the **AddTo***XXXX* method to add a new entity
  - Use the **DeleteObject** method to remove an existing entity
  - Use the **UpdateObject** method to modify an existing entity

After you have referenced the WCF Data Service by generating a client library, you can then consume the EDM and any service operations that the service exposes.

Retrieving Data

To retrieve data from a WCF Data Service by using the client library, perform the following steps:

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 **DataServiceContext** object. 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 objects that are returned.

The following code example connects to the **FourthCoffeeDataService** WCF Data Service by using the FourthCoffeeEntities type in the client library (this is the class that is derived from **DataServiceContext**). The parameter to the constructor is the address of the service. The code then queries the **SalesPersons DataServiceQuery** property to fetch all sales people and reads the email address for each record.

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;
}
```

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

The following code example shows how to invoke the **SalesPersonByArea** service operation and iterate the results.

Querying the FourthCoffeeDataService WCF Data Service by Using a Service Operation

```
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;
}
```

Modifying Data

After you have retrieved data, you can modify the entities as you would when working with an EDM and then save your changes back to the WCF Data Service.

The **DataServiceContext** class provides the following methods that enable you to work with the entities in your EDM.

- ***AddToXXXX***. These methods enable you to add a new entity to the entity collection. The following code example shows how to use the **AddToSalesPersons** method to add a new **SalesPerson** entity.

```
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();
```

- **DeleteObject**. This method enables you to remove an existing object from the entity collection. The following code example shows how to use the **DeleteObject** method to delete a sales person with the email address **roya@fourthcoffee.com**.

```
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();
```

- **UpdateObject**. This method enables you to update an existing object in the entity collection. The following code example shows how to use the **UpdateObject** method to change the area to which a sales person belongs.

```
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();
```

Implementing Eager Loading of Entities

When retrieving data by using WCF Data Services, by default only the entity you requested is returned in the response. 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. However, you can use the **Expand** or **LoadProperty** methods to get related entities.

Using the eager loading strategy that the **Expand** method implements causes the data for the specified related entities to be retrieved as part of the same request that fetches the primary data for the query. This approach is useful if you know that you will always need this related data, but it can be wasteful of bandwidth for the cases where you do not actually use these entities.

The following code example shows how to use the **Expand** method to fetch the sales associated with each **SalesPerson** object.

Eager Loading of Entities

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

As an alternative, you can use explicit loading. This strategy sends an additional query to the WCF Data Service that is requesting the related data for a specific object, but it has the advantage that it does not waste bandwidth by automatically fetching data that is not used.

You can implement explicit loading by using the **LoadProperty** method of the **DataServiceContext** object. You call the **LoadProperty** method each time you require data that is related to a particular entity; you specify the entity and the name of the **DataServiceQuery** collection property that holds the related data.

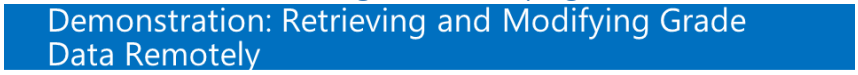
The following code example shows how to use the **LoadProperty** method to fetch the sales that are associated with each **SalesPerson** object.

Explicit Loading of Entities

```
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)
{
...
}
}
```

Additional Reading: For more information about modifying entities, refer to the How to: Add, Modify, and Delete Entities (WCF Data Services) page at http://go.microsoft.com/fwlink/?LinkID=267820.

Demonstration: Retrieving and Modifying Grade Data Remotely



In this demonstration, you will learn about the tasks that you will perform in the lab for this module

In this demonstration, you will learn about the tasks that you will perform in the lab for this module.

Demonstration Steps

You will find the steps in the **Demonstration: Retrieving and Modifying Grade Data Remotely** section on the following page: https://github.com/MicrosoftLearning/20483-Programming-in-C-Sharp/blob/master/Instructions/20483C_MOD08_DEMO.md.

Lab Scenario

Lab Scenario

Currently, the application retrieves data from a local database. However, you have decided to store the data in the cloud and must configure the application so that it can retrieve data across the web.

You must create a WCF Data Service for the **SchoolGrades** database that will be integrated into the application to enable access to the data.

Finally, you have been asked to write code that displays student images by retrieving them from across the web.

Lab: Retrieving and Modifying Grade Data Remotely

Lab: Retrieving and Modifying Grade Data Remotely

- Exercise 1: Creating a WCF Data Service for the SchoolGrades Database
- Exercise 2: Integrating the Data Service into the Application
- Exercise 3: Retrieving Student Photographs Over the Web (If Time Permits)

Estimated Time: 60 minutes

Scenario

Currently, the application retrieves data from a local database. However, you have decided to store the data in the cloud and must configure the application so that it can retrieve data across the web.

You must create a WCF Data Service for the **SchoolGrades** database that will be integrated into the application to enable access to the data.

Finally, you have been asked to write code that displays student images by retrieving them from across the web.

Objectives

After completing this lab, you will be able to:

- Create a WCF Data Service.
- Use an OData Connected Service.
- Retrieve data over the web.

Lab Setup

Ensure that you have followed all the steps listed in the Setup Guide for this course. The Setup Guide contains installations instructions specific to this module's lab. The Setup guide can be found in the **Instructions** folder (https://github.com/MicrosoftLearning/20483-Programming-in-C-Sharp/tree/master/Instructions)
The specific installation resources can be found in the **Allfiles** directory: https://github.com/MicrosoftLearning/20483-Programming-in-C-Sharp/tree/master/Allfiles/Assets.
You will find the high-level steps on the following page: https://github.com/MicrosoftLearning/20483-Programming-in-C-Sharp/blob/master/Instructions/20483C_MOD08_LAB_MANUAL.md.
You will find the detailed steps on the following page: https://github.com/MicrosoftLearning/20483-Programming-in-C-Sharp/blob/master/Instructions/20483C_MOD08_LAK.md.

Module Review and Takeaways

Module Review and Takeaways

- Review Questions

In this module, you have learned how to use the request and response classes in the **System.Net** namespace to manipulate remote data sources directly and how to use WCF Data Services to expose and consume an entity data model over the web.

Review Questions

1. Which of the following correctly describes how to access data that is provided in an HTTP response?
( )Invoke the GetResponseStream static method on the HttpWebResponse class.
( )Read the ContentLength instance property on the HttpWebResponse object.
( )Invoke the GetRequestStream instance method on the HttpWebResponse object.
( )Invoke the GetResponseStream instance method on the HttpWebResponse object.
( )Invoke the GetResponseStream instance method on the HttpWebRequest object.

2. 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?
( )Do nothing. All entity sets in the EDM are automatically available to client applications.
( )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.
( )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.
( )Define a data contract for each entity set.
( )Configure the service to enable HTTP GET requests for each entity set.