Contents

WCF Data Services 4.5

8/31/2018 • 2 minutes to read • Edit Online

WCF Data Services (formerly known as "ADO.NET Data Services") is a component of the .NET Framework that enables you to create services that use the Open Data Protocol (OData) to expose and consume data over the Web or intranet by using the semantics of representational state transfer (REST). OData exposes data as resources that are addressable by URIs. Data is accessed and changed by using standard HTTP verbs of GET, PUT, POST, and DELETE. OData uses the entity-relationship conventions of the Entity Data Model to expose resources as sets of entities that are related by associations.

WCF Data Services uses the OData protocol for addressing and updating resources. In this way, you can access these services from any client that supports OData. OData enables you to request and write data to resources by using well-known transfer formats: Atom, a set of standards for exchanging and updating data as XML, and JavaScript Object Notation (JSON), a text-based data exchange format used extensively in AJAX application.

WCF Data Services can expose data that originates from various sources as OData feeds. Visual Studio tools make it easier for you to create an OData-based service by using an ADO.NET Entity Framework data model. You can also create OData feeds based on common language runtime (CLR) classes and even late-bound or un-typed data.

WCF Data Services also includes a set of client libraries, one for general .NET Framework client applications and another specifically for Silverlight-based applications. These client libraries provide an object-based programming model when you access an OData feed from environments such as the .NET Framework and Silverlight.

Where Should I Start?

Depending on your interests, consider getting started with WCF Data Services in one of the following topics.

I want to jump right in...

- Quickstart
- Getting Started
- Silverlight Quickstart
- Silverlight Quickstart for Windows Phone Development

Just show me some code...

- Quickstart
- How to: Execute Data Service Queries
- How to: Bind Data to Windows Presentation Foundation Elements

I want to know more about OData...

- Whitepaper: Introducing OData
- Open Data Protocol Web site
- OData: SDK
- OData: Frequently Asked Questions

I want to watch some videos...

- Beginner's Guide to WCF Data Services
- WCF Data Services Developer Videos
- OData: Developers Web site

I want to see end-to-end samples...

- WCF Data Services Documentation Samples on MSDN Samples Gallery
- Other WCF Data Services Samples on MSDN Samples Gallery
- OData: SDK

How does it integrate with Visual Studio?

- Generating the Data Service Client Library
- Creating the Data Service
- Entity Framework Provider

What can I do with it?

- Overview
- Whitepaper: Introducing OData
- Application Scenarios

I want to use Silverlight...

- Silverlight Quickstart
- WCF Data Services (Silverlight)
- Getting Started with Silverlight

I want to use LINQ...

- Querying the Data Service
- LINQ Considerations
- How to: Execute Data Service Queries

I still need some more information...

- WCF Data Services Team Blog
- Resources
- WCF Data Services Developer Center
- Open Data Protocol Web site

In This Section

Overview

Provides an overview of the features and functionality available in WCF Data Services.

What's New in WCF Data Services

Describes new functionality in WCF Data Services and support for new OData features.

Getting Started

Describes how to expose and consume OData feeds by using WCF Data Services.

Defining WCF Data Services

Describes how to create and configure a data service that exposes OData feeds.

WCF Data Services Client Library

Describes how to use client libraries to consume OData feeds from a .NET Framework client application.

See Also

- Representational State Transfer (REST)

WCF Data Services Overview

8/31/2018 • 5 minutes to read • Edit Online

WCF Data Services enables creation and consumption of data services for the Web or an intranet by using the Open Data Protocol (OData). OData enables you to expose your data as resources that are addressable by URIs. This enables you to access and change data by using the semantics of representational state transfer (REST), specifically the standard HTTP verbs of GET, PUT, POST, and DELETE. This topic provides an overview of both the patterns and practices defined by OData and also the facilities provided by WCF Data Services to take advantage of OData in .NET Framework-based applications.

Address Data as Resources

OData exposes data as resources that are addressable by URIs. The resource paths are constructed based on the entity-relationship conventions of the Entity Data Model. In this model, entities represent operational units of data in an application domain, such as customers, orders, items, and products. For more information, see Entity Data Model.

In OData, you address entity resources as an entity set that contains instances of entity types. For example, the URI `http://services.odata.org/Northwind/Northwind.svc/Customers('ALFKI')/Orders` returns all of the orders from the `Northwind` data service that are related to the customer with a `CustomerID` value of `ALFKI.`

Query expressions enable you to perform traditional query operations against resources, such as filtering, sorting, and paging. For example, the URI `http://services.odata.org/Northwind/Northwind.svc/Customers('ALFKI')/Orders?$filter=Freight gt 50` filters the resources to return only the orders with a freight cost of more than $50. For more information, see Accessing Data Service Resources.

Interoperable Data Access

OData builds on standard Internet protocols to make data services interoperable with applications that do not use the .NET Framework. Because you can use standard URIs to address data, your application can access and change data by using the semantics of representational state transfer (REST), specifically the standard HTTP verbs of GET, PUT, POST, and DELETE. This enables you to access these services from any client that can parse and access data that is transmitted over standard HTTP protocols.

OData defines a set of extensions to the Atom Publishing Protocol (AtomPub). It supports HTTP requests and responses in more than one data format to accommodate various client applications and platforms. An OData feed can represent data in Atom, JavaScript Object Notation (JSON), and as plain XML. While Atom is the default format, the format of the feed is specified in the header of the HTTP request. For more information, see OData: Atom Format and OData: JSON Format.

When publishing data as an OData feed, WCF Data Services relies on other existing Internet facilities for such operations as caching and authentication. To accomplish this, WCF Data Services integrates with existing hosting applications and services, such as ASP.NET, Windows Communication Foundation (WCF), and Internet Information Services (IIS).

Storage Independence

Although resources are addressed based on an entity-relationship model, WCF Data Services expose OData feeds regardless of the underlying data source. After WCF Data Services accepts an HTTP request for a resource that a URI identifies, the request is deserialized and a representation of that request is passed to an WCF Data Services provider. This provider translates the request into a data source-specific format and executes the request on the underlying data source. WCF Data Services achieves storage independence by separating the conceptual model that addresses resources prescribed by OData from the specific schema of the underlying data source.

WCF Data Services integrates with the ADO.NET Entity Framework to enable you to create data services that expose relational data. You can use the Entity Data Model tools to create a data model that contains addressable resources as entities and at the same time define the mapping between this model and the tables in the underlying database. For more information, see Entity Framework Provider.

WCF Data Services also enables you to create data services that expose any data structures that return an implementation of the IQueryable<T> interface. This enables you to create data services that expose data from .NET Framework types. Create, update, and delete operations are supported when you also implement the IUpdatable interface. For more information, see Reflection Provider.

For an illustration of how WCF Data Services integrates with these data providers, see the architectural diagram later in this topic.

Custom Business Logic

WCF Data Services makes it easy to add custom business logic to a data service through service operations and interceptors. Service operations are methods defined on the server that are addressable by URIs in the same form as data resources. Service operations can also use query expression syntax to filter, order, and page data returned by an operation. For example, the URI `http://localhost:12345/Northwind.svc/GetOrdersByCity?city='London'&$orderby=OrderDate&$top=10&$skip=10` represents a call to a service operation named `GetOrdersByCity` on the Northwind data service that returns orders for customers from London, with paged results sorted by `OrderDate`. For more information, see Service Operations.

Interceptors enable custom application logic to be integrated in the processing of request or response messages by a data service. Interceptors are called when a query, insert, update, or delete action occurs on the specified entity set. An interceptor then may alter the data, enforce authorization policy, or even terminate the operation. Interceptor methods must be explicitly registered for a given entity set that is exposed by a data service. For more information, see Interceptors.

Client Libraries

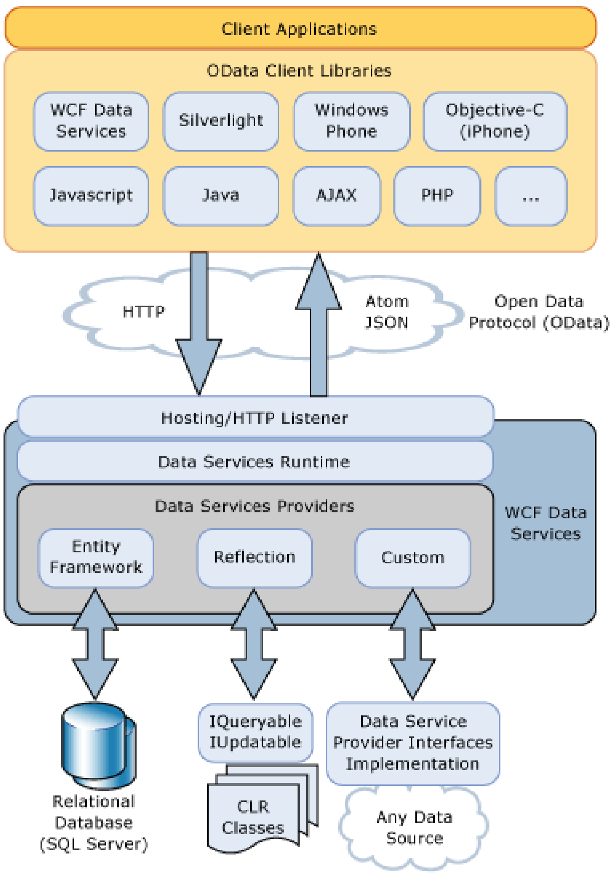
OData defines a set of uniform patterns for interacting with data services. This provides an opportunity to create reusable components that are based on these services, such as client-side libraries that make it easier to consume data services.

WCF Data Services includes client libraries for both .NET Framework-based and Silverlight-based client applications. These client libraries enable you to interact with data services by using .NET Framework objects. They also support object-based queries and LINQ queries, loading related objects, change tracking, and identity resolution. For more information, see WCF Data Services Client Library.

In addition to the OData client libraries included with the .NET Framework and with Silverlight, there are other client libraries that enable you to consume an OData feed in client applications, such as PHP, AJAX, and Java applications. For more information, see the OData SDK.

Architecture Overview

The following diagram illustrates the WCF Data Services architecture for exposing OData feeds and using these feeds in OData-enabled client libraries:



See Also

WCF Data Services 4.5
Getting Started
Defining WCF Data Services
Accessing a Data Service (WCF Data Services)
WCF Data Services Client Library
Representational State Transfer (REST)

Getting Started with WCF Data Services

8/31/2018 • 2 minutes to read • Edit Online

The topics in this section help you quickly understand Open Data Protocol (OData) and how to use WCF Data Services to expose and consume OData feeds by explaining the underlying technologies. This section includes both conceptual content and a quickstart tutorial.

In This Section

The following topics describe how to create data services by using WCF Data Services.

Exposing Your Data as a Service
Describes the steps that are required to create a data service by using WCF Data Services.

Accessing Data Service Resources
Describes how to work with OData feeds.

Using a Data Service in a Client Application
Describes how to work with an OData feed in a .NET Framework client application.

Quickstart
Shows how to create and access a simple OData-based service that exposes a feed based on the Northwind sample database.

Application Scenarios
Highlights a core set of OData scenarios that are supported by WCF Data Services.

Resources
Provides links to WCF Data Services and OData resources.

Related Sections

WCF Data Services (Silverlight)

Getting Started

See Also

OData SDK

Expose Your Data as a Service (WCF Data Services)

8/29/2018 • 2 minutes to read • Edit Online

WCF Data Services integrates with Visual Studio to enable you to more easily define services to expose your data as Open Data Protocol (OData) feeds. Creating a data service that exposes an OData feed involves the following basic steps:

1. **Define the data model.** WCF Data Services natively supports data models that are based on the ADO.NET Entity Framework. For more information, see How to: Create a Data Service Using an ADO.NET Entity Framework Data Source.

   WCF Data Services also supports data models that are based on common language runtime (CLR) objects that return an instance of the IQueryable<T> interface. This enables you to deploy data services that are based on lists, arrays, and collections in the .NET Framework. To enable create, update, and delete operations over these data structures, you must also implement the IUpdatable interface. For more information, see How to: Create a Data Service Using the Reflection Provider.

   For more advanced scenarios, WCF Data Services includes a set of providers that enable you to define a data model based on late-bound data types. For more information, see Custom Data Service Providers.

2. **Create the data service.** The most basic data service exposes a class that inherits from the DataService<T> class, with a type `T` that is the namespace-qualified name of the entity container. For more information, see Defining WCF Data Services.

3. **Configure the data service.** By default, WCF Data Services disables access to resources that are exposed by an entity container. The DataServiceConfiguration interface enables you to configure access to resources and service operations, specify the supported version of OData, and to define other service-wide behaviors, such as batching behaviors or the maximum number of entities that can be returned in a single response. For more information, see Configuring the Data Service.

For an example of how to create a simple data service that is based on the Northwind sample database, see Quickstart.

See Also

- Getting Started
- Overview

Accessing Data Service Resources (WCF Data Services)

8/31/2018 • 5 minutes to read • Edit Online

WCF Data Services supports the Open Data Protocol (OData) to expose your data as a feed with resources that are addressable by URIs. These resources are represented according to the entity-relationship conventions of the Entity Data Model. In this model, entities represent operational units of data that are data types in an application domain, such as customers, orders, items, and products. Entity data is accessed and changed by using the semantics of representational state transfer (REST), specifically the standard HTTP verbs of GET, PUT, POST, and DELETE.

Addressing Resources

In OData, you address any data exposed by the data model by using a URI. For example, the following URI returns a feed that is the Customers entity set, which contains entries for all instances of the Customer entity type:

```
http://services.odata.org/Northwind/Northwind.svc/Customers
```

Entities have special properties called entity keys. An entity key is used to uniquely identify a single entity in an entity set. This enables you to address a specific instance of an entity type in the entity set. For example, the following URI returns an entry for a specific instance of the Customer entity type that has a key value of `ALFKI`:

```
http://services.odata.org/Northwind/Northwind.svc/Customers('ALFKI')
```

Primitive and complex properties of an entity instance can also be individually addressed. For example, the following URI returns an XML element that contains the `ContactName` property value for a specific Customer:

```
http://services.odata.org/Northwind/Northwind.svc/Customers('ALFKI')/ContactName
```

When you include the `$value` endpoint in the previous URI, only the value of the primitive property is returned in the response message. The following example returns only the string "Maria Anders" without the XML element:

```
http://services.odata.org/Northwind/Northwind.svc/Customers('ALFKI')/ContactName/$value
```

Relationships between entities are defined in the data model by associations. These associations enable you to address related entities by using navigation properties of an entity instance. A navigation property can return either a single related entity, in the case of a many-to-one relationship, or a set of related entities, in the case of a one-to-many relationship. For example, the following URI returns a feed that is the set of all the Orders that are related to a specific Customer:

```
http://services.odata.org/Northwind/Northwind.svc/Customers('ALFKI')/Orders
```

Relationships, which are usually bi-directional, are represented by a pair of navigation properties. As the reverse of the relationship shown in the previous example, the following URI returns a reference to the Customer entity to which a specific Order entity belongs:

```
http://services.odata.org/Northwind/Northwind.svc/Orders(10643)/Customer
```

OData also enables you to address resources based on the results of query expressions. This makes it possible to filter sets of resources based on an evaluated expression. For example, the following URI filters the resources to return only the Orders for the specified Customer that have shipped since September 22, 1997:

```
http://services.odata.org/Northwind/Northwind.svc/Customers('ALFKI')/Orders?$filter=ShippedDate gt datetime'1997-09-
22T00:00:00'
```

For more information, see OData: URI Conventions.

System Query Options

OData defines a set of system query options that you can use to perform traditional query operations against resources, such as filtering, sorting, and paging. For example, the following URI returns the set of all the `Order` entities, along with related `Order_Detail` entities, the postal codes of which do not end in `100`:

```
http://services.odata.org/Northwind/Northwind.svc/Orders?$filter=not
endswith(ShipPostalCode,'100')&$expand=Order_Details&$orderby=ShipCity
```

The entries in the returned feed are also ordered by the value of the ShipCity property of the orders.

WCF Data Services supports the following OData system query options:

QUERY OPTION	DESCRIPTION

`$orderby`	Defines a default sort order for entities in the returned feed. The following query orders the returned customers feed by county and city:
	`http://services.odata.org/Northwind/Northwind.svc/Customers?`
	`$orderby=Country,City`
	For more information, see OData: OrderBy System Query Option ($orderby).

`$top`	Specifies the number of entities to include in the returned feed. The following example skips the first 10 customers and then returns the next 10:
	`http://services.odata.org/Northwind/Northwind.svc/Customers?`
	`$skip=10&$top=10`
	For more information, see OData: Top System Query Option ($top).

`$skip`	Specifies the number of entities to skip before starting to return entities in the feed. The following example skips the first 10 customers and then returns the next 10:
	`http://services.odata.org/Northwind/Northwind.svc/Customers?`
	`$skip=10&$top=10`
	For more information, see OData: Skip System Query Option ($skip).

`$filter`	Defines an expression that filters the entities returned in the feed based on specific criteria. This query option supports a set of logical comparison operators, arithmetic operators, and predefined query functions that are used to evaluate the filter expression. The following example returns all orders the postal codes of which do not end in 100:
	`http://services.odata.org/Northwind/Northwind.svc/Orders?`
	`$filter=not endswith(ShipPostalCode,'100')`
	For more information, see OData: Filter System Query Option ($filter).

`$expand`	Specifies which related entities are returned by the query. Related entities are included as either a feed or an entry inline with the entity returned by the query. The following example returns the order for the customer 'ALFKI' along with the item details for each order:
	`http://services.odata.org/Northwind/Northwind.svc/Customers('ALFKI')/`
	`$expand=Order_Details`
	For more information, see OData: Expand System Query Option ($expand).

QUERY OPTION	DESCRIPTION
`$select`	Specifies a projection that defines the properties of the entity are returned in the projection. By default, all properties of an entity are returned in a feed. The following query returns only three properties of the `Customer` entity: `http://services.odata.org/Northwind/Northwind.svc/Customers?$select=CustomerID,CompanyName,City` For more information, see OData: Select System Query Option ($select).
`$inlinecount`	Requests that a count of the number of entities returned in the feed be included with the feed. For more information, see OData: Inlinecount System Query Option ($inlinecount).

Addressing Relationships

In addition to addressing entity sets and entity instances, OData also enables you to address the associations that represent relationships between entities. This functionality is required to be able to create or change a relationship between two entity instances, such as the shipper that is related to a given order in the Northwind sample database. OData supports a `$link` operator to specifically address the associations between entities. For example, the following URI is specified in an HTTP PUT request message to change the shipper for the specified order to a new shipper.

```
http://services.odata.org/Northwind/Northwind.svc/Orders(10643)/$links/Shipper
```

For more information, see OData: Addressing Links between Entries.

Consuming the Returned Feed

The URI of an OData resource enables you to address entity data exposed by the service. When you enter a URI into the address field of a Web browser, a OData feed representation of the requested resource is returned. For more information, see the WCF Data Services Quickstart. Although a Web browser may be useful for testing that a data service resource returns the expected data, production data services that can also create, update, and delete data are generally accessed by application code or scripting languages in a Web page. For more information, see Using a Data Service in a Client Application.

See Also

Open Data Protocol Web site

Using a Data Service in a Client Application (WCF Data Services)

8/31/2018 • 4 minutes to read • Edit Online

You can access a service that exposes an Open Data Protocol (OData) feed by supplying a URI to a Web browser. The URI provides the address of a resource, and request messages are sent to these addresses to access or change the underlying data that the resource represents. The browser issues an HTTP GET command and returns the requested resource as an OData feed. For more information, see Accessing the Service from a Web Browser.

Although a Web browser may be useful for testing that an OData service returns the expected data, production OData services that enable you to also create, update, and delete data are generally accessed by application code or scripting languages in a Web page. This topic provides an overview of how to access OData feeds from a client application.

Accessing and Changing Data Using REST Semantics

OData helps guarantee interoperability between services that expose OData feeds and applications that consume OData feeds. Applications access and change data in an OData-based service by sending request messages of a specific HTTP action and with a URI that addresses an entity resource against which the action should be performed. When entity data must be supplied, it is supplied as a specifically encoded payload in the body of the message.

HTTP Actions

OData supports the following HTTP actions to perform create, read, update, and delete operations on the entity data that the addressed resource represents:

- **HTTP GET** - This is the default action when a resource is accessed from a browser. No payload is supplied in the request message, and a response method with a payload that contains the requested data is returned.
- **HTTP POST** - Inserts new entity data into the supplied resource. Data to be inserted is supplied in the payload of the request message. The payload of the response message contains the data for the newly created entity. This includes any autogenerated key values. The header also contains the URI that addresses the new entity resource.
- **HTTP DELETE** - Deletes the entity data that the specified resource represents. A payload is not present in the request or response messages.
- **HTTP PUT** - Replaces existing entity data at the requested resource with new data that is supplied in the payload of the request message.
- **HTTP MERGE** - Because of inefficiencies in executing a delete followed by an insert in the data source just to change entity data, OData introduces a new HTTP MERGE action. The payload of the request message contains the properties that must be changed on the addressed entity resource. Because HTTP MERGE is not defined in the HTTP specification, it may require additional processing to route a HTTP MERGE request through non-OData aware servers.

For more information, see OData: Operations.

Payload Formats

For an HTTP PUT, HTTP POST, or HTTP MERGE request, the payload of a request message contains the entity data that you send to the data service. The contents of the payload depend on the data format of the message. The

HTTP responses to all actions except DELETE also contain such a payload. OData supports the following payload formats for accessing and changing data with the service:

- **Atom** - An XML-based message encoding that is defined by OData as an extension to the Atom Publishing Protocol (AtomPub) to enable data exchange over HTTP for Web feeds, podcasts, wikis, and XML-based Internet functionality. For more information, see OData: Atom Format.

- **JSON** - JavaScript Object Notation (JSON) is a lightweight data interchange format that is based on a subset of the JavaScript Programming Language. For more information, see OData: JSON Format.

The message format of the payload is requested in the header of the HTTP request message. For more information, see OData: Operations.

Accessing and Changing Data Using Client Libraries

WCF Data Services includes client libraries that enable you to more easily consume an OData feed from .NET Framework and Silverlight-based client applications. These libraries simplify sending and receiving HTTP messages. They also translate the message payload into CLR objects that represent entity data. The client libraries feature the two core classes DataServiceContext and DataServiceQuery<TElement>. These classes enable you to query a data service and then work with the returned entity data as CLR objects. For more information, see WCF Data Services Client Library and WCF Data Services (Silverlight).

You can use the **Add Service Reference** dialog in Visual Studio to add a reference to a data service. This tool requests the service metadata from a referenced data service and generates the DataServiceContext that represents a data service, as well as generates the client data service classes that represent entities. For more information, see Generating the Data Service Client Library.

There are programming libraries available that you can use to consume an OData feed in other kinds of client applications. For more information, see the OData SDK.

See Also

Accessing Data Service Resources
Quickstart

Quickstart (WCF Data Services)

8/31/2018 • 2 minutes to read • Edit Online

This quickstart helps you become familiar with WCF Data Services and the Open Data Protocol (OData) through a series of tasks that support the topics in Getting Started.

What you'll learn

The first task in this quickstart shows how to create a data service to expose an OData feed from the Northwind sample database. In later topics, you will access the OData feed by using a Web browser, and also create a Windows Presentation Foundation (WPF) client application that consumes the OData feed by using client libraries.

Prerequisites

To complete this quickstart, you must install the following components:

- Visual Studio
- An instance of SQL Server. This includes SQL Server Express, which is included in a default installation of Visual Studio 2015, or as part of the **Data storage and processing** workload in Visual Studio 2017.
- The Northwind sample database. To download this sample database, see the download page, Sample Databases for SQL Server.

WCF data services quickstart tasks

Create the Data Service

Define the ASP.NET application, define the data model, create the data service, and enable access to resources.

Access the Service from a Web Browser

Start the service from Visual Studio and access the service by submitting HTTP GET requests through a Web browser to the exposed feed.

Create the .NET Framework Client Application

Create a WPF app to consume the OData feed, bind data to Windows controls, change data in the bound controls, and then send the changes back to the data service.

NOTE

Project files from a completed version of the quickstart can be downloaded from the WCF Data Services Documentation Samples page.

Next steps

Start the quickstart

See also

- ADO.NET Entity Framework

Create the data service

8/31/2018 • 3 minutes to read • Edit Online

In this topic, you create a sample data service that uses WCF Data Services to expose an Open Data Protocol (OData) feed that's based on the Northwind sample database. The task involves the following basic steps:

1. Create an ASP.NET Web application.
2. Define the data model by using the Entity Data Model tools.
3. Add the data service to the Web application.
4. Enable access to the data service.

Create the ASP.NET web app

1. In Visual Studio, on the **File** menu, select **New** > **Project**.
2. In the **New Project** dialog box, under either Visual Basic or Visual C# select the **Web** category, and then select **ASP.NET Web Application**.
3. Enter `NorthwindService` as the name of the project and then select **OK**.
4. In the **New ASP.NET Web Application** dialog, select **Empty** and then select **OK**.
5. (Optional) Specify a specific port number for your Web application. Note: the port number `12345` is used in this series of quickstart topics.
   a. In **Solution Explorer**, right-click on the ASP.NET project that you just created, and then choose **Properties**.
   b. Select the **Web** tab, and set the value of the **Specific port** text box to `12345`.

Define the data model

1. In **Solution Explorer**, right-click the name of the ASP.NET project, and then click **Add** > **New Item**.
2. In the **Add New Item** dialog box, select the **Data** category, and then select **ADO.NET Entity Data Model**.
3. For the name of the data model, enter `Northwind.edmx`.
4. In the **Entity Data Model Wizard**, select **EF Designer from Database**, and then click **Next**.
5. Connect the data model to the database by doing one of the following steps, and then click **Next**:
   - If you don't have a database connection already configured, click **New Connection** and create a new connection. For more information, see How to: Create Connections to SQL Server Databases. This SQL Server instance must have the Northwind sample database attached.

     - or -

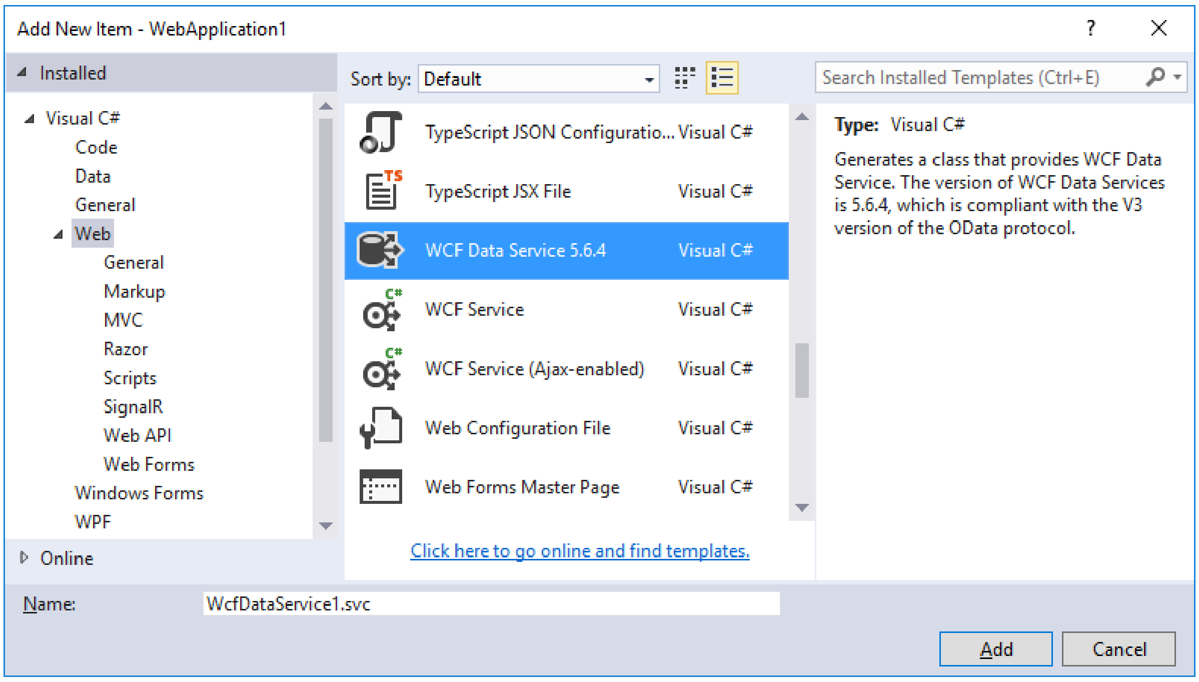
   - If you have a database connection already configured to connect to the Northwind database, select that connection from the list of connections.
6. On the final page of the wizard, select the check boxes for all tables in the database, and clear the check boxes for views and stored procedures.

7. Click **Finish** to close the wizard.

Create the WCF data service

1. In **Solution Explorer**, right-click on the ASP.NET project, and then choose **Add** > **New Item**.

2. In the **Add New Item** dialog box, select the **WCF Data Service** item template from the **Web** category.


Add New Item - WebApplication1
?
Installed
Visual C#
Code
Data
General
Web
General
Markup
MVC
Razor
Scripts
SignalR
Web API
Web Forms
Windows Forms
WPF
Online
Sort by: Default
Search Installed Templates (Ctrl+E)
TypeScript JSON Configuratio... Visual C#
TypeScript JSX File Visual C#
WCF Data Service 5.6.4 Visual C#
WCF Service Visual C#
WCF Service (Ajax-enabled) Visual C#
Web Configuration File Visual C#
Web Forms Master Page Visual C#
Click here to go online and find templates.
Add
Cancel

Type: Visual C#

Generates a class that provides WCF Data Service. The version of WCF Data Services is 5.6.4, which is compliant with the V3 version of the OData protocol.

Name: WcfDataService1.svc

NOTE

The **WCF Data Service** template is available in Visual Studio 2015, but not in Visual Studio 2017.

3. For the name of the service, type `Northwind`.

   Visual Studio creates the XML markup and code files for the new service. By default, the code-editor window opens. In **Solution Explorer**, the service has the name Northwind with the extension *.svc.cs* or *.svc.vb*.

4. In the code for the data service, replace the comment `/* TODO: put your data source class name here */` in the definition of the class that defines the data service with the type that is the entity container of the data model, which in this case is `NorthwindEntities`. The class definition should look this the following:

```
public class Northwind : DataService<NorthwindEntities>
```

```
Public Class Northwind
    Inherits DataService(Of NorthwindEntities)
```

Enable access to data service resources

1. In the code for the data service, replace the placeholder code in the `InitializeService` function with the following:

```
 // Grant only the rights needed to support the client application.
config.SetEntitySetAccessRule("Orders", EntitySetRights.AllRead
     | EntitySetRights.WriteMerge
     | EntitySetRights.WriteReplace );
 config.SetEntitySetAccessRule("Order_Details", EntitySetRights.AllRead
     | EntitySetRights.AllWrite);
 config.SetEntitySetAccessRule("Customers", EntitySetRights.AllRead);
```

```
' Grant only the rights needed to support the client application.
config.SetEntitySetAccessRule("Orders", EntitySetRights.AllRead _
     Or EntitySetRights.WriteMerge _
     Or EntitySetRights.WriteReplace)
config.SetEntitySetAccessRule("Order_Details", EntitySetRights.AllRead _
    Or EntitySetRights.AllWrite)
config.SetEntitySetAccessRule("Customers", EntitySetRights.AllRead)
```

This enables authorized clients to have read and write access to resources for the specified entity sets.

NOTE

Any client that can access the ASP.NET application can also access the resources exposed by the data service. In a production data service, to prevent unauthorized access to resources you should also secure the application itself. For more information, see Securing WCF Data Services.

Next steps

You have successfully created a new data service that exposes an OData feed that is based on the Northwind sample database, and you have enabled access to the feed for clients that have permissions on the ASP.NET Web application. Next, you'll start the data service from Visual Studio and access the OData feed by submitting HTTP GET requests through a Web browser:

Access the service from a web browser

See also

- ADO.NET Entity Data Model Tools

Accessing the Service from a Web Browser (WCF Data Services Quickstart)

8/29/2018 • 2 minutes to read • Edit Online

This is the second task of the WCF Data Services quickstart. In this task, you start the WCF Data Services from Visual Studio and optionally disable feed reading in the Web browser. You then retrieve the service definition document as well as access data service resources by submitting HTTP GET requests through a Web browser to the exposed resources.

NOTE

By default, Visual Studio auto-assigns a port number to the `localhost` URI on your computer. This task uses the port number `12345` in the URI examples. For more information about how to set a specific port number in your Visual Studio project see Creating the Data Service.

To request the default service document by using Internet Explorer

1. In Internet Explorer, from the **Tools** menu, select **Internet Options**, click the **Content** tab, click **Settings**, and clear **Turn on feed viewing**.

   This makes sure that feed reading is disabled. If you do not disable this functionality, then the Web browser will treat the returned AtomPub encoded document as an XML feed instead of displaying the raw XML data.

   NOTE

   If your browser cannot display the feed as raw XML data, you should still be able to view the feed as the source code for the page.

2. In Visual Studio, press the **F5** key to start debugging the application.

3. Open a Web browser on the local computer. In the address bar, enter the following URI:

   ```
   http://localhost:12345/northwind.svc
   ```

   This returns the default service document, which contains a list of entity sets that are exposed by this data service.

To access entity set resources from a Web browser

1. In the address bar of your Web browser, enter the following URI:

   ```
   http://localhost:12345/northwind.svc/Customers
   ```

   This returns a set of all customers in the Northwind sample database.

2. In the address bar of your Web browser, enter the following URI:

```
http://localhost:12345/northwind.svc/Customers('ALFKI')
```

This returns an entity instance for the specific customer, `ALFKI`.

3. In the address bar of your Web browser, enter the following URI:

   ```
   http://localhost:12345/northwind.svc/Customers('ALFKI')/Orders
   ```

   This traverses the relationship between customers and orders to return a set of all orders for the specific customer `ALFKI`.

4. In the address bar of your Web browser, enter the following URI:

   ```
   http://localhost:12345/northwind.svc/Customers('ALFKI')/Orders?$filter=OrderID eq 10643
   ```

   This filters orders that belong to the specific customer `ALFKI` so that only a specific order is returned based on the supplied `OrderID` value.

Next Steps

You have successfully accessed the WCF Data Services from a Web browser, with the browser issuing HTTP GET requests to specified resources. A Web browser provides an easy way to experiment with the addressing syntax of requests and view the results. However, a production data service is not generally accessed by this method. Typically, applications interact with the data service through application code or scripting languages. Next, you will create a client application that uses client libraries to access data service resources as if they were common language runtime (CLR) objects:

Creating the .NET Framework Client Application

See Also

- Accessing Data Service Resources

Creating the .NET Framework Client Application (WCF Data Services Quickstart)

8/29/2018 • 4 minutes to read • Edit Online

This is the final task of the WCF Data Services quickstart. In this task, you will add a console application to the solution, add a reference to the Open Data Protocol (OData) feed into this new client application, and access the OData feed from the client application by using the generated client data service classes and client libraries.

> **NOTE**
>
> A .NET Framework-based client application is not required to access a data feed. The data service can be accessed by any application component that consumes an OData feed. For more information, see Using a Data Service in a Client Application.

To create the client application by using Visual Studio

1. In **Solution Explorer**, right-click the solution, click **Add**, and then click **New Project**.
2. In the left pane, select **Installed** > [**Visual C#** or **Visual Basic**] > **Windows Desktop**, and then select the **WPF App** template.
3. Enter `NorthwindClient` for the project name, and then click **OK**.
4. Open the file MainWindow.xaml and replace the XAML code with the following code:

```
    <Window x:Class="Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="Northwind Orders" Height="335" Width="425"
        Name="OrdersWindow" Loaded="Window1_Loaded">
    <Grid Name="orderItemsGrid">
        <ComboBox DisplayMemberPath="OrderID" ItemsSource="{Binding}"
                  IsSynchronizedWithCurrentItem="true"
                  Height="23" Margin="92,12,198,0" Name="comboBoxOrder" VerticalAlignment="Top"/>
        <DataGrid ItemsSource="{Binding Path=Order_Details}"
                  CanUserAddRows="False" CanUserDeleteRows="False"
                  Name="orderItemsDataGrid" Margin="34,46,34,50"
                  AutoGenerateColumns="False">
            <DataGrid.Columns>
                <DataGridTextColumn  Header="Product" Binding="{Binding ProductID, Mode=OneWay}" />
                <DataGridTextColumn  Header="Quantity" Binding="{Binding Quantity, Mode=TwoWay}" />
                <DataGridTextColumn  Header="Price" Binding="{Binding UnitPrice, Mode=TwoWay}" />
                <DataGridTextColumn  Header="Discount" Binding="{Binding Discount, Mode=TwoWay}" />
            </DataGrid.Columns>
        </DataGrid>
        <Label Height="28" Margin="34,12,0,0" Name="orderLabel" VerticalAlignment="Top"
               HorizontalAlignment="Left" Width="65">Order:</Label>
        <StackPanel Name="Buttons" Orientation="Horizontal" HorizontalAlignment="Right"
                    Height="40" Margin="0,257,22,0">
            <Button Height="23" HorizontalAlignment="Right" Margin="0,0,12,12"
                Name="buttonSave" VerticalAlignment="Bottom" Width="75"
                    Click="buttonSaveChanges_Click">Save Changes
            </Button>
            <Button Height="23" Margin="0,0,12,12"
                Name="buttonClose" VerticalAlignment="Bottom" Width="75"
                    Click="buttonClose_Click">Close</Button>
        </StackPanel>
    </Grid>
</Window>
```

To add a data service reference to the project

1. In **Solution Explorer**, right-click the NorthwindClient project, click **Add** > **Service Reference**, and then click **Discover**.

   This displays the Northwind data service that you created in the first task.

2. In the **Namespace** text box, type `Northwind`, and then click **OK**.

   This adds a new code file to the project, which contains the data classes that are used to access and interact with data service resources as objects. The data classes are created in the namespace `NorthwindClient.Northwind`.

To access data service data in the WPF application

1. In **Solution Explorer** under **NorthwindClient**, right-click the project and click **Add Reference**.

2. In the **Add Reference** dialog box, click the **.NET** tab, select the System.Data.Services.Client.dll assembly, and then click **OK**.

3. In **Solution Explorer** under **NorthwindClient**, open the code page for the MainWindow.xaml file, and add the following `using` statement (`Imports` in Visual Basic).

```
using System.Data.Services.Client;
using NorthwindClient.Northwind;
```

```
Imports System.Data.Services.Client
Imports NorthwindClient.Northwind
```

4. Insert the following code that queries that data service and binds the result to a DataServiceCollection<T> into the `MainWindow` class:

NOTE

You must replace the host name `localhost:12345` with the server and port that is hosting your instance of the Northwind data service.

```
private NorthwindEntities context;
private string customerId = "ALFKI";

// Replace the host server and port number with the values
// for the test server hosting your Northwind data service instance.
private Uri svcUri = new Uri("http://localhost:12345/Northwind.svc");

private void Window1_Loaded(object sender, RoutedEventArgs e)
{
    try
    {
        // Instantiate the DataServiceContext.
        context = new NorthwindEntities(svcUri);

        context.IgnoreMissingProperties = true;

        // Define a LINQ query that returns Orders and
        // Order_Details for a specific customer.
        var ordersQuery = from o in context.Orders.Expand("Order_Details")
                          where o.Customer.CustomerID == customerId
                          select o;

        // Create an DataServiceCollection<T> based on
        // execution of the LINQ query for Orders.
        DataServiceCollection<Order> customerOrders = new
            DataServiceCollection<Order>(ordersQuery);

        // Make the DataServiceCollection<T> the binding source for the Grid.
        this.orderItemsGrid.DataContext = customerOrders;
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.ToString());
    }
}
```

```
Private context As NorthwindEntities
Private customerId As String = "ALFKI"

' Replace the host server and port number with the values
' for the test server hosting your Northwind data service instance.
Private svcUri As Uri = New Uri("http://localhost:12345/Northwind.svc")

Private Sub Window1_Loaded(ByVal sender As Object, ByVal e As RoutedEventArgs)
    Try
        ' Instantiate the DataServiceContext.
        context = New NorthwindEntities(svcUri)

        ' Define a LINQ query that returns Orders and
        ' Order_Details for a specific customer.
        Dim ordersQuery = From o In context.Orders.Expand("Order_Details") _
                              Where o.Customer.CustomerID = customerId _
                              Select o

        ' Create an DataServiceCollection(Of T) based on
        ' execution of the LINQ query for Orders.
        Dim customerOrders As DataServiceCollection(Of Order) = New  _
            DataServiceCollection(Of Order)(ordersQuery)

        ' Make the DataServiceCollection<T> the binding source for the Grid.
        Me.orderItemsGrid.DataContext = customerOrders
    Catch ex As Exception
        MessageBox.Show(ex.ToString())
    End Try
End Sub
```

5. Insert the following code that saves changes into the `MainWindow` class:

```
private void buttonSaveChanges_Click(object sender, RoutedEventArgs e)
{
    try
    {
        // Save changes made to objects tracked by the context.
        context.SaveChanges();
    }
    catch (DataServiceRequestException ex)
    {
        MessageBox.Show(ex.ToString());

    }
}
private void buttonClose_Click(object sender, RoutedEventArgs e)
{
    this.Close();
}
```

```
Private Sub buttonSaveChanges_Click(ByVal sender As Object, ByVal e As RoutedEventArgs)
    Try
        ' Save changes made to objects tracked by the context.
        context.SaveChanges()
    Catch ex As DataServiceRequestException
        MessageBox.Show(ex.ToString())
    End Try
End Sub
Private Sub buttonClose_Click(ByVal sender As Object, ByVal a As RoutedEventArgs)
    Me.Close()
End Sub
```

To build and run the NorthwindClient application

1. In **Solution Explorer**, right-click the NorthwindClient project and select **Set as startup project**.

2. Press **F5** to start the application.

   This builds the solution and starts the client application. Data is requested from the service and displayed in the console.

3. Edit a value in the **Quantity** column of the data grid, and then click **Save**.

   Changes are saved to the data service.

   > **NOTE**
   > This version of the NorthwindClient application does not support adding and deleting of entities.

Next Steps

You have successfully created the client application that accesses the sample Northwind OData feed. You've also completed the WCF Data Services quickstart!

For more information about accessing an OData feed from a .NET Framework application, see WCF Data Services Client Library.

See Also

- Getting Started
- Resources

Application Scenarios (WCF Data Services)

8/31/2018 • 2 minutes to read • Edit Online

WCF Data Services supports a core set of scenarios for exposing and consuming data as Open Data Protocol (OData) feeds. This topic points you to the topics relevant to these scenarios.

Expose relational data from a database as an OData feed.

- Quickstart
- Exposing Your Data as a Service
- How to: Create a Data Service Using an ADO.NET Entity Framework Data Source

Expose arbitrary CLR data classes as an OData feed.

- Exposing Your Data as a Service
- How to: Create a Data Service Using the Reflection Provider
- Data Services Providers

Consume an OData feed in a .NET Framework-based client application.

- Quickstart
- Using a Data Service in a Client Application
- WCF Data Services Client Library

Consume an OData feed in a Silverlight-based client application.

- WCF Data Services (Silverlight)
- Asynchronous Operations
- How to: Bind Data Service Data to Controls (WCF Data Services/Silverlight)

Consume an OData feed in an AJAX-based client application.

- Using a Data Service in a Client Application
- OData: URI Conventions
- OData: JavaScript Object Notation (JSON) Format

Create an end-to-end data solution that uses OData to transfer data between client and server.

- Quickstart
- Using a Data Service in a Client Application
- WCF Data Services Client Library

Create a .NET Framework-based client application that consumes an OData feed asynchronously to avoid latency issues on the client.

- How to: Execute Asynchronous Data Service Queries
- Asynchronous Operations

- WCF Data Services (Silverlight)

Expose and consume an OData feed with a binary large object that is accessed and changed as a stream.

- Streaming Provider
- Working with Binary Data

Bind OData feeds to controls in a Windows Presentation Framework (WPF) application.

- Binding Data to Controls
- How to: Bind Data to Windows Presentation Foundation Elements
- How to: Bind Data Using a Project Data Source

Intercept incoming messages to the data service to perform data validation and role-based filtering of queries.

- How to: Intercept Data Service Messages
- Interceptors

Create endpoints on a data service to enable custom service behaviors.

- How to: Define a Service Operation
- Service Operations

See Also

Quickstart
Resources

WCF Data Services Resources

8/31/2018 • 2 minutes to read • Edit Online

WCF Data Services introductory topics can be found in the following locations. Many of these topics also contain links to related topics that contain more detailed information.

Overview
Provides an overview of the features and functionality available in WCF Data Services.

Getting Started
Describes how to design and access a data service by using WCF Data Services, as illustrated by the quickstart.

Application Scenarios
Provides a task-based approach to creating WCF Data Services and applications that access Open Data Protocol (OData) feeds.

Visual Studio and .NET Framework Glossary
Includes terms used in WCF Data Services and OData documentation.

External Resources

The following external resources provide additional information and support for creating WCF Data Services applications that expose and consume OData feeds.

WCF Data Services Forum
Data programming support for WCF Data Services developers.

WCF Data Services Team Blog
Blog that contains updates and discussion of WCF Data Services features and functionality.

OData Web Site
The primary source of information about OData.

OData SDK
Contains sample services, samples, and programming libraries that enable you to work with OData feeds.

OData Blog
Blog that contains discussions about OData programming.

Data Access and Storage Developer Center
Central location for finding information and resources for working with data.

Data Platform How Do I? Videos: WCF Data Services Series
Contains a set of video screen casts that demonstrate how to build and access data services.

Overview: WCF Data Services
A white paper that provides more high-level information about the benefits of WCF Data Services.

Using Microsoft WCF Data Services
A white paper that provides additional information and examples for creating data services and accessing data services in client applications.

The Entity-Relationship Model: Toward a Unified View of Data by Peter Pin-Shan Chen, Massachusetts Institute of Technology
Describes the basis for the entity-relational model that is implemented by WCF Data Services. Written in 1976,

this is one of the most frequently cited papers in the computer field.

See Also

Getting Started

Defining WCF Data Services

8/29/2018 • 2 minutes to read • Edit Online

This section describes how to create and configure WCF Data Services to expose data as an Open Data Protocol (OData) feed. For more information about the basic steps required to create a data service, see Exposing Your Data as a Service.

In This Section

Configuring the Data Service

Describes the data service configuration options provided by WCF Data Services.

Data Services Providers

Describes the provider models for exposing data as a data service.

Service Operations

Describes how to define service operations that expose methods on the server.

Feed Customization

Describes how to create a mapping between entities in the data model defined by the data service provider and elements in the data feed.

Interceptors

Describes how to define interceptor methods to perform custom business logic on requests to the data service.

Developing and Deploying WCF Data Services

Describes how to develop and deploy a data service by using Visual Studio.

Securing WCF Data Services

Describes authentication and authorization for the data service and other security considerations.

Hosting the Data Service

Describes how to select a host for your data service.

Data Service Versioning

Describes how to work with different versions of the OData.

WCF Data Services Protocol Implementation Details

Describes optional functionalities of the OData protocol that are not currently implemented by WCF Data Services.

See Also

- WCF Data Services Client Library
- Accessing Data Service Resources
- Getting Started

Configuring the Data Service (WCF Data Services)

10/16/2018 • 7 minutes to read • Edit Online

With WCF Data Services, you can create data services that expose Open Data Protocol (OData) feeds. Data in these feeds can come from a variety of data sources. WCF Data Services uses data providers to expose this data as an OData feed. These providers include an Entity Framework provider, a reflection provider, and a set of custom data service provider interfaces. The provider implementation defines the data model for the service. For more information, see Data Services Providers.

In WCF Data Services, a data service is a class that inherits from the DataService<T> class, where the type of the data service is the entity container of the data model. This entity container has one or more properties that return an IQueryable<T>, which are used to access entity sets in the data model.

The behaviors of the data service are defined by the members of the DataServiceConfiguration class, and by members of the DataServiceBehavior class, which is accessed from the DataServiceBehavior property of the DataServiceConfiguration class. The DataServiceConfiguration class is supplied to the `InitializeService` method that is implemented by the data service, as in the following implementation of a Northwind data service:

```
// This method is called only once to initialize service-wide policies.
public static void InitializeService(DataServiceConfiguration config)
{
    // Set the access rules of feeds exposed by the data service, which is
    // based on the requirements of client applications.
    config.SetEntitySetAccessRule("Customers", EntitySetRights.AllRead);
    config.SetEntitySetAccessRule("Employees", EntitySetRights.ReadSingle);
    config.SetEntitySetAccessRule("Orders", EntitySetRights.All
        | EntitySetRights.WriteAppend
        | EntitySetRights.WriteMerge);
    config.SetEntitySetAccessRule("Order_Details", EntitySetRights.All);
    config.SetEntitySetAccessRule("Products", EntitySetRights.All);

    // Set page size defaults for the data service.
    config.SetEntitySetPageSize("Orders", 20);
    config.SetEntitySetPageSize("Order_Details", 50);
    config.SetEntitySetPageSize("Products", 50);

    // Paging requires v2 of the OData protocol.
    config.DataServiceBehavior.MaxProtocolVersion =
        System.Data.Services.Common.DataServiceProtocolVersion.V2;
}
```

```
' This method is called only once to initialize service-wide policies.
Public Shared Sub InitializeService(ByVal config As DataServiceConfiguration)
    ' Set the access rules of feeds exposed by the data service, which is
    ' based on the requirements of client applications.
    config.SetEntitySetAccessRule("Customers", EntitySetRights.AllRead)
    config.SetEntitySetAccessRule("Employees", EntitySetRights.ReadSingle)
    config.SetEntitySetAccessRule("Orders", EntitySetRights.AllRead _
        Or EntitySetRights.WriteAppend _
        Or EntitySetRights.WriteMerge)
    config.SetEntitySetAccessRule("Order_Details", EntitySetRights.All)
    config.SetEntitySetAccessRule("Products", EntitySetRights.ReadMultiple)

    ' Set page size defaults for the data service.
    config.SetEntitySetPageSize("Orders", 20)
    config.SetEntitySetPageSize("Order_Details", 50)
    config.SetEntitySetPageSize("Products", 50)

    ' Paging requires v2 of the OData protocol.
    config.DataServiceBehavior.MaxProtocolVersion = _
        System.Data.Services.Common.DataServiceProtocolVersion.V2
End Sub
```

Data Service Configuration Settings

The DataServiceConfiguration class enables you to specify the following data service behaviors:

MEMBER	BEHAVIOR
AcceptCountRequests	Enables you to disable count requests that are submitted to the data service by using the `$count` path segment and the `$inlinecount` query option. For more information, see OData: URI Conventions.
AcceptProjectionRequests	Enables you to disable support for data projection in requests that are submitted to the data service by using the `$select` query option. For more information, see OData: URI Conventions.
EnableTypeAccess	Enables a data type to be exposed in the metadata for a dynamic metadata provider defined by using the IDataServiceMetadataProvider interface.
EnableTypeConversion	Enables you to specify whether the data service runtime should convert the type that is contained in the payload to the actual property type that is specified in the request.
InvokeInterceptorsOnLinkDelete	Enables you to specify whether or not registered change interceptors are invoked on the related entities when a relationship link between two entities is deleted.
MaxBatchCount	Enables you to limit the number of change sets and query operations that are allowed in a single batch. For more information, see OData: Batch and Batching Operations.
MaxChangesetCount	Enables you to limit the number of changes that can be included in a single change set. For more information, see How to: Enable Paging of Data Service Results.

MEMBER	BEHAVIOR
MaxExpandCount	Enables you to limit the size of a response by limiting the number of related entities that can be included in a single request by using the `$expand` query operator. For more information, see OData: URI Conventions and Loading Deferred Content.
MaxExpandDepth	Enables you to limit the size of a response by limiting the depth of the graph of related entities that can be included in a single request by using the `$expand` query operator. For more information, see OData: URI Conventions and Loading Deferred Content.
MaxObjectCountOnInsert	Enables you to limit the number of entities to be inserted that can be contained in a single POST request.
MaxProtocolVersion	Defines the version of the Atom protocol that is used by the data service. When the value of the MaxProtocolVersion is set to a value less than the maximum value of DataServiceProtocolVersion, the latest functionality of WCF Data Services is not available to clients accessing the data service. For more information, see Data Service Versioning.
MaxResultsPerCollection	Enables you to limit the size of a response by limiting the number of entities in each entity set that is returned as a data feed.
RegisterKnownType	Adds a data type to the list of types that are recognized by the data service.
SetEntitySetAccessRule	Sets the access rights for entity set resources that are available on the data service. An asterisk (`*`) value can be supplied for the name parameter to set access for all remaining entity sets to the same level. We recommend that you set access to entity sets to provide the least privilege access to data service resources that are required by client applications. For more information, see Securing WCF Data Services. For examples of the minimum access rights required for a given URI and HTTP action, see the table in the Minimum Resource Access Requirements section.
SetEntitySetPageSize	Sets the maximum page size for an entity set resource. For more information, see How to: Enable Paging of Data Service Results.
SetServiceOperationAccessRule	Sets the access rights for service operations that are defined on the data service. For more information, see Service Operations. An asterisk (`*`) value can be supplied for the name parameter to set access for all service operations to the same level. We recommend that you set access to service operations to provide the least privilege access to data service resources that are required by client applications. For more information, see Securing WCF Data Services.

MEMBER	BEHAVIOR
UseVerboseErrors	This configuration property enables you to more easily troubleshoot a data service by returning more information in the error response message. This option is not intended to be used in a production environment. For more information, see Developing and Deploying WCF Data Services.

Minimum Resource Access Requirements

The following table details the minimum entity set rights that must be granted to execute a specific operation. Path examples are based on the Northwind data service that is created when you complete the quickstart. Because both the EntitySetRights enumeration and the ServiceOperationRights enumeration are defined by using the FlagsAttribute, you can use a logical OR operator to specify multiple permissions for a single entity set or operation. For more information, see How to: Enable Access to the Data Service.

PATH/ACTION	`GET`	`DELETE`	`MERGE`	`POST`	`PUT`
`/Customers`	ReadMultiple	Not supported	Not supported	WriteAppend	Not supported
`/Customers('ALFKI')`	ReadSingle	ReadSingle and WriteDelete	ReadSingle and WriteMerge	n/a	ReadSingle and WriteReplace
`/Customers('ALFKI'`	`Customers` : ReadSingle -and- `Orders` : ReadMultiple	Not supported	Not supported	`Customers` : ReadSingle and WriteMerge or WriteReplace -and- `Orders` `:` and WriteAppend	Not supported
`/Customers('ALFKI'` `643)`	`Customers` : ReadSingle -and- `Orders` : ReadSingle	`Customers` : ReadSingle -and- `Orders` : ReadSingle and WriteDelete	`Customers` : ReadSingle -and- `Orders` : ReadSingle and WriteMerge	Not supported	`Customers` : ReadSingle -and- `Orders` : ReadSingle and WriteReplace
`/Orders(10643)/Cus`	`Customers` : ReadSingle -and- `Orders` : ReadSingle	`Customers` : ReadSingle and WriteDelete -and- `Orders` : ReadSingle	`Customers` : ReadSingle and WriteMerge; -and- `Orders` : ReadSingle	`Customers` : WriteAppend -and- `Orders` : WriteAppend and ReadSingle	Not supported

PATH/ACTION	GET	DELETE	MERGE	POST	PUT
`/Customers('ALFKI'` `ders`	`Customers` : ReadSingle -and- `Orders` : ReadMultiple	Not supported	Not supported	`Customers` : ReadSingle and WriteMerge or WriteReplace -and- `Orders` : ReadSingle	Not supported
`/Customers('ALFKI'` `ders(1064`	`Customers` : ReadSingle -and- `Orders` : ReadSingle	`Customers` : ReadSingle and WriteMerge or WriteReplace -and- `Orders` : ReadSingle	Not supported	Not supported	Not supported
`/Orders(10643)/$li` `r`	`Customers` : ReadSingle -and- `Orders` : ReadSingle	`Orders` : ReadSingle and WriteMerge or WriteReplace	`Customers` : ReadSingle -and- `Orders` : ReadSingle and WriteMerge	Not supported	`Customers` : ReadSingle; -and- `Orders` : ReadSingle and WriteReplace
`/Customers/$count`	ReadMultiple	Not supported	Not supported	Not supported	Not supported
`/Customers('ALFKI')/ContactName`	ReadSingle	Not supported	WriteMerge	Not supported	WriteReplace
`/Customers('ALFKI')/Address/StreetAddress/$value` 1	ReadSingle	WriteDelete	Not supported	Not supported	Not supported
`/Customers('ALFKI')/ContactName/$value`	ReadSingle	ReadSingle and WriteDelete	WriteMerge	Not supported	WriteReplace
`/Customers('ALFKI')/$value` 2	ReadSingle	Not supported	Not supported	Not supported	WriteReplace
`/Customers?$select=Orders/*&$expand=Orders`	`Customers` : ReadSingle -and- `Orders` : ReadMultiple	Not supported	Not supported	`Customers` : WriteAppend	Not supported

PATH/ACTION	`GET`	`DELETE`	`MERGE`	`POST`	`PUT`
`/Customers('ALFKI')/$select=Orders/*&$expand=Orders`	`Customers` : ReadSingle -and- `Orders` : ReadMultiple	Not supported	Not supported	Not supported	Not supported

[1] In this example, `Address` represents a complex type property of the `Customers` entity that has a property named `StreetAddress`. The model used by the Northwind data services does not explicitly define this complex type. When the data model is defined by using the Entity Framework provider, you can use the Entity Data Model tools to define such a complex type. For more information, see How to: Create and Modify Complex Types.

[2] This URI is supported when a property that returns a binary large object (BLOB) is defined as the media resource that belongs to an entity that is a media link entry, which in this case, is `Customers`. For more information, see Streaming Provider.

Versioning Requirements

The following data service configuration behaviors require version 2 of the OData protocol, or later versions:

- Support for count requests.
- Support for the $select query option for projection.

For more information, see Data Service Versioning.

See Also

Defining WCF Data Services
Hosting the Data Service

How to: Develop a WCF data service running on IIS

8/31/2018 • 4 minutes to read • Edit Online

This topic shows how to use WCF Data Services to create a data service that is based on the Northwind sample database that is hosted by an ASP.NET Web application that is running on Internet Information Services (IIS). For an example of how to create the same Northwind data service as an ASP.NET Web application that runs on the ASP.NET Development Server, see the WCF Data Services quickstart.

NOTE

To create the Northwind data service, you must have installed the Northwind sample database on the local computer. To download this sample database, see the download page, Sample Databases for SQL Server.

This topic shows how to create a data service by using the Entity Framework provider. Other data services providers are available. For more information, see Data Services Providers.

After you create the service, you must explicitly provide access to data service resources. For more information, see How to: Enable Access to the Data Service.

Create the ASP.NET web application that runs on IIS

1. In Visual Studio, on the **File** menu, select **New** > **Project**.
2. In the **New Project** dialog box, select the **Installed** > [**Visual C#** or **Visual Basic**] > **Web** category.
3. Select the **ASP.NET Web Application** template.
4. Enter `NorthwindService` as the name of the project.
5. Click **OK**.
6. On the **Project** menu, select **NorthwindService Properties**.
7. Select the **Web** tab, and then select **Use Local IIS Web Server**.
8. Click **Create Virtual Directory** and then click **OK**.
9. From the command prompt with administrator privileges, execute one of the following commands (depending on the operating system):
   - 32-bit systems:

     ```
     "%windir%\Microsoft.NET\Framework\v3.0\Windows Communication Foundation\ServiceModelReg.exe" -i
     ```

   - 64-bit systems:

     ```
     "%windir%\Microsoft.NET\Framework64\v3.0\Windows Communication Foundation\ServiceModelReg.exe" -i
     ```

   This makes sure that Windows Communication Foundation (WCF) is registered on the computer.
10. From the command prompt with administrator privileges, execute one of the following commands (depending on the operating system):

- 32-bit systems:

  ```
  "%windir%\Microsoft.NET\Framework\v4.0.30319\aspnet_regiis.exe" -i -enable
  ```

- 64-bit systems:

  ```
  "%windir%\Microsoft.NET\Framework64\v4.0.30319\aspnet_regiis.exe" -i -enable
  ```

This makes sure that IIS runs correctly after WCF has been installed on the computer. You might have to also restart IIS.

11. When the ASP.NET application runs on IIS7, you must also perform the following steps:

    a. Open IIS Manager and navigate to the PhotoService application under **Default Web Site**.

    b. In **Features View**, double-click **Authentication**.

    c. On the **Authentication** page, select **Anonymous Authentication**.

    d. In the **Actions** pane, click **Edit** to set the security principal under which anonymous users will connect to the site.

    e. In the **Edit Anonymous Authentication Credentials** dialog box, select **Application pool identity**.

    IMPORTANT

    When you use the Network Service account, you grant anonymous users all the internal network access associated with that account.

12. By using SQL Server Management Studio, the sqlcmd.exe utility, or the Transact-SQL Editor in Visual Studio, execute the following Transact-SQL command against the instance of SQL Server that has the Northwind database attached:

    ```
    CREATE LOGIN [NT AUTHORITY\NETWORK SERVICE] FROM WINDOWS;
    GO
    ```

    This creates a login in the SQL Server instance for the Windows account used to run IIS. This enables IIS to connect to the SQL Server instance.

13. With the Northwind database attached, execute the following Transact-SQL commands:

    ```
    USE Northwind
    GO
    CREATE USER [NT AUTHORITY\NETWORK SERVICE]
    FOR LOGIN [NT AUTHORITY\NETWORK SERVICE] WITH DEFAULT_SCHEMA=[dbo];
    GO
    ALTER LOGIN [NT AUTHORITY\NETWORK SERVICE]
    WITH DEFAULT_DATABASE=[Northwind];
    GO
    EXEC sp_addrolemember 'db_datareader', 'NT AUTHORITY\NETWORK SERVICE'
    GO
    EXEC sp_addrolemember 'db_datawriter', 'NT AUTHORITY\NETWORK SERVICE'
    GO
    ```

    This grants rights to the new login, which enables IIS to read data from and write data to the Northwind database.

Define the data model

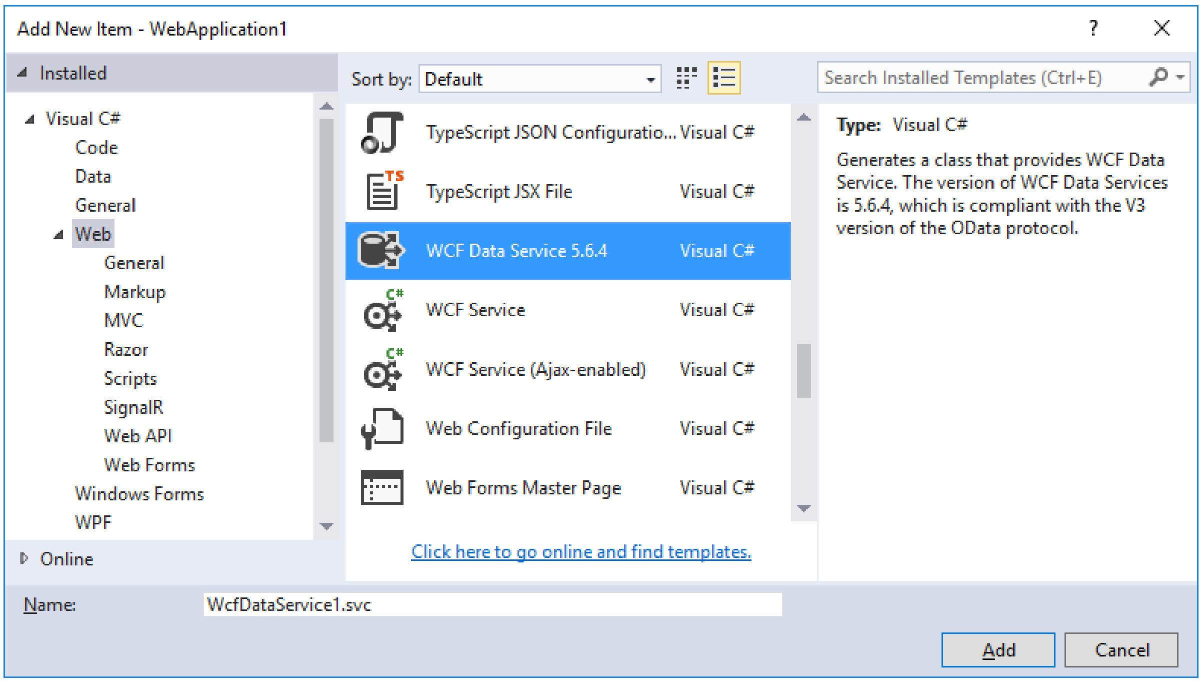
1. In **Solution Explorer**, right-click the name of the ASP.NET project, and then click **Add** > **New Item**.
2. In the **Add New Item** dialog box, select **ADO.NET Entity Data Model**.
3. For the name of the data model, type `Northwind.edmx`.
4. In the Entity Data Model Wizard, select **Generate from Database**, and then click **Next**.
5. Connect the data model to the database by doing one of the following steps, and then click **Next**:
   - If you do not have a database connection already configured, click **New Connection** and create a new connection. For more information, see How to: Create Connections to SQL Server Databases. This SQL Server instance must have the Northwind sample database attached.

     - or -

   - If you have a database connection already configured to connect to the Northwind database, select that connection from the list of connections.
6. On the final page of the wizard, select the check boxes for all tables in the database, and clear the check boxes for views and stored procedures.
7. Click **Finish** to close the wizard.

Create the data service

1. In **Solution Explorer**, right-click the name of your ASP.NET project, and then click **Add** > **New Item**.
2. In the **Add New Item** dialog box, select **WCF Data Service**.


Add New Item - WebApplication1

Sort by: Default

Type: Visual C#

Generates a class that provides WCF Data Service. The version of WCF Data Services is 5.6.4, which is compliant with the V3 version of the OData protocol.

TypeScript JSON Configuratio...	Visual C#
TypeScript JSX File	Visual C#
WCF Data Service 5.6.4	Visual C#
WCF Service	Visual C#
WCF Service (Ajax-enabled)	Visual C#
Web Configuration File	Visual C#
Web Forms Master Page	Visual C#

Name: WcfDataService1.svc

> **NOTE**
>
> The **WCF Data Service** template is available in Visual Studio 2015, but not in Visual Studio 2017.

3. For the name of the service, enter `Northwind`.

Visual Studio creates the XML markup and code files for the new service. By default, the code-editor window opens. In **Solution Explorer**, the service has the name, Northwind, and the extension .svc.cs or .svc.vb.

4. In the code for the data service, replace the comment `/* TODO: put your data source class name here */` in the definition of the class that defines the data service with the type that is the entity container of the data model, which in this case is `NorthwindEntities`. The class definition should look this the following:

```
public class Northwind : DataService<NorthwindEntities>
```

```
Public Class Northwind
    Inherits DataService(Of NorthwindEntities)
```

See also

- Exposing Your Data as a Service

How to: Enable Access to the Data Service (WCF Data Services)

8/31/2018 • 2 minutes to read • Edit Online

In WCF Data Services, you must explicitly grant access to the resources that are exposed by a data service. This means that after you create a new data service, you must still explicitly provide access to individual resources as entity sets. This topic shows how to enable read and write access to five of the entity sets in the Northwind data service that is created when you complete the quickstart. Because the EntitySetRights enumeration is defined by using the FlagsAttribute, you can use a logical OR operator to specify multiple permissions for a single entity set.

NOTE

Any client that can access the ASP.NET application can also access the resources exposed by the data service. In a production data service, to prevent unauthorized access to resources, you should also secure the application itself. For more information, see NIB: ASP.NET Security.

To enable access to the data service

- In the code for the data service, replace the placeholder code in the `InitializeService` function with the following:

```
 // Grant only the rights needed to support the client application.
config.SetEntitySetAccessRule("Orders", EntitySetRights.AllRead
     | EntitySetRights.WriteMerge
     | EntitySetRights.WriteReplace );
 config.SetEntitySetAccessRule("Order_Details", EntitySetRights.AllRead
     | EntitySetRights.AllWrite);
 config.SetEntitySetAccessRule("Customers", EntitySetRights.AllRead);
```

```
' Grant only the rights needed to support the client application.
config.SetEntitySetAccessRule("Orders", EntitySetRights.AllRead _
     Or EntitySetRights.WriteMerge _
     Or EntitySetRights.WriteReplace)
config.SetEntitySetAccessRule("Order_Details", EntitySetRights.AllRead _
    Or EntitySetRights.AllWrite)
config.SetEntitySetAccessRule("Customers", EntitySetRights.AllRead)
```

This enables clients to have read and write access to the `Orders` and `Order_Details` entity sets and read-only access to the `Customers` entity sets.

See Also

How to: Develop a WCF Data Service Running on IIS
Configuring the Data Service

How to: Enable Paging of Data Service Results (WCF Data Services)

5/2/2018 • 2 minutes to read • Edit Online

WCF Data Services enables you to limit the number of entities returned by a data service query. Page limits are defined in the method that is called when the service is initialized and can be set separately for each entity set.

When paging is enabled, the final entry in the feed contains a link to the next page of data. For more information, see Configuring the Data Service.

This topic shows how to modify a data service to enable paging of returned `Customers` and `Orders` entity sets. The example in this topic uses the Northwind sample data service. This service is created when you complete the WCF Data Services quickstart.

How to enable paging of returned Customers and Orders entity sets

- In the code for the data service, replace the placeholder code in the `InitializeService` function with the following:

```
// Set page size defaults for the data service.
config.SetEntitySetPageSize("Orders", 20);
config.SetEntitySetPageSize("Order_Details", 50);
config.SetEntitySetPageSize("Products", 50);

// Paging requires v2 of the OData protocol.
config.DataServiceBehavior.MaxProtocolVersion =
    System.Data.Services.Common.DataServiceProtocolVersion.V2;
```

```
' Set page size defaults for the data service.
config.SetEntitySetPageSize("Orders", 20)
config.SetEntitySetPageSize("Order_Details", 50)
config.SetEntitySetPageSize("Products", 50)

' Paging requires v2 of the OData protocol.
config.DataServiceBehavior.MaxProtocolVersion = _
    System.Data.Services.Common.DataServiceProtocolVersion.V2
```

See Also

Loading Deferred Content
How to: Load Paged Results

Data Services Providers (WCF Data Services)

5/2/2018 • 2 minutes to read • Edit Online

WCF Data Services supports multiple provider models for exposing data as an Open Data Protocol (OData) feed. This topic provides information to enable you to select the best WCF Data Services provider for your data source.

Data Source Providers

WCF Data Services supports the following providers for defining the data model of a data service.

PROVIDER	DESCRIPTION
Entity Framework provider	This provider uses the ADO.NET Entity Framework to enable you to use relational data with a data service by defining a data model that maps to relational data. Your data source can be SQL Server or any other data source with third-party provider support for the Entity Framework. You should use the Entity Framework provider when you have a relational data source, such as a SQL Server database. For more information, see Entity Framework Provider.
Reflection provider	This provider uses reflection to enable you to define a data model based on existing data classes that can be exposed as instances of the IQueryable<T> interface. Updates are enabled by implementing the IUpdatable interface. You should use this provider when you have static data classes that are defined at runtime, such as those generated by LINQ to SQL or defined by a typed DataSet. For more information, see Reflection Provider.
Custom Data Service Providers	WCF Data Services includes a set of providers that enable you to dynamically define a data model based on late-bound data types. You should implement these interfaces when the data being exposed is not known when the application is being designed or when the Entity Framework or reflection providers are not sufficient. For more information, see Custom Data Service Providers.

Other Data Service Providers

WCF Data Services has the following additional data service provider that enhances the performance of a data source defined by using one of the other providers.

PROVIDER	DESCRIPTION
Streaming provider	This provider enables you to expose binary large object data types by using WCF Data Services. A streaming provider is created by implementing the IDataServiceStreamProvider interface. This provider can be implemented together with any data source provider. For more information, see Streaming Provider.

See Also

Defining WCF Data Services
Configuring the Data Service
Hosting the Data Service

Entity Framework Provider (WCF Data Services)

8/31/2018 • 2 minutes to read • Edit Online

Like WCF Data Services, the ADO.NET Entity Framework is based on the Entity Data Model, which is a type of entity-relationship model. The Entity Framework translates operations against its implementation of the Entity Data Model, which is called the *conceptual model*, into equivalent operations against a data source. This makes the Entity Framework an ideal provider for data services that are based on relational data, and any database that has a data provider that supports the Entity Framework can be used with WCF Data Services. For a list of the data sources that currently support the Entity Framework, see Third-Party Providers for the Entity Framework.

In a conceptual model, the entity container is the root of the service. You must define a conceptual model in the Entity Framework before the data can be exposed by a data service. For more information, see How to: Create a Data Service Using an ADO.NET Entity Framework Data Source.

WCF Data Services supports the optimistic concurrency model by enabling you to define a concurrency token for an entity. This concurrency token, which includes one or more properties of the entity, is used by the data service to determine whether a change has occurred in the data that is being requested, updated, or deleted. When token values obtained from the eTag in the request differ from the current values of the entity, an exception is raised by the data service. To indicate that a property is part of the concurrency token, you must apply the attribute `ConcurrencyMode="Fixed"` in the data model defined by the Entity Framework provider. The concurrency token cannot include a key property or a navigation property. For more information, see Updating the Data Service.

To learn more about the Entity Framework, see Entity Framework Overview.

See Also

Data Services Providers
Reflection Provider
Entity Data Model

How to: Create a Data Service Using an ADO.NET Entity Framework Data Source (WCF Data Services)

9/17/2018 • 2 minutes to read • Edit Online

WCF Data Services exposes entity data as a data service. This entity data is provided by the ADO.NETEntity Framework when the data source is a relational database. This topic shows you how to create an Entity Framework-based data model in a Visual Studio Web application that is based on an existing database and use this data model to create a new data service.

The Entity Framework also provides a command line tool that can generate an Entity Framework model outside of a Visual Studio project. For more information, see How to: Use EdmGen.exe to Generate the Model and Mapping Files.

To add an Entity Framework model that is based on an existing database to an existing Web application

1. On the **Project** menu, click **Add** > **New Item**.
2. In the **Templates** pane, click the **Data** category, and then select **ADO.NET Entity Data Model**.
3. Enter the model name and then click **Add**.

   The first page of the Entity Data Model Wizard is displayed.

4. In the **Choose Model Contents** dialog box, select **Generate from database**. Then click **Next**.
5. Click the **New Connection** button.
6. In the **Connection Properties** dialog box, type your server name, select the authentication method, type the database name, and then click **OK**.

   The **Choose Your Data Connection** dialog box is updated with your database connection settings.

7. Ensure that the **Save entity connection settings in App.Config as:** checkbox is checked. Then click **Next**.
8. In the **Choose Your Database Objects** dialog box, select all of database objects that you plan to expose in the data service.

   > **NOTE**
   >
   > Objects included in the data model are not automatically exposed by the data service. They must be explicitly exposed by the service itself. For more information, see Configuring the Data Service.

9. Click **Finish** to complete the wizard.

   This creates a default data model based on the specific database. The Entity Framework enables to customize the data model. For more information, see Tasks.

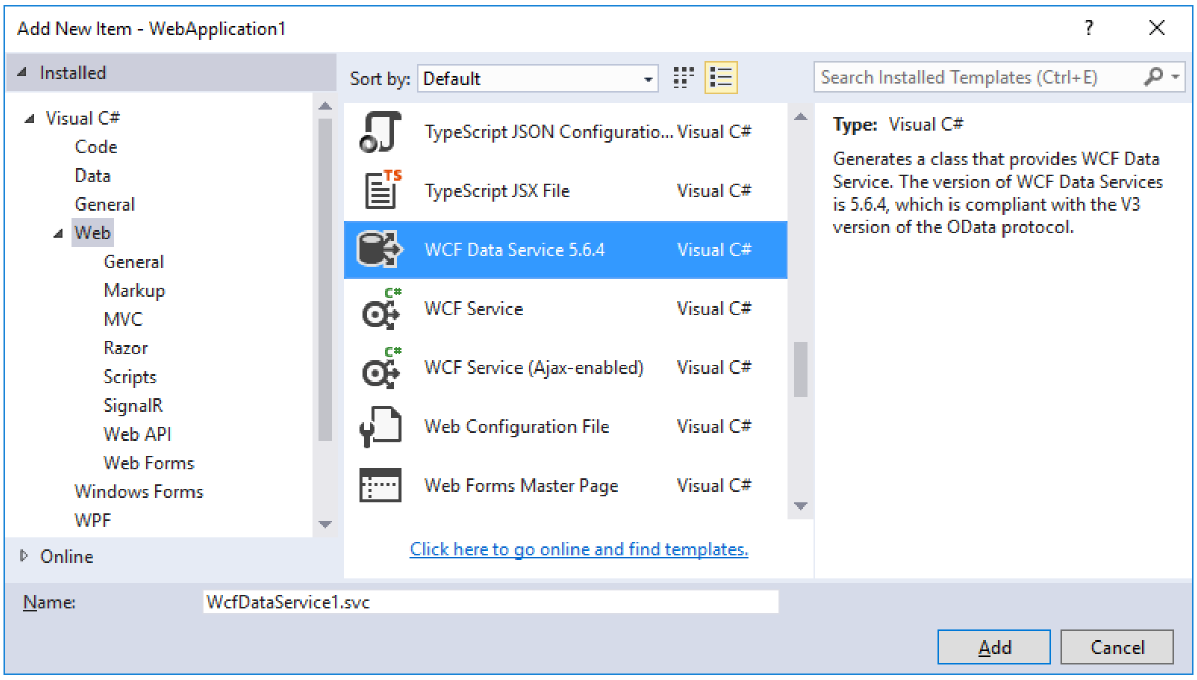
To create the data service by using the new data model

1. In Visual Studio, open the .edmx file that represents the data model.

2. In the **Model Browser**, right-click the model, click **Properties**, and then note the name of the entity container.

3. In **Solution Explorer**, right-click the name of your ASP.NET project, and then click **Add** > **New Item**.

4. In the **Add New Item** dialog box, select the **WCF Data Service** template in the **Web** category.


Add New Item - WebApplication1
Installed
Visual C#
Code
Data
General
Web
General
Markup
MVC
Razor
Scripts
SignalR
Web API
Web Forms
Windows Forms
WPF
Online
Sort by: Default
Search Installed Templates (Ctrl+E)
TypeScript JSON Configuratio... Visual C#
TypeScript JSX File Visual C#
WCF Data Service 5.6.4 Visual C#
WCF Service Visual C#
WCF Service (Ajax-enabled) Visual C#
Web Configuration File Visual C#
Web Forms Master Page Visual C#
Click here to go online and find templates.
Add
Cancel

Type: Visual C#

Generates a class that provides WCF Data Service. The version of WCF Data Services is 5.6.4, which is compliant with the V3 version of the OData protocol.

Name: WcfDataService1.svc

NOTE

The **WCF Data Service** template is available in Visual Studio 2015, but not in Visual Studio 2017.

5. Supply a name for the service, and then click **OK**.

   Visual Studio creates the XML markup and code files for the new service. By default, the code-editor window opens.

6. In the code for the data service, replace the comment `/* TODO: put your data source class name here */` in the definition of the class that defines the data service with the type that inherits from the ObjectContext class and that is the entity container of the data model, which was noted in step 2.

7. In the code for the data service, enable authorized clients to access the entity sets that the data service exposes. For more information, see Creating the Data Service.

8. To test the Northwind.svc data service by using a Web browser, follow the instructions in the topic Accessing the Service from a Web Browser.

See Also

- Defining WCF Data Services
- Data Services Providers
- How to: Create a Data Service Using the Reflection Provider
- How to: Create a Data Service Using a LINQ to SQL Data Source

Reflection Provider (WCF Data Services)

5/2/2018 • 5 minutes to read • Edit Online

In addition to exposing data from a data model through the Entity Framework, WCF Data Services can expose data that is not strictly defined in an entity-based model. The reflection provider exposes data in classes that return types that implement the IQueryable<T> interface. WCF Data Services uses reflection to infer a data model for these classes and can translate address-based queries against resources into language integrated query (LINQ)-based queries against the exposed IQueryable<T> types.

> **NOTE**
>
> You can use the AsQueryable method to return an IQueryable<T> interface from any class that implements the IEnumerable<T> interface. This enables most generic collection types to be used as a data source for your data service.

The reflection provider supports type hierarchies. For more information, see How to: Create a Data Service Using the Reflection Provider.

Inferring the Data Model

When you create the data service, the provider infers the data model by using reflection. The following list shows how the reflection provider infers the data model:

- Entity container - the class that exposes the data as properties that return an IQueryable<T> instance. When you address a reflection-based data model, the entity container represents the root of the service. Only one entity container class is supported for a given namespace.
- Entity sets - properties that return IQueryable<T> instances are treated as entity sets. Entity sets are addressed directly as resources in the query. Only one property on the entity container can return an IQueryable<T> instance of a given type.
- Entity types - the type `T` of the IQueryable<T> that the entity set returns. Classes that are part of an inheritance hierarchy are translated by the reflection provider into an equivalent entity type hierarchy.
- Entity keys - each data class that is an entity type must have a key property. This property is attributed with the DataServiceKeyAttribute attribute (`[DataServiceKeyAttribute]`).

  > **NOTE**
  >
  > You should only apply the DataServiceKeyAttribute attribute to a property that can be used to uniquely identify an instance of the entity type. This attribute is ignored when applied to a navigation property.

- Entity type properties - other than the entity key, the reflection provider treats the accessible, non-indexer properties of a class that is an entity type as follows:
  - If the property returns a primitive type, then the property is assumed to be a property of an entity type.
  - If the property returns a type that is also an entity type, then the property is assumed to be a navigation property that represents the "one" end of a many-to-one or one-to-one relationship.
  - If the property returns an IEnumerable<T> of an entity type, then the property is assumed to be a

navigation property that represents the "many" end of a one-to-many or many-to-many relationship.

- If the return type of the property is a value type, then the property represents a complex type.

> **NOTE**
> Unlike a data model that is based on the entity-relational model, models that are based on the reflection provider do not understand relational data. You should use the Entity Framework to expose relational data through WCF Data Services.

Data Type Mapping

When a data model is inferred from .NET Framework classes, the primitive types in the data model are mapped to .NET Framework data types as follows:

.NET FRAMEWORK DATA TYPE	DATA MODEL TYPE
Byte `[]`	`Edm.Binary`
Boolean	`Edm.Boolean`
Byte	`Edm.Byte`
DateTime	`Edm.DateTime`
Decimal	`Edm.Decimal`
Double	`Edm.Double`
Guid	`Edm.Guid`
Int16	`Edm.Int16`
Int32	`Edm.Int32`
Int64	`Edm.Int64`
SByte	`Edm.SByte`
Single	`Edm.Single`
String	`Edm.String`

> **NOTE**
> .NET Framework nullable value types are mapped to the same data model types as the corresponding value types that cannot be assigned a null.

Enabling Updates in the Data Model

To allow updates to data that is exposed through this kind of data model, the reflection provider defines an

IUpdatable interface. This interface instructs the data service on how to persist updates to the exposed types. To enable updates to resources that are defined by the data model, the entity container class must implement the IUpdatable interface. For an example of an implementation of the IUpdatable interface, see How to: Create a Data Service Using a LINQ to SQL Data Source.

The IUpdatable interface requires that the following members be implemented so that updates can be propagated to the data source by using the reflection provider:

MEMBER	DESCRIPTION
AddReferenceToCollection	Provides the functionality to add an object to a collection of related objects that are accessed from a navigation property.
ClearChanges	Provides the functionality that cancels pending changes to the data.
CreateResource	Provides the functionality to create a new resource in the specified container.
DeleteResource	Provides the functionality to delete a resource.
GetResource	Provides the functionality to retrieve a resource that is identified by a specific query and type name.
GetValue	Provides the functionality to return the value of a property of a resource.
RemoveReferenceFromCollection	Provides the functionality to remove an object to a collection of related objects accessed from a navigation property.
ResetResource	Provides the functionality to update a specified resource.
ResolveResource	Provides the functionality to return the resource that is represented by a specific object instance.
SaveChanges	Provides the functionality to save all pending changes.
SetReference	Provides the functionality to set a related object reference by using a navigation property.
SetValue	Provides the functionality to set the value of the property of a resource.

Handling Concurrency

WCF Data Services supports an optimistic concurrency model by enabling you to define a concurrency token for an entity. This concurrency token, which includes one or more properties of the entity, is used by the data service to determine whether a change has occurred in the data that is being requested, updated, or deleted. When token values obtained from the eTag in the request differ from the current values of the entity, an exception is raised by the data service. The ETagAttribute is applied to an entity type to define a concurrency token in the reflection provider. The concurrency token cannot include a key property or a navigation property. For more information, see Updating the Data Service.

Using LINQ to SQL with the Reflection Provider

Because the Entity Framework is natively supported by default, it is the recommended data provider for using relational data with WCF Data Services. However, you can use the reflection provider to use LINQ to SQL classes with a data service. The Table<TEntity> result sets that are returned by methods on the DataContext generated by the LINQ to SQL Object Relational Designer (O/R Designer) implement the IQueryable<T> interface. This enables the reflection provider to access these methods and return entity data from SQL Server by using the generated LINQ to SQL classes. However, because LINQ to SQL does not implement the IUpdatable interface, you need to add a partial class that extends the existing DataContext partial class to add the IUpdatable implementation. For more information, see How to: Create a Data Service Using a LINQ to SQL Data Source.

See Also

Data Services Providers

How to: Create a Data Service Using the Reflection Provider (WCF Data Services)

5/2/2018 • 3 minutes to read • Edit Online

WCF Data Services enables you to define a data model that is based on arbitrary classes as long as those classes are exposed as objects that implement the IQueryable<T> interface. For more information, see Data Services Providers.

Example

The following example defines a data model that includes `Orders` and `Items`. The entity container class `OrderItemData` has two public methods that return IQueryable<T> interfaces. These interfaces are the entity sets of the `Orders` and `Items` entity types. An `Order` can include multiple `Items`, so the `Orders` entity type has an `Items` navigation property that returns a collection of `Items` objects. The `OrderItemData` entity container class is the generic type of the DataService<T> class from which the `OrderItems` data service is derived.

NOTE

Because this example demonstrates an in-memory data provider and changes are not persisted outside of the current object instances, there is no benefit derived from implementing the IUpdatable interface. For an example that implements the IUpdatable interface, see How to: Create a Data Service Using a LINQ to SQL Data Source.

```
using System;
using System.Collections.Generic;
using System.Data.Services;
using System.Data.Services.Common;
using System.Linq;

namespace CustomDataServiceClient
{
    [DataServiceKeyAttribute("OrderId")]
    public class Order
    {
        public int OrderId { get; set; }
        public string Customer { get; set; }
        public IList<Item> Items { get; set; }
    }
    [DataServiceKeyAttribute("Product")]
    public class Item
    {
        public string Product { get; set; }
        public int Quantity { get; set; }
    }
    public partial class OrderItemData
    {
        #region Populate Service Data
        static IList<Order> _orders;
        static IList<Item> _items;
        static OrderItemData()
        {
            _orders = new Order[]{
              new Order(){ OrderId=0, Customer = "Peter Franken", Items = new List<Item>()},
              new Order(){ OrderId=1, Customer = "Ana Trujillo", Items = new List<Item>()}};
            _items = new Item[]{
              new Item(){ Product="Chai", Quantity=10 },
              new Item(){ Product="Chang", Quantity=25 },
              new Item(){ Product="Aniseed Syrup", Quantity = 5 },
              new Item(){ Product="Chef Anton's Cajun Seasoning", Quantity=30}};
            _orders[0].Items.Add(_items[0]);
            _orders[0].Items.Add(_items[1]);
            _orders[1].Items.Add(_items[2]);
            _orders[1].Items.Add(_items[3]);
        }
        #endregion
        public IQueryable<Order> Orders
        {
            get { return _orders.AsQueryable<Order>(); }
        }
        public IQueryable<Item> Items
        {
            get { return _items.AsQueryable<Item>(); }
        }
    }
    public class OrderItems : DataService<OrderItemData>
    {
        // This method is called only once to initialize
        //service-wide policies.
        public static void InitializeService(IDataServiceConfiguration
                                             config)
        {
            config.SetEntitySetAccessRule("Orders", EntitySetRights.All);
            config.SetEntitySetAccessRule("Items", EntitySetRights.All);
        }
    }
}
```

```
Imports System
Imports System.Collections.Generic
Imports System.Data.Services
Imports System.Data.Services.Common
Imports System.Linq

Namespace CustomDataServiceClient
    <DataServiceKeyAttribute("OrderId")> _
    Public Class Order
        Private _orderId As Integer
        Private _customer As String
        Private _items As IList(Of Item)
        Public Property OrderId() As Integer
            Get
                Return _orderId
            End Get
            Set(ByVal value As Integer)
                _orderId = value
            End Set
        End Property
        Public Property Customer() As String
            Get
                Return _customer
            End Get
            Set(ByVal value As String)
                _customer = value
            End Set
        End Property
        Public Property Items() As IList(Of Item)
            Get
                Return _items
            End Get
            Set(ByVal value As IList(Of Item))
                _items = value
            End Set
        End Property
    End Class
    <EntityPropertyMappingAttribute("Product", "productname", _
        "orders", "http://schema.examples.microsoft.com/dataservices", True)> _
    <DataServiceKeyAttribute("Product")> _
    Public Class Item
        Private _product As String
        Private _quantity As Integer
        Public Property Product() As String
            Get
                Return _product
            End Get
            Set(ByVal value As String)
                _product = value
            End Set
        End Property
        Public Property Quantity() As Integer
            Get
                Return _quantity
            End Get
            Set(ByVal value As Integer)
                _quantity = value
            End Set
        End Property
    End Class
    Partial Public Class OrderItemData
#Region "Populate Service Data"
        Shared _orders As IList(Of Order)
        Shared _items As IList(Of Item)
        Sub New()
            _orders = New Order() { _
                New Order() With {.OrderId = 0, .Customer = "Peter Franken", .Items = New List(Of Item)()}, _
              New Order() With {.OrderId = 1, .Customer = "Ana Trujillo", .Items = New List(Of Item)()}}
            _items = New Item() { _
              New Item() With {.Product = "Chai", .Quantity = 10}, _
              New Item() With {.Product = "Chang", .Quantity = 25}, _
```

```
                New Item() With {.Product = "Aniseed Syrup", .Quantity = 5}, _
                New Item() With {.Product = "Chef Anton's Cajun Seasoning", .Quantity = 30}}
            _orders(0).Items.Add(_items(0))
            _orders(0).Items.Add(_items(1))
            _orders(1).Items.Add(_items(2))
            _orders(1).Items.Add(_items(3))
        End Sub
#End Region
        Public ReadOnly Property Orders() As IQueryable(Of Order)
            Get
                Return _orders.AsQueryable()
            End Get
        End Property
        Public ReadOnly Property Items() As IQueryable(Of Item)
            Get
                Return _items.AsQueryable()
            End Get
        End Property
    End Class
    Public Class OrderItems
        Inherits DataService(Of OrderItemData)
        ' This method is called only once to initialize
        ' service-wide policies.
        Shared Sub InitializeService(ByVal config As DataServiceConfiguration)
            config.SetEntitySetAccessRule("Orders", EntitySetRights.All)
            config.SetEntitySetAccessRule("Items", EntitySetRights.All)
            config.DataServiceBehavior.MaxProtocolVersion = DataServiceProtocolVersion.V2
        End Sub
    End Class
End Namespace
```

See Also

How to: Create a Data Service Using a LINQ to SQL Data Source
Data Services Providers
How to: Create a Data Service Using an ADO.NET Entity Framework Data Source

How to: Create a Data Service Using a LINQ to SQL Data Source (WCF Data Services)

8/29/2018 • 7 minutes to read • Edit Online

WCF Data Services exposes entity data as a data service. The reflection provider enables you to define a data model that is based on any class that exposes members that return an IQueryable<T> implementation. To be able to make updates to data in the data source, these classes must also implement the IUpdatable interface. For more information, see Data Services Providers. This topic shows you how to create LINQ to SQL classes that access the Northwind sample database by using the reflection provider, as well as how to create the data service that is based on these data classes.

To add LINQ to SQL classes to a project

1. From within a Visual Basic or C# application, on the **Project** menu, click **Add** > **New Item**.

2. Click the **LINQ to SQL Classes** template.

3. Change the name to **Northwind.dbml**.

4. Click **Add**.

   The Northwind.dbml file is added to the project and the Object Relational Designer (O/R Designer) opens.

5. In **Server**/**Database Explorer**, under Northwind, expand **Tables** and drag the `Customers` table onto the Object Relational Designer (O/R Designer).

   A `Customer` entity class is created and appears on the design surface.

6. Repeat step 6 for the `Orders`, `Order_Details`, and `Products` tables.

7. Right-click the new .dbml file that represents the LINQ to SQL classes and click **View Code**.

   This creates a new code-behind page named Northwind.cs that contains a partial class definition for the class that inherits from the DataContext class, which in this case is `NorthwindDataContext`.

8. Replace the contents of the Northwind.cs code file with the following code. This code implements the reflection provider by extending the DataContext and data classes generated by LINQ to SQL:

```
using System;
using System.ComponentModel;
using System.Collections;
using System.Linq;
using System.Reflection;
using System.Data.Linq;
using System.Data.Linq.Mapping;
using System.Data.Services;
using System.Data.Services.Common;

namespace NorthwindService
{
    // Define the key properties for the LINQ to SQL data classes.
    [DataServiceKeyAttribute("CustomerID")]
    public partial class Customer { }

    [DataServiceKeyAttribute("ProductID")]
    public partial class Product { }
```

```
[DataServiceKeyAttribute("OrderID")]
public partial class Order { }

[DataServiceKeyAttribute("OrderID", "ProductID")]
public partial class Order_Detail { }

#region IUpdatable implementation
// Define the IUpdatable implementation for LINQ to SQL.
public partial class NorthwindDataContext : IUpdatable
{
    // Creates an object in the container.
    object IUpdatable.CreateResource(string containerName, string fullTypeName)
    {
        Type t = Type.GetType(fullTypeName, true);
        ITable table = GetTable(t);
        object resource = Activator.CreateInstance(t);
        table.InsertOnSubmit(resource);
        return resource;
    }

    // Gets the object referenced by the resource.
    object IUpdatable.GetResource(IQueryable query, string fullTypeName)
    {
        object resource = query.Cast<object>().SingleOrDefault();

        // fullTypeName can be null for deletes
        if (fullTypeName != null && resource.GetType().FullName != fullTypeName)
            throw new ApplicationException("Unexpected type for this resource.");
        return resource;
    }


    // Resets the value of the object to its default value.
    object IUpdatable.ResetResource(object resource)
    {
        Type t = resource.GetType();
        MetaTable table = Mapping.GetTable(t);
        object dummyResource = Activator.CreateInstance(t);
        foreach (var member in table.RowType.DataMembers)
        {
            if (!member.IsPrimaryKey && !member.IsDeferred &&
                !member.IsAssociation && !member.IsDbGenerated)
            {
                object defaultValue = member.MemberAccessor.GetBoxedValue(dummyResource);
                member.MemberAccessor.SetBoxedValue(ref resource, defaultValue);
            }
        }
        return resource;
    }

    // Sets the value of the given property on the object.
    void IUpdatable.SetValue(object targetResource, string propertyName, object propertyValue)
    {
        MetaTable table = Mapping.GetTable(targetResource.GetType());
        MetaDataMember member = table.RowType.DataMembers.Single(x => x.Name == propertyName);
        member.MemberAccessor.SetBoxedValue(ref targetResource, propertyValue);
    }

    // Gets the value of a property on an object.
    object IUpdatable.GetValue(object targetResource, string propertyName)
    {
        MetaTable table = Mapping.GetTable(targetResource.GetType());
        MetaDataMember member =
            table.RowType.DataMembers.Single(x => x.Name == propertyName);
        return member.MemberAccessor.GetBoxedValue(targetResource);
    }

    // Sets the related object for a reference.
    void IUpdatable.SetReference(
        void IUpdatable.SetReference(
            object targetResource, string propertyName, object propertyValue)
        {
            ((IUpdatable)this).SetValue(targetResource, propertyName, propertyValue);
        }

        // Adds the object to the related objects collection.
        void IUpdatable.AddReferenceToCollection(
            object targetResource, string propertyName, object resourceToBeAdded)
        {
            PropertyInfo pi = targetResource.GetType().GetProperty(propertyName);
            if (pi == null)
                throw new Exception("Can't find property");
            IList collection = (IList)pi.GetValue(targetResource, null);
            collection.Add(resourceToBeAdded);
        }

        // Removes the object from the related objects collection.
        void IUpdatable.RemoveReferenceFromCollection(
            object targetResource, string propertyName, object resourceToBeRemoved)
        {
            PropertyInfo pi = targetResource.GetType().GetProperty(propertyName);
            if (pi == null)
                throw new Exception("Can't find property");
            IList collection = (IList)pi.GetValue(targetResource, null);
            collection.Remove(resourceToBeRemoved);
        }

        // Deletes the resource.
        void IUpdatable.DeleteResource(object targetResource)
        {
            ITable table = GetTable(targetResource.GetType());
            table.DeleteOnSubmit(targetResource);
        }

        // Saves all the pending changes.
        void IUpdatable.SaveChanges()
        {
            SubmitChanges();
        }

        // Returns the actual instance of the resource represented
        // by the resource object.
        object IUpdatable.ResolveResource(object resource)
        {
            return resource;
        }

        // Reverts all the pending changes.
        void IUpdatable.ClearChanges()
        {
            // Raise an exception as there is no real way to do this with LINQ to SQL.
            // Comment out the following line if you'd prefer a silent failure
            throw new NotSupportedException();
        }
    #endregion
    }
}
```

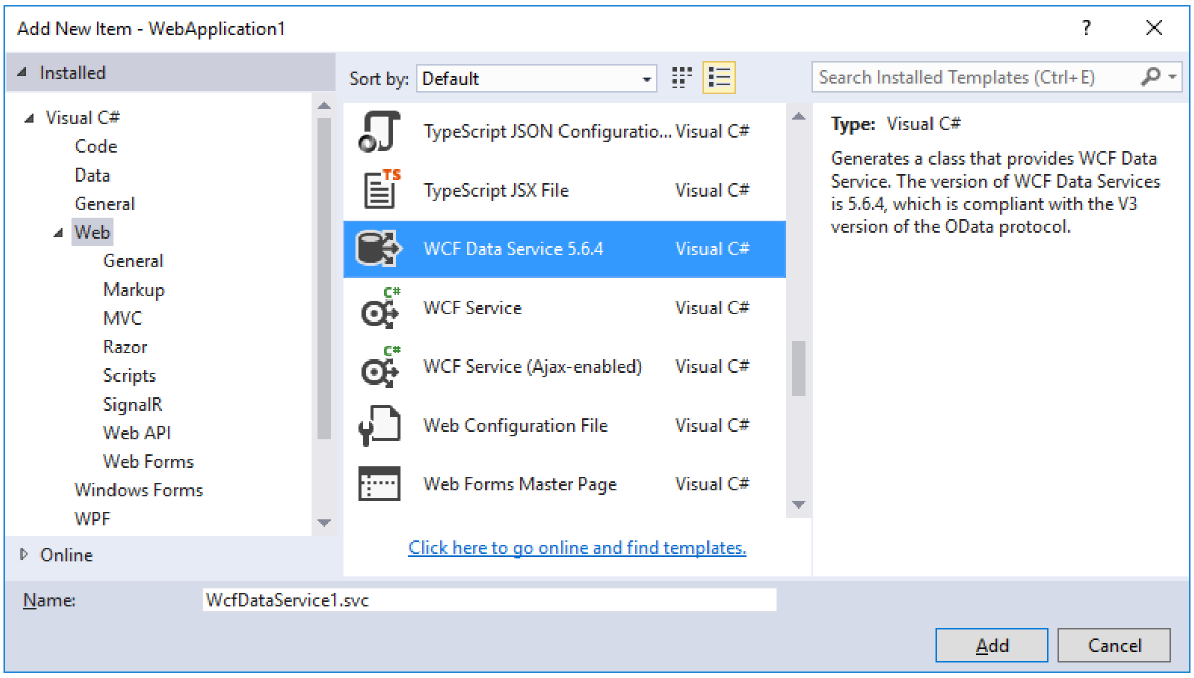
```
Imports System.ComponentModel
Imports System.Collections
Imports System.Linq
Imports System.Reflection
Imports System.Data.Linq
Imports System.Data.Linq.Mapping
Imports System.Data.Services
Imports System.Data.Services.Common
```

```
' Define the key properties for the LINQ to SQL data classes.
<DataServiceKeyAttribute("CustomerID")> _
Partial Public Class Customer
End Class
<DataServiceKeyAttribute("ProductID")> _
Partial Public Class Product
End Class
<DataServiceKeyAttribute("OrderID")> _
Partial Public Class Order
End Class
<DataServiceKeyAttribute("OrderID", "ProductID")> _
Partial Public Class Order_Detail
End Class
#Region "IUpdatable implementation"
' Define the IUpdatable implementation for LINQ to SQL.
Partial Public Class NorthwindDataContext
    Implements IUpdatable
    ' Creates an object in the container.
    Function CreateResource(ByVal containerName As String, ByVal fullTypeName As String) _
        As Object Implements IUpdatable.CreateResource

        Dim t = Type.GetType(fullTypeName, True)
        Dim table = GetTable(t)
        Dim resource = Activator.CreateInstance(t)
        table.InsertOnSubmit(resource)
        Return resource
    End Function
    ' Gets the object referenced by the resource.
    Function GetResource(ByVal query As IQueryable, ByVal fullTypeName As String) As Object _
     Implements IUpdatable.GetResource
        Dim resource = query.Cast(Of Object)().SingleOrDefault()

        ' fullTypeName can be null for deletes
        If fullTypeName IsNot Nothing AndAlso resource.GetType().FullName <> fullTypeName Then
            Throw New ApplicationException("Unexpected type for this resource.")
        End If
        Return resource
    End Function
    ' Resets the value of the object to its default value.
    Function ResetResource(ByVal resource As Object) As Object _
        Implements IUpdatable.ResetResource
        Dim t = resource.GetType()
        Dim table = Mapping.GetTable(t)
        Dim dummyResource = Activator.CreateInstance(t)
        For Each member In table.RowType.DataMembers

            If Not member.IsPrimaryKey AndAlso Not member.IsDeferred AndAlso _
                Not member.IsAssociation AndAlso Not member.IsDbGenerated Then
                Dim defaultValue = member.MemberAccessor.GetBoxedValue(dummyResource)
                member.MemberAccessor.SetBoxedValue(resource, defaultValue)
            End If
            Next
        Return resource
    End Function
    ' Sets the value of the given property on the object.
    Sub SetValue(ByVal targetResource As Object, ByVal propertyName As String, _
                 ByVal propertyValue As Object) Implements IUpdatable.SetValue
        Dim table = Mapping.GetTable(targetResource.GetType())
        Dim member = table.RowType.DataMembers.Single(Function(x) x.Name = propertyName)
        member.MemberAccessor.SetBoxedValue(targetResource, propertyValue)
    End Sub
    ' Gets the value of a property on an object.
    Function GetValue(ByVal targetResource As Object, ByVal propertyName As String) _
    As Object Implements IUpdatable.GetValue
        Dim table = Mapping.GetTable(targetResource.GetType())
        Dim member = _
        table.RowType.DataMembers.Single(Function(x) x.Name = propertyName)
        Return member.MemberAccessor.GetBoxedValue(targetResource)
    End Function
    ' Sets the related object for a reference.
    Sub SetReference(ByVal targetResource As Object, ByVal propertyName As String, _
                     ByVal propertyValue As Object) Implements IUpdatable.SetReference
        CType(Me, IUpdatable).SetValue(targetResource, propertyName, propertyValue)
    End Sub
' Adds the object to the related objects collection.
    Sub AddReferenceToCollection(ByVal targetResource As Object, ByVal propertyName As String, _
                                 ByVal resourceToBeAdded As Object) _
                                 Implements IUpdatable.AddReferenceToCollection
        Dim pi = targetResource.GetType().GetProperty(propertyName)
        If pi Is Nothing Then
            Throw New Exception("Can't find property")
        End If
        Dim collection = CType(pi.GetValue(targetResource, Nothing), IList)
        collection.Add(resourceToBeAdded)
    End Sub
    ' Removes the object from the related objects collection.
    Sub RemoveReferenceFromCollection(ByVal targetResource As Object, ByVal propertyName As String, _
                                      ByVal resourceToBeRemoved As Object) _
                                      Implements IUpdatable.RemoveReferenceFromCollection
        Dim pi = targetResource.GetType().GetProperty(propertyName)
        If pi Is Nothing Then
            Throw New Exception("Can't find property")
        End If
        Dim collection = CType(pi.GetValue(targetResource, Nothing), IList)
        collection.Remove(resourceToBeRemoved)
    End Sub
        ' Deletes the resource.
    Sub DeleteResource(ByVal targetResource As Object) _
    Implements IUpdatable.DeleteResource
        Dim table = GetTable(targetResource.GetType())
        table.DeleteOnSubmit(targetResource)
    End Sub
    ' Saves all the pending changes.
    Sub SaveChanges() Implements IUpdatable.SaveChanges
        SubmitChanges()
    End Sub
    ' Returns the actual instance of the resource represented
    ' by the resource object.
    Function ResolveResource(ByVal resource As Object) As Object Implements IUpdatable.ResolveResource
        Return resource
    End Function
        ' Reverts all the pending changes.
    Sub ClearChanges() Implements IUpdatable.ClearChanges
        ' Raise an exception as there is no real way to do this with LINQ to SQL.
        ' Comment out the following line if you'd prefer a silent failure
        Throw New NotSupportedException()
    End Sub
End Class
#End Region
```

To create a data service by using a LINQ to SQL-based data model

1. In **Solution Explorer**, right-click the name of your ASP.NET project, and then click **Add** > **New Item**.

2. In the **Add New Item** dialog box, select the **WCF Data Service** template from the **Web** category.


Add New Item - WebApplication1
Installed
Visual C#
Code
Data
General
Web
General
Markup
MVC
Razor
Scripts
SignalR
Web API
Web Forms
Windows Forms
WPF
Online
Search Installed Templates (Ctrl+E)
TypeScript JSON Configuratio... Visual C#
TypeScript JSX File Visual C#
WCF Data Service 5.6.4 Visual C#
WCF Service Visual C#
WCF Service (Ajax-enabled) Visual C#
Web Configuration File Visual C#
Web Forms Master Page Visual C#
Click here to go online and find templates.
Add
Cancel

Sort by: Default

Type: Visual C#

Generates a class that provides WCF Data Service. The version of WCF Data Services is 5.6.4, which is compliant with the V3 version of the OData protocol.

Name: WcfDataService1.svc

NOTE

The **WCF Data Service** template is available in Visual Studio 2015, but not in Visual Studio 2017.

3. Supply a name for the service, and then click **OK**.

   Visual Studio creates the XML markup and code files for the new service. By default, the code-editor window opens.

4. In the code for the data service, replace the comment `/* TODO: put your data source class name here */` in the definition of the class that defines the data service with the type that is the entity container of the data model, which in this case is `NorthwindDataContext`.

5. In the code for the data service, replace the placeholder code in the `InitializeService` function with the following:

```
config.SetEntitySetAccessRule("Customers", EntitySetRights.ReadMultiple);
config.SetEntitySetAccessRule("Orders", EntitySetRights.AllRead
                              | EntitySetRights.WriteMerge);
config.SetEntitySetAccessRule("Order_Details", EntitySetRights.AllRead
                              | EntitySetRights.AllWrite);
config.SetEntitySetAccessRule("Products", EntitySetRights.ReadMultiple);
```

```
config.SetEntitySetAccessRule("Customers", EntitySetRights.ReadMultiple)
config.SetEntitySetAccessRule("Orders", EntitySetRights.AllRead _
                             Or EntitySetRights.WriteMerge)
config.SetEntitySetAccessRule("Order_Details", EntitySetRights.AllRead _
                             Or EntitySetRights.AllWrite)
config.SetEntitySetAccessRule("Products", EntitySetRights.ReadMultiple)
```

   This enables authorized clients to access resources for the three specified entity sets.

6. To test the Northwind.svc data service by using a Web browser, follow the instructions in the topic Accessing the Service from a Web Browser.

See Also

- How to: Create a Data Service Using an ADO.NET Entity Framework Data Source
- How to: Create a Data Service Using the Reflection Provider
- Data Services Providers

Custom Data Service Providers (WCF Data Services)

8/31/2018 • 2 minutes to read • Edit Online

WCF Data Services includes a set of providers that enables you to define a data model based on late-bound data types.

PROVIDER	DESCRIPTION
Metadata provider	This is the core custom data service provider that enables you to define a custom data model at runtime by implementing the IDataServiceMetadataProvider interface.
Query provider	This provider enables you to execute queries against a custom data model that is defined by using the IDataServiceMetadataProvider interface. The query provider is created by implementing the IDataServiceQueryProvider interface.
Update provider	This provider enables you to make updates to types that are exposed in a custom data service provider and to manage concurrency. An update provider is created by implementing the IDataServiceUpdateProvider interface
Paging provider	This provider is used with the custom data service provider to enable server-driven paging support. A paging provider for a custom data service is created by implementing the IDataServicePagingProvider interface.
Streaming provider	This provider enables you to expose binary large object data types as a stream. A streaming provider is created by implementing the IDataServiceStreamProvider interface. The streaming provider can also be used with Entity Framework and reflection data source providers. For more information, see Streaming Provider.

For more information, see the article Custom Data Service Providers and the Open Data Protocol (OData) Provider Toolkit in the OData SDK.

See Also

Data Services Providers
Entity Framework Provider
Reflection Provider

Streaming Provider (WCF Data Services)

8/31/2018 • 10 minutes to read • Edit Online

A data service can expose large object binary data. This binary data might represent video and audio streams, images, document files, or other types of binary media. When an entity in the data model includes one or more binary properties, the data service returns this binary data encoded as base-64 inside the entry in the response feed. Because loading and serializing large binary data in this manner can affect performance, the Open Data Protocol (OData) defines a mechanism for retrieving binary data independent of the entity to which it belongs. This is accomplished by separating the binary data from the entity into one or more data streams.

- Media resource - binary data that belongs to an entity, such as a video, audio, image or other type of media resource stream.
- Media link entry - an entity that has a reference to a related media resource stream.

With WCF Data Services, you define a binary resource stream by implementing a streaming data provider. The streaming provider implementation supplies the data service with the media resource stream associated with a specific entity as an Stream object. This implementation enables the data service to accept and return media resources over HTTP as binary data streams of a specified MIME type.

Configuring a data service to support the streaming of binary data requires the following steps:

1. Attribute one or more entities in the data model as a media link entry. These entities should not include the binary data to be streamed. Any binary properties of an entity are always returned in the entry as base-64 encoded binary.
2. Implement the T:System.Data.Services.Providers.IDataServiceStreamProvider interface.
3. Define a data service that implements the IServiceProvider interface. The data service uses the GetService implementation to access the streaming data provider implementation. This method returns the appropriate streaming provider implementation.
4. Enable large message streams in the Web application configuration.
5. Enable access to binary resources on the server or in a data source.

The examples in this topic are based on a sample streaming photo service, which is discussed in depth in the post Data Services Streaming Provider Series: Implementing a Streaming Provider (Part 1). The source code for this sample service is available on the Streaming Photo Data Service Sample page on MSDN Code Gallery.

Defining a Media Link Entry in the Data Model

The data source provider determines the way that an entity is defined as a media link entry in the data model.

Entity Framework Provider
To indicate that an entity is a media link entry, add the `HasStream` attribute to the entity type definition in the conceptual model, as in the following example:

```
<EntityType xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata"
            Name="PhotoInfo" m:HasStream="true">
  <Key>
    <PropertyRef Name="PhotoId" />
  </Key>
  <Property Name="PhotoId" Type="Int32" Nullable="false"
            annotation:StoreGeneratedPattern="Identity" />
  <Property Name="FileName" Type="String" Nullable="false" />
  <Property Name="FileSize" Type="Int32" Nullable="true" />
  <Property Name="DateTaken" Type="DateTime" Nullable="true" />
  <Property Name="TakenBy" Type="String" Nullable="true" />
  <Property Name="DateAdded" Type="DateTime" Nullable="false" />
  <Property Name="Exposure" Type="PhotoData.Exposure" Nullable="false" />
  <Property Name="Dimensions" Type="PhotoData.Dimensions" Nullable="false" />
  <Property Name="DateModified" Type="DateTime" Nullable="false" />
  <Property Name="Comments" Type="String" MaxLength="Max"
            FixedLength="false" Unicode="true" />
  <Property Name="ContentType" Type="String" MaxLength="50" FixedLength="false" Unicode="true" />
</EntityType>
```

You must also add the namespace `xmlns:m=http://schemas.microsoft.com/ado/2007/08/dataservices/metadata` either to the entity or to the root of the .edmx or .csdl file that defines the data model.

For an example of a data service that uses the Entity Framework provider and exposes a media resource, see the post Data Services Streaming Provider Series: Implementing a Streaming Provider (Part 1).

Reflection Provider
To indicate that an entity is a media link entry, add the HasStreamAttribute to the class that defines the entity type in the reflection provider.

Custom Data Service Provider
When using custom service providers, you implement the IDataServiceMetadataProvider interface to define the metadata for your data service. For more information, see Custom Data Service Providers. You indicate that a binary resource stream belongs to a ResourceType by setting the IsMediaLinkEntry property to `true` on the ResourceType that represents the entity type, which is a media link entry.

Implementing the IDataServiceStreamProvider Interface

To create a data service that supports binary data streams, you must implement the IDataServiceStreamProvider interface. This implementation enables the data service to return binary data as a stream to the client and consume binary data as a stream sent from the client. The data service creates an instance of this interface whenever it needs to access binary data as a stream. The IDataServiceStreamProvider interface specifies the following members:

MEMBER NAME	DESCRIPTION
DeleteStream	This method is invoked by the data service to delete the corresponding media resource when its media link entry is deleted. When you implement IDataServiceStreamProvider, this method contains the code that deletes the media resource associated with the supplied media link entry.
GetReadStream	This method is invoked by the data service to return a media resource as a stream. When you implement IDataServiceStreamProvider, this method contains the code that provides a stream that is used by the data service to the return media resource that is associated with the provided media link entry.

MEMBER NAME	DESCRIPTION
GetReadStreamUri	This method is invoked by the data service to return the URI that is used to request the media resource for the media link entry. This value is used to create the `src` attribute in the content element of the media link entry and that is used to request the data stream. When this method returns `null`, the data service automatically determines the URI. Use this method when you need to provide clients with direct access to binary data without using the steam provider.
GetStreamContentType	This method is invoked by the data service to return the Content-Type value of the media resource that is associated with the specified media link entry.
GetStreamETag	This method is invoked by the data service to return the eTag of the data stream that is associated with the specified entity. This method is used when you manage concurrency for the binary data. When this method returns null, the data service does not track concurrency.
GetWriteStream	This method is invoked by the data service to obtain the stream that is used when receiving the stream sent from the client. When you implement IDataServiceStreamProvider, you must return a writable stream to which the data service writes received stream data.
ResolveType	Returns a namespace-qualified type name that represents the type that the data service runtime must create for the media link entry that is associated with the data stream for the media resource that is being inserted.

Creating the Streaming Data Service

To provide the WCF Data Services runtime with access to the IDataServiceStreamProvider implementation, the data service that you create must also implement the IServiceProvider interface. The following example shows how to implement the GetService method to return an instance of the `PhotoServiceStreamProvider` class that implements IDataServiceStreamProvider.

```
public partial class PhotoData : DataService<PhotoDataContainer>, IServiceProvider
{
    // This method is called only once to initialize service-wide policies.
    public static void InitializeService(DataServiceConfiguration config)
    {
        config.SetEntitySetAccessRule("PhotoInfo",
            EntitySetRights.ReadMultiple |
            EntitySetRights.ReadSingle |
            EntitySetRights.AllWrite);

        config.DataServiceBehavior.MaxProtocolVersion = DataServiceProtocolVersion.V2;
    }
    public object GetService(Type serviceType)
    {
        if (serviceType == typeof(IDataServiceStreamProvider))
        {
            // Return the stream provider to the data service.
            return new PhotoServiceStreamProvider(this.CurrentDataSource);
        }

        return null;
    }
}
```

```
Partial Public Class PhotoData
    Inherits DataService(Of PhotoDataContainer)
    Implements IServiceProvider

    ' This method is called only once to initialize service-wide policies.
    Public Shared Sub InitializeService(ByVal config As DataServiceConfiguration)
        config.SetEntitySetAccessRule("PhotoInfo", _
            EntitySetRights.ReadMultiple Or _
            EntitySetRights.ReadSingle Or _
            EntitySetRights.AllWrite)

        ' Named streams require version 3 of the OData protocol.
        config.DataServiceBehavior.MaxProtocolVersion = DataServiceProtocolVersion.V3
    End Sub
#Region "IServiceProvider Members"
    Public Function GetService(ByVal serviceType As Type) As Object _
    Implements IServiceProvider.GetService
        If serviceType Is GetType(IDataServiceStreamProvider) _
            Or serviceType Is GetType(IDataServiceStreamProvider) Then
            Return New PhotoServiceStreamProvider(Me.CurrentDataSource)
        End If
        Return Nothing
    End Function
#End Region
End Class
```

For general information about how to create a data service, see Configuring the Data Service.

Enabling Large Binary Streams in the Hosting Environment

When you create a data service in an ASP.NET Web application, Windows Communication Foundation (WCF) is used to provide the HTTP protocol implementation. By default, WCF limits the size of HTTP messages to only 65K bytes. To be able to stream large binary data to and from the data service, you must also configure the Web application to enable large binary files and to use streams for transfer. To do this, add the following in the `<configuration />` element of the application's Web.config file:

NOTE

You must use a TransferMode.Streamed transfer mode to ensure that the binary data in both the request and response messages are streamed and not buffered by WCF.

For more information, see Streaming Message Transfer and Transport Quotas.

By default, Internet Information Services (IIS) also limits the size of requests to 4MB. To enable your data service to receive streams larger than 4MB when running on IIS, you must also set the `maxRequestLength` attribute of the httpRuntime Element (ASP.NET Settings Schema) in the `<system.web />` configuration section, as shown in the following example:

Using Data Streams in a Client Application

The WCF Data Services client library enables you to both retrieve and update these exposed resources as binary streams on the client. For more information, see Working with Binary Data.

Considerations for Working with a Streaming Provider

The following are things to consider when you implement a streaming provider and when you access media resources from a data service.

- MERGE requests are not supported for media resources. Use a PUT request to change the media resource of an existing entity.
- A POST request cannot be used to create a new media link entry. Instead, you must issue a POST request to create a new media resource, and the data service creates a new media link entry with default values. This new entity can be updated by a subsequent MERGE or PUT request. You may also consider caching the entity and make updates in the disposer, such as setting the property value to the value of the Slug header in the POST request.
- When a POST request is received, the data service calls GetWriteStream to create the media resource before it calls SaveChanges to create the media link entry.
- An implementation of GetWriteStream should not return a MemoryStream object. When you use this kind of stream, memory resource issues will occur when the service receives very large data streams.
- The following are things to consider when storing media resources in a database:
  - A binary property that is a media resource should not be included in the data model. All properties exposed in a data model are returned in the entry in a response feed.
  - To improve performance with a large binary stream, we recommend that you create a custom stream class to store binary data in the database. This class is returned by your GetWriteStream implementation and sends the binary data to the database in chunks. For a SQL Server database, we recommend that you use a FILESTREAM to stream data into the database when the binary data is larger than 1MB.
  - Ensure that your database is designed to store the binary large streams that are to be received by your data service.
  - When a client sends a POST request to insert a media link entry with a media resource in a single request, GetWriteStream is called to obtain the stream before the data service inserts the new entity into the database. A streaming provider implementation must be able to handle this data service behavior. Consider using a separate data table to store the binary data or store the data stream in a file until after the entity has been inserted into the database.

- When you implement the DeleteStream, GetReadStream, or GetWriteStream methods, you must use the eTag and Content-Type values that are supplied as method parameters. Do not set eTag or Content-Type headers in your IDataServiceStreamProvider provider implementation.
- By default, the client sends large binary streams by using a chunked HTTP Transfer-Encoding. Because the ASP.NET Development Server does not support this kind of encoding, you cannot use this Web server to host a streaming data service that must accept large binary streams. For more information on ASP.NET Development Server, see Web Servers in Visual Studio for ASP.NET Web Projects.

Versioning Requirements

The streaming provider has the following OData protocol versioning requirements:

- The streaming provider requires that the data service support version 2.0 of the OData protocol and later versions.

For more information, see Data Service Versioning.

See Also

Data Services Providers
Custom Data Service Providers
Working with Binary Data

Service Operations (WCF Data Services)

5/2/2018 • 5 minutes to read • Edit Online

WCF Data Services enables you to define service operations on a data service to expose methods on the server. Like other data service resources, service operations are addressed by URIs. Service operations enable you to expose business logic in a data service, such as to implement validation logic, to apply role-based security, or to expose specialized querying capabilities. Service operations are methods added to the data service class that derives from DataService<T>. Like all other data service resources, you can supply parameters to the service operation method. For example, the following service operation URI (based on the quickstart data service) passes the value `London` to the `city` parameter:

```
http://localhost:12345/Northwind.svc/GetOrdersByCity?city='London'
```

The definition for this service operation is as follows:

```
[WebGet]
public IQueryable<Order> GetOrdersByCity(string city)
```

```
<WebGet()> _
Public Function GetOrdersByCity(ByVal city As String) As IQueryable(Of Order)
```

You can use the CurrentDataSource of the DataService<T> to directly access the data source that the data service is using. For more information, see How to: Define a Service Operation.

For information on how to call a service operation from a .NET Framework client application, see Calling Service Operations.

Service Operation Requirements

The following requirements apply when defining service operations on the data service. If a method does not meet these requirements, it will not be exposed as a service operation for the data service.

- The operation must be a public instance method that is a member of the data service class.
- The operation method may only accept input parameters. Data sent in the message body cannot be accessed by the data service.
- If parameters are defined, the type of each parameter must be a primitive type. Any data of a non-primitive type must be serialized and passed into a string parameter.
- The method must return one of the following:
  - `void` (`Nothing` in Visual Basic)
  - IEnumerable<T>
  - IQueryable<T>
  - An entity type in the data model that the data service exposes.
  - A primitive class such as integer or string.

- In order to support query options such as sorting, paging, and filtering, service operation methods should return IQueryable<T>. Requests to service operations that include query options are rejected for operations that only return IEnumerable<T>.
- In order to support accessing related entities by using navigation properties, the service operation must return IQueryable<T>.
- The method must be annotated with the `[WebGet]` or `[WebInvoke]` attribute.
  - `[WebGet]` enables the method to be invoked by using a GET request.
  - `[WebInvoke(Method = "POST")]` enables the method to be invoked by using a POST request. Other WebInvokeAttribute methods are not supported.
- A service operation may be annotated with the SingleResultAttribute that specifies that the return value from the method is a single entity rather than a collection of entities. This distinction dictates the resulting serialization of the response and the manner in which additional navigation property traversals are represented in the URI. For example, when using AtomPub serialization, a single resource type instance is represented as an entry element and a set of instances as a feed element.

Addressing Service Operations

You can address service operations by placing the name of the method in the first path segment of a URI. As an example, the following URI accesses a `GetOrdersByState` operation that returns an IQueryable<T> collection of `Orders` objects.

```
http://localhost:12345/Northwind.svc/GetOrdersByState?state='CA'&includeItems=true
```

When calling a service operation, parameters are supplied as query options. The previous service operation accepts both a string parameter `state` and a Boolean parameter `includeItems` that indicates whether to include related `Order_Detail` objects in the response.

The following are valid return types for a service operation:

VALID RETURN TYPES	URI RULES
`void` (`Nothing` in Visual Basic) -or- Entity types -or- Primitive types	The URI must be a single path segment that is the name of the service operation. Query options are not allowed.
IEnumerable<T>	The URI must be a single path segment that is the name of the service operation. Because the result type is not an IQueryable<T> type, query options are not allowed.
IQueryable<T>	Query path segments in addition to the path that is the name of the service operation are allowed. Query options are also allowed.

Additional path segments or query options may be added to the URI depending on the return type of the service operation. For example, the following URI accesses a `GetOrdersByCity` operation that returns an IQueryable<T> collection of `Orders` objects, ordered by `RequiredDate` in descending order, along with the related `Order_Details` objects:

```
http://localhost:12345/Northwind.svc/GetOrdersByCity?
city='London'&$expand=Order_Details&$orderby=RequiredDate desc
```

Service Operations Access Control

Service-wide visibility of service operations is controlled by the SetServiceOperationAccessRule method on the IDataServiceConfiguration class in much the same way that entity set visibility is controlled by using the SetEntitySetAccessRule method. For example, the following line of code in the data service definition enables access to the `CustomersByCity` service operation.

```
config.SetServiceOperationAccessRule(
    "GetOrdersByCity", ServiceOperationRights.AllRead);
```

```
config.SetServiceOperationAccessRule( _
    "GetOrdersByCity", ServiceOperationRights.AllRead)
```

NOTE

If a service operation has a return type that has been hidden by restricting access on the underlying entity sets, then the service operation will not be available to client applications.

For more information, see How to: Define a Service Operation.

Raising Exceptions

We recommend that you use the DataServiceException class whenever you raise an exception in the data service execution. This is because the data service runtime knows how to map properties of this exception object correctly to the HTTP response message. When you raise a DataServiceException in a service operation, the returned exception is wrapped in a TargetInvocationException. To return the base DataServiceException without the enclosing TargetInvocationException, you must override the HandleException method in the DataService<T>, extract the DataServiceException from the TargetInvocationException, and return it as the top-level error, as in the following example:

```
// Override to manage returned exceptions.
protected override void HandleException(HandleExceptionArgs args)
{
    // Handle exceptions raised in service operations.
    if (args.Exception.GetType() ==
        typeof(TargetInvocationException)
        && args.Exception.InnerException != null)
    {
        if (args.Exception.InnerException.GetType()
            == typeof(DataServiceException))
        {

            // Unpack the DataServiceException.
            args.Exception = args.Exception.InnerException as DataServiceException;

        }
        else
        {
            // Return a new DataServiceException as "400: bad request."
            args.Exception =
                new DataServiceException(400,
                    args.Exception.InnerException.Message);
        }
    }
}
```

```
' Override to manage returned exceptions.
Protected Overrides Sub HandleException(args As HandleExceptionArgs)
    ' Handle exceptions raised in service operations.
    If args.Exception.GetType() = GetType(TargetInvocationException) _
        AndAlso args.Exception.InnerException IsNot Nothing Then
        If args.Exception.InnerException.GetType() = GetType(DataServiceException) Then
            ' Unpack the DataServiceException.
            args.Exception = _
                TryCast(args.Exception.InnerException, DataServiceException)
        Else
            ' Return a new DataServiceException as "400: bad request."
            args.Exception = _
                New DataServiceException(400, args.Exception.InnerException.Message)
        End If
    End If
End Sub
```

See Also

Interceptors

How to: Define a Service Operation (WCF Data Services)

5/2/2018 • 2 minutes to read • Edit Online

WCF Data Services expose methods that are defined on the server as service operations. Service operations allow a data service to provide access through a URI to a method that is defined on the server. To define a service operation, apply the [`WebGet]` or `[WebInvoke]` attribute to the method. To support query operators, the service operation must return an IQueryable<T> instance. Service operations may access the underlying data source through the CurrentDataSource property on the DataService<T>. For more information, see Service Operations.

The example in this topic defines a service operation named `GetOrdersByCity` that returns a filtered IQueryable<T> instance of `Orders` and related `Order_Details` objects. The example accesses the ObjectContext instance that is the data source for the Northwind sample data service. This service is created when you complete the WCF Data Services quickstart.

To define a service operation in the Northwind data service

1. In the Northwind data service project, open the Northwind.svc file.

2. In the `Northwind` class, define a service operation method named `GetOrdersByCity` as follows:

   ```
   [WebGet]
   public IQueryable<Order> GetOrdersByCity(string city)
   ```

   ```
   <WebGet()> _
   Public Function GetOrdersByCity(ByVal city As String) As IQueryable(Of Order)
   ```

3. In the `InitializeService` method of the `Northwind` class, add the following code to enable access to the service operation:

   ```
   config.SetServiceOperationAccessRule(
       "GetOrdersByCity", ServiceOperationRights.AllRead);
   ```

   ```
   config.SetServiceOperationAccessRule( _
       "GetOrdersByCity", ServiceOperationRights.AllRead)
   ```

To query the GetOrdersByCity service operation

- In a Web browser, enter one of the following URIs to invoke the service operation that is defined in the following example:
  - `http://localhost:12345/Northwind.svc/GetOrdersByCity?city='London'`
  - `http://localhost:12345/Northwind.svc/GetOrdersByCity?city='London'&$top=2`
  - `http://localhost:12345/Northwind.svc/GetOrdersByCity?city='London'&$expand=Order_Details&$orderby=RequiredDate desc`

Example

The following example implements a service operation named `GetOrderByCity` on the Northwind data service. This operation uses the ADO.NET Entity Framework to return a set of `Orders` and related `Order_Details` objects as an IQueryable<T> instance based on the provided city name.

NOTE

Query operators are supported on this service operation endpoint because the method returns an IQueryable<T> instance.

```
[WebGet]
public IQueryable<Order> GetOrdersByCity(string city)
{
    if (string.IsNullOrEmpty(city))
    {
        throw new ArgumentNullException("city",
            "You must provide a value for the parameter'city'.");
    }

    // Get the ObjectContext that is the data source for the service.
    NorthwindEntities context = this.CurrentDataSource;

    try
    {

        var selectedOrders = from order in context.Orders.Include("Order_Details")
                             where order.Customer.City == city
                             select order;

        return selectedOrders;
    }
    catch (Exception ex)
    {
        throw new ApplicationException(string.Format(
            "An error occurred: {0}", ex.Message));
    }
}
```

```
<WebGet()> _
Public Function GetOrdersByCity(ByVal city As String) As IQueryable(Of Order)
    If String.IsNullOrEmpty(city) Then
        Throw New ArgumentNullException("city", _
            "You must provide a value for the parameter'city'.")
    End If

    ' Get the ObjectContext that is the data source for the service.
    Dim context As NorthwindEntities = Me.CurrentDataSource

    Try
        Dim selectedOrders = From order In context.Orders.Include("Order_Details") _
                                 Where order.Customer.City = city _
                                 Select order
        Return selectedOrders
    Catch ex As Exception
        Throw New ApplicationException("An error occurred: {0}", ex)
    End Try
End Function
```

See Also

Defining WCF Data Services

Feed Customization (WCF Data Services)

8/31/2018 • 8 minutes to read • Edit Online

WCF Data Services uses the Open Data Protocol (OData) to expose data as a feed. OData supports both Atom and JavaScript Object Notation (JSON) formats for data feeds. When you use an Atom feed, OData provides a standard method to serialize data, such as entities and relationships, into an XML format that can be included in the body of HTTP message. OData defines a default entity-property mapping between the data that is contained in entities and Atom elements. For more information, see OData: Atom Format.

You may have an application scenario that requires that the property data returned by the data service be serialized in a customized manner rather than in the standard feed format. With OData, you can customize the serialization in a data feed so that properties of an entity may be mapped to unused elements and attributes of an entry or to custom elements of an entry in the feed.

NOTE

Feed customization is only supported for Atom feeds. Custom feeds are not returned when the JSON format is requested for the returned feed.

With WCF Data Services, you can define an alternate entity-property mapping for an Atom payload by manually applying attributes to entity types in the data model. The data source provider of the data service determines how you should apply these attributes.

IMPORTANT

When you define custom feeds, you must guarantee that all entity properties that have custom mappings defined are included in the projection. When a mapped entity property is not included in the projection, data loss might occur. For more information, see Query Projections.

Customizing Feeds with the Entity Framework Provider

The data model used with the Entity Framework provider is represented as XML in the .edmx file. In this case, the attributes that define custom feeds are added to the `EntityType` and `Property` elements that represent entity types and properties in the data model. These feed customization attributes are not defined in [MC-CSDL]: Conceptual Schema Definition File Format, which is the format that the Entity Framework provider uses to define the data model. Therefore, you must declare feed customization attributes in a specific schema namespace, which is defined as `m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata"`. The following XML fragment shows feed customization attributes applied to `Property` elements of the `Products` entity type that define the `ProductName`, `ReorderLevel`, and `UnitsInStock` properties.

```
<Property Name="ProductName" Type="String" Nullable="false"
          MaxLength="40" Unicode="true" FixedLength="false"
          m:FC_TargetPath="SyndicationAuthorName"
          m:FC_ContentKind="text"
          m:FC_KeepInContent="true"
          />
<Property Name="UnitsInStock" Type="Int16"
          m:FC_TargetPath="UnitsInStock"
          m:FC_NsPrefix="Northwind"
          m:FC_NsUri="http://schemas.examples.microsoft.com/dataservices"
          m:FC_KeepInContent="true"
          />
<Property Name="ReorderLevel" Type="Int16"
          m:FC_TargetPath="UnitsInStock/@ReorderLevel"
          m:FC_NsPrefix="Northwind"
          m:FC_NsUri="http://schemas.examples.microsoft.com/dataservices"
          m:FC_KeepInContent="false"
          />
```

These attributes produce the following customized data feed for the `Products` entity set. In the customized data feed, the `ProductName` property value is displayed in both in the `author` element and as the `ProductName` property element, and the `UnitsInStock` property is displayed in a custom element that has its own unique namespace and with the `ReorderLevel` property as an attribute:

```
<entry xml:base="http://localhost:12345/Northwind.svc/"
         xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices"
         xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata"
         xmlns="http://www.w3.org/2005/Atom">
  <id>http://localhost:12345/Northwind.svc/Products(1)</id>
  <title type="text" />
  <updated>2009-10-02T05:09:44Z</updated>
  <author>
    <name>Chai</name>
  </author>
  <link rel="edit" title="Products" href="Products(1)" />
  <link rel="http://schemas.microsoft.com/ado/2007/08/dataservices/related/Order_Details"
        type="application/atom+xml;type=feed" title="Order_Details"
        href="Products(1)/Order_Details" />
  <category term="NorthwindModel.Products"
            scheme="http://schemas.microsoft.com/ado/2007/08/dataservices/scheme" />
  <content type="application/xml">
    <m:properties>
      <d:ProductID m:type="Edm.Int32">1</d:ProductID>
      <d:ProductName>Chai</d:ProductName>
      <d:UnitsInStock m:type="Edm.Int16">39</d:UnitsInStock>
      <d:SupplierID m:type="Edm.Int32">1</d:SupplierID>
      <d:CategoryID m:type="Edm.Int32">1</d:CategoryID>
      <d:QuantityPerUnit>10 boxes x 20 bags</d:QuantityPerUnit>
      <d:UnitPrice m:type="Edm.Decimal">18.0000</d:UnitPrice>
      <d:UnitsOnOrder m:type="Edm.Int16">0</d:UnitsOnOrder>
      <d:Discontinued m:type="Edm.Boolean">false</d:Discontinued>
    </m:properties>
  </content>
  <Northwind:UnitsInStock
    Northwind:ReorderLevel="10"
    xmlns:Northwind="http://schemas.examples.microsoft.com/dataservices">39</Northwind:UnitsInStock>
</entry>
```

For more information, see How to: Customize Feeds with the Entity Framework Provider.

NOTE

Because extensions to the data model are not supported by the Entity Designer, you must manually modify the XML file that contains the data model. For more information about the .edmx file that is generated by the Entity Data Model tools, see .edmx File Overview.

Custom Feed Attributes

The following table shows the XML attributes that customize feeds that you can add to the conceptual schema definition language (CSDL) that defines the data model. These attributes are equivalent to the properties of the EntityPropertyMappingAttribute used with the reflection provider.

ATTRIBUTE NAME	DESCRIPTION
`FC_ContentKind`	Indicates the type of the content. The following keywords define syndication content types. `text:` The property value is displayed in the feed as text. `html:` The property value is displayed in the feed as HTML. `xhtml:` The property value is displayed in the feed as XML-formatted HTML. These keywords are equivalent to the values of the SyndicationTextContentKind enumeration used with the reflection provider. This attribute is not supported when the `FC_NsPrefix` and `FC_NsUri` attributes are used. When you specify a value of `xhtml` for the `FC_ContentKind` attribute, you must ensure that the property value contains properly formatted XML. The data service returns the value without performing any transformations. You must also ensure that any XML element prefixes in the returned XML have a namespace URI and prefix defined in the mapped feed.
`FC_KeepInContent`	Indicates that the referenced property value should be included both in the content section of the feed and in the mapped location. Valid values are `true` and `false`. To make the resulting feed backward-compatible with earlier versions of WCF Data Services, specify a value of `true` to make sure that the value is included in the content section of the feed.
`FC_NsPrefix`	The namespace prefix of the XML element in a non-syndication mapping. This attribute must be used with the `FC_NsUri` attribute and cannot be used with the `FC_ContentKind` attribute.
`FC_NsUri`	The namespace URI of the XML element in a non-syndication mapping. This attribute must be used with the `FC_NsPrefix` attribute and cannot be used with the `FC_ContentKind` attribute.

ATTRIBUTE NAME	DESCRIPTION

`FC_SourcePath`	The path of the property of the entity to which this feed mapping rule applies. This attribute is only supported when it is used in an `EntityType` element.
	
	The SourcePath property cannot directly reference a complex type. For complex types, you must use a path expression where property names are separated by a backslash (`/`) character. For example, the following values are allowed for an entity type `Person` with an integer property `Age` and a complex property
	
	`Address` :
	
	`Age`
	
	`Address/Street`
	
	The SourcePath property cannot be set to a value that contains a space or any other character that is not valid in a property name.

ATTRIBUTE NAME	DESCRIPTION
`FC_TargetPath`	The name of the target element of the resulting feed to map the property. This element can be an element defined by the Atom specification or a custom element.
	The following keywords are predefined syndication target-path values that point to specific location in an OData feed.
	`SyndicationAuthorEmail:` The `atom:email` child element of the `atom:author` element.
	`SyndicationAuthorName:` The `atom:name` child element of the `atom:author` element.
	`SyndicationAuthorUri:` The `atom:uri` child element of the `atom:author` element.
	`SyndicationContributorEmail:` The `atom:email` child element of the `atom:contributor` element.
	`SyndicationContributorName:` The `atom:name` child element of the `atom:contributor` element.
	`SyndicationContributorUri:` The `atom:uri` child element of the `atom:contributor` element.
	`SyndicationCustomProperty:` A custom property element. When mapping to a custom element, the target must be a path expression in which nested elements are separated by a backslash (`/`) and attributes are specified by an ampersand (`@`). In the following example, the string `UnitsInStock/@ReorderLevel` maps a property value to an attribute named `ReorderLevel` on a child element named `UnitsInStock` of the root entry element.
	`<Property Name="ReorderLevel" Type="Int16"`
	`m:FC_TargetPath="UnitsInStock/@ReorderLevel"`
	`m:FC_NsPrefix="Northwind"`
	`m:FC_NsUri="http://schemas.examples.microsoft.com/dataservices`
	`m:FC_KeepInContent="false" />`
	When the target is a custom element name, the `FC_NsPrefix` and `FC_NsUri` attributes must also be specified.
	`SyndicationPublished:` The `atom:published` element.
	`SyndicationRights:` The `atom:rights` element.
	`SyndicationSummary:` The `atom:summary` element.
	`SyndicationTitle:` The `atom:title` element.
	`SyndicationUpdated:` The `atom:updated` element.
	These keywords are equivalent to the values of the SyndicationItemProperty enumeration used with the reflection provider.

NOTE

Attribute names and values are case-sensitive. Attributes can be applied either to the `EntityType` element or to one or more `Property` elements, but not to both.

Customizing Feeds with the Reflection Provider

To customize feeds for a data model that was implemented by using the reflection provider, add one or more instances of the EntityPropertyMappingAttribute attribute to the classes that represent entity types in the data model. The properties of the EntityPropertyMappingAttribute class correspond to the feed customization attributes that are described in the previous section. The following is an example of the declaration of the `Order` type, with custom feed mapping defined for both properties.

NOTE

The data model for this example is defined in the topic How to: Create a Data Service Using the Reflection Provider.

```
[EntityPropertyMappingAttribute("Customer",
    SyndicationItemProperty.AuthorName,
    SyndicationTextContentKind.Plaintext, true)]
[EntityPropertyMapping("OrderId",
    SyndicationItemProperty.Title,
    SyndicationTextContentKind.Plaintext, false)]
[DataServiceKeyAttribute("OrderId")]
public class Order
```

```
<EntityPropertyMappingAttribute("Customer", _
    SyndicationItemProperty.AuthorName, _
    SyndicationTextContentKind.Plaintext, True)> _
<EntityPropertyMapping("OrderId", _
    SyndicationItemProperty.Title, _
    SyndicationTextContentKind.Plaintext, False)> _
<DataServiceKeyAttribute("OrderId")> _
Public Class Order
```

These attributes produce the following customized data feed for the `Orders` entity set. In this customized feed, the `OrderId` property value displays only in the `title` element of the `entry` and the `Customer` property value displays both in the `author` element and as the `Customer` property element:

```
<entry xml:base="http://localhost:12345/OrderItems.svc/"
       xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices"
       xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata"
       xmlns="http://www.w3.org/2005/Atom">
  <id>http://localhost:12345/OrderItems.svc/Orders(0)</id>
  <title type="text">0</title>
  <updated>2009-07-25T21:11:11Z</updated>
  <author>
    <name>Peter Franken</name>
  </author>
  <link rel="edit" title="Order" href="Orders(0)" />
  <link rel="http://schemas.microsoft.com/ado/2007/08/dataservices/related/Items"
        type="application/atom+xml;type=feed" title="Items" href="Orders(0)/Items" />
  <category term="CustomDataService.Order"
            scheme="http://schemas.microsoft.com/ado/2007/08/dataservices/scheme" />
  <content type="application/xml">
    <m:properties>
      <d:Customer>Peter Franken</d:Customer>
    </m:properties>
  </content>
</entry>
```

For more information, see How to: Customize Feeds with the Reflection Provider.

Customizing Feeds with a Custom Data Service Provider

Feed customization for a data model defined by using a custom data service provider is defined for a resource type by calling the AddEntityPropertyMappingAttribute on the ResourceType that represents an entity type in the data model. For more information, see Custom Data Service Providers.

Consuming Custom Feeds

When your application directly consumes an OData feed, it must be able to process any customized elements and attributes in the returned feed. When you have implemented custom feeds in your data model, regardless of the data service provider, the `$metadata` endpoint returns custom feed information as custom feed attributes in the CSDL returned by the data service. When you use the **Add Service Reference** dialog or the datasvcutil.exe tool to generate client data service classes, the customized feed attributes are used to guarantee that requests and responses to the data service are handled correctly.

Feed Customization Considerations

You should consider the following when defining custom feed mappings.

- The WCF Data Services client treats mapped elements in a feed as empty when they contain only white space. Because of this, mapped elements that contain only white space are not materialized on the client with the same white space. To preserve this white space on the client, you must set the value of `KeepInContext` to `true` in the feed mapping attribute.

Versioning Requirements

Feed customization has the following OData protocol versioning requirements:

- Feed customization requires that both the client and data service support version 2.0 of the OData protocol and later versions.

For more information, see Data Service Versioning.

See Also

Reflection Provider

Entity Framework Provider

How to: Customize Feeds with the Entity Framework Provider (WCF Data Services)

8/31/2018 • 2 minutes to read • Edit Online

WCF Data Services enables you to customize the Atom serialization in a data service response so that properties of an entity may be mapped to unused elements that are defined in the AtomPub protocol. This topic shows how to define mapping attributes for the entity types in a data model that is defined in an .edmx file by using the Entity Framework provider. For more information, see Feed Customization.

In this topic you will manually modify the tool-generated .edmx file that contains the data model. You must manually modify the file because extensions to the data model are not supported by the Entity Designer. For more information about the .edmx file that the Entity Data Model tools generate, see .edmx File Overview. The example in this topic uses the Northwind sample data service and autogenerated client data service classes. This service and the client data classes are created when you complete the WCF Data Services quickstart.

To manually modify the Northwind.edmx file to add feed customization attributes

1. In **Solution Explorer**, right-click the `Northwind.edmx` file, and then click **Open with**.
2. In the **Open With - Northwind.edmx** dialog box, select **XML Editor**, and then click **OK**.
3. Locate the `ConceptualModels` element and replace the existing `Customers` entity type with the following element that contains feed customization mapping attributes:

```
<EntityType Name="Customers"
            m:FC_SourcePath="CustomerID"
            m:FC_TargetPath="SyndicationTitle"
            m:FC_ContentKind="text"
            m:FC_KeepInContent="false"
            >
   <Key>
    <PropertyRef Name="CustomerID" />
  </Key>
  <Property Name="CustomerID" Type="String" Nullable="false"
           MaxLength="5" Unicode="true" FixedLength="true" />
  <Property Name="ContactName" Type="String" MaxLength="30"
            Unicode="true" FixedLength="false"
            m:FC_TargetPath="SyndicationAuthorName"
            m:FC_ContentKind="text"
            m:FC_KeepInContent="true"
            />
  <Property Name="CompanyName" Type="String" Nullable="false"
            MaxLength="40" Unicode="true" FixedLength="false"
            m:FC_TargetPath="CompanyName"
            m:FC_NsPrefix="Northwind"
            m:FC_NsUri="http://schemas.examples.microsoft.com/dataservices"
            m:FC_KeepInContent="true"
            />
  <Property Name="ContactTitle" Type="String" MaxLength="30"
            Unicode="true" FixedLength="false" />
  <Property Name="Address" Type="String" MaxLength="60"
            Unicode="true" FixedLength="false" />
  <Property Name="City" Type="String" MaxLength="15"
            Unicode="true" FixedLength="false" />
  <Property Name="Region" Type="String" MaxLength="15"
            Unicode="true" FixedLength="false" />
  <Property Name="PostalCode" Type="String" MaxLength="10"
            Unicode="true" FixedLength="false" />
  <Property Name="Country" Type="String" MaxLength="15"
            Unicode="true" FixedLength="false" />
  <Property Name="Phone" Type="String" MaxLength="24"
            Unicode="true" FixedLength="false" />
  <Property Name="Fax" Type="String" MaxLength="24"
            Unicode="true" FixedLength="false" />
  <NavigationProperty Name="Orders"
                      Relationship="NorthwindModel.FK_Orders_Customers"
                      FromRole="Customers" ToRole="Orders" />
</EntityType>
```

4. Save changes and close the Northwind.edmx file.

5. (Optional) Right-click the Northwind.edmx file and then click **Run Custom Tool**.

   This regenerates the object layer file, which may be required.

6. Recompile the project.

Example

The previous example returns the following result for the URI `http://myservice/``Northwind.svc/Customers('ALFKI')`.

```
<entry xml:base="http://localhost:12345/Northwind.svc/"
       xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices"
       xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata"
       xmlns="http://www.w3.org/2005/Atom">
  <id>http://localhost:12345/Northwind.svc/Customers('ALFKI')</id>
  <title type="text">ALFKI</title>
  <updated>2009-07-27T07:59:43Z</updated>
  <author>
    <name>Peter Franken</name>
  </author>
  <link rel="edit" title="Customers" href="Customers('ALFKI')" />
  <link rel="http://schemas.microsoft.com/ado/2007/08/dataservices/related/Orders"
        type="application/atom+xml;type=feed" title="Orders"
        href="Customers('ALFKI')/Orders" />
  <category term="NorthwindModel.Customers"
            scheme="http://schemas.microsoft.com/ado/2007/08/dataservices/scheme" />
  <content type="application/xml">
    <m:properties>
      <d:ContactName>Peter Franken</d:ContactName>
      <d:CompanyName>Alfreds Futterkiste</d:CompanyName>
      <d:ContactTitle>Marketing Manager</d:ContactTitle>
      <d:Address>Obere Str. 57</d:Address>
      <d:City>Berlin</d:City>
      <d:Region m:null="true" />
      <d:PostalCode>12209</d:PostalCode>
      <d:Country>Germany</d:Country>
      <d:Phone>089-0877310</d:Phone>
      <d:Fax>089-0877554</d:Fax>
    </m:properties>
  </content>
  <Northwind:CompanyName
    xmlns:Northwind="http://schemas.examples.microsoft.com/dataservices">Alfreds
Futterkiste</Northwind:CompanyName>
</entry>
```

See Also

Entity Framework Provider

How to: Customize Feeds with the Reflection Provider (WCF Data Services)

5/2/2018 • 4 minutes to read • Edit Online

WCF Data Services enables you to customize the Atom serialization in a data service response so that properties of an entity may be mapped to unused elements that are defined in the AtomPub protocol. This topic shows how to define mapping attributes for the entity types in a data model that is defined by using the reflection provider. For more information, see Feed Customization.

The data model for this example is defined in the topic How to: Create a Data Service Using the Reflection Provider

Example

In the following example, both properties of the `Order` type are mapped to existing Atom elements. The `Product` property of the `Item` type is mapped to a custom feed attribute in a separate namespace.

```
using System;
using System.Collections.Generic;
using System.Data.Services;
using System.Data.Services.Common;
using System.Linq;

namespace CustomDataService
{
    [EntityPropertyMappingAttribute("Customer",
        SyndicationItemProperty.AuthorName,
        SyndicationTextContentKind.Plaintext, true)]
    [EntityPropertyMapping("OrderId",
        SyndicationItemProperty.Title,
        SyndicationTextContentKind.Plaintext, false)]
    [DataServiceKeyAttribute("OrderId")]
    public class Order
    {
        public int OrderId { get; set; }
        public string Customer { get; set; }
        public IList<Item> Items { get; set; }
    }
    [EntityPropertyMappingAttribute("Product", "productname",
     "orders", "http://schema.examples.microsoft.com/dataservices", true)]
    [DataServiceKeyAttribute("Product")]
    public class Item
    {
        public string Product { get; set; }
        public int Quantity { get; set; }
    }
    public partial class OrderItemData
    {
        #region Populate Service Data
        static IList<Order> _orders;
        static IList<Item> _items;
        static OrderItemData()
        {
            _orders = new Order[]{
              new Order(){ OrderId=0, Customer = "Peter Franken", Items = new List<Item>()},
              new Order(){ OrderId=1, Customer = "Ana Trujillo", Items = new List<Item>()}};
            _items = new Item[]{
              new Item(){ Product="Chai", Quantity=10 },
              new Item(){ Product="Chang", Quantity=25 },
              new Item(){ Product="Aniseed Syrup", Quantity = 5 },
              new Item(){ Product="Chef Anton's Cajun Seasoning", Quantity=30}};
            _orders[0].Items.Add(_items[0]);
            _orders[0].Items.Add(_items[1]);
            _orders[1].Items.Add(_items[2]);
            _orders[1].Items.Add(_items[3]);
        }
        #endregion
        public IQueryable<Order> Orders
        {
            get { return _orders.AsQueryable(); }
        }
        public IQueryable<Item> Items
        {
            get { return _items.AsQueryable(); }
        }
    }

    public class OrderItems : DataService<OrderItemData>
    {
        // This method is called only once to initialize
        //service-wide policies.
        public static void InitializeService(DataServiceConfiguration
                                             config)
        {
            config.SetEntitySetAccessRule("Orders",
               EntitySetRights.AllRead |
               EntitySetRights.AllWrite);
            config.SetEntitySetAccessRule("Items",
                EntitySetRights.AllRead |
                EntitySetRights.AllWrite);
            config.DataServiceBehavior.MaxProtocolVersion =
                DataServiceProtocolVersion.V2;
        }
    }
}
```

```
Imports System
Imports System.Collections.Generic
Imports System.Data.Services
Imports System.Data.Services.Common
Imports System.Linq

Namespace CustomDataService
    <EntityPropertyMappingAttribute("Customer", _
        SyndicationItemProperty.AuthorName, _
        SyndicationTextContentKind.Plaintext, True)> _
    <EntityPropertyMapping("OrderId", _
        SyndicationItemProperty.Title, _
        SyndicationTextContentKind.Plaintext, False)> _
    <DataServiceKeyAttribute("OrderId")> _
    Public Class Order
        Private _orderId As Integer
        Private _customer As String
        Private _items As IList(Of Item)
        Public Property OrderId() As Integer
            Get
                Return _orderId
            End Get
            Set(ByVal value As Integer)
                _orderId = value
            End Set
        End Property
        Public Property Customer() As String
            Get
                Return _customer
            End Get
            Set(ByVal value As String)
            Set(ByVal value As String)
                _customer = value
            End Set
        End Property
        Public Property Items() As IList(Of Item)
            Get
                Return _items
            End Get
            Set(ByVal value As IList(Of Item))
                _items = value
            End Set
        End Property
    End Class
    <EntityPropertyMappingAttribute("Product", "productname", _
        "orders", "http://schema.examples.microsoft.com/dataservices", True)> _
    <DataServiceKeyAttribute("Product")> _
    Public Class Item
        Private _product As String
        Private _quantity As Integer
        Public Property Product() As String
            Get
                Return _product
            End Get
            Set(ByVal value As String)
                _product = value
            End Set
        End Property
        Public Property Quantity() As Integer
            Get
                Return _quantity
            End Get
            Set(ByVal value As Integer)
                _quantity = value
            End Set
        End Property
    End Class
    Partial Public Class OrderItemData
#Region "Populate Service Data"
        Shared _orders As IList(Of Order)
        Shared _items As IList(Of Item)
        Sub New()
            _orders = New Order() { _
                New Order() With {.OrderId = 0, .Customer = "Peter Franken", .Items = New List(Of Item)()}, _
              New Order() With {.OrderId = 1, .Customer = "Ana Trujillo", .Items = New List(Of Item)()}}
            _items = New Item() { _
              New Item() With {.Product = "Chai", .Quantity = 10}, _
              New Item() With {.Product = "Chang", .Quantity = 25}, _
              New Item() With {.Product = "Aniseed Syrup", .Quantity = 5}, _
              New Item() With {.Product = "Chef Anton's Cajun Seasoning", .Quantity = 30}}
            _orders(0).Items.Add(_items(0))
            _orders(0).Items.Add(_items(1))
            _orders(1).Items.Add(_items(2))
            _orders(1).Items.Add(_items(3))
        End Sub
#End Region
        Public ReadOnly Property Orders() As IQueryable(Of Order)
            Get
                Return _orders.AsQueryable()
            End Get
        End Property
        Public ReadOnly Property Items() As IQueryable(Of Item)
            Get
                Return _items.AsQueryable()
            End Get
        End Property
    End Class
    Public Class OrderItems
        Inherits DataService(Of OrderItemData)
        ' This method is called only once to initialize
        ' service-wide policies.
        Shared Sub InitializeService(ByVal config As DataServiceConfiguration)
            config.SetEntitySetAccessRule("Orders", _
                                          EntitySetRights.AllRead _
                                          Or EntitySetRights.AllWrite)
            config.SetEntitySetAccessRule("Items", _
                                          EntitySetRights.AllRead _
                                          Or EntitySetRights.AllWrite)
            config.DataServiceBehavior.MaxProtocolVersion =
                DataServiceProtocolVersion.V2
        End Sub
    End Class
End Namespace
```

Example

The previous example returns the following result for the URI `http://myservice/OrderItems.svc/Orders(0)?$expand=Items`.

```
<entry xml:base="http://localhost:12345/OrderItems.svc/"
       xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices"
       xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata"
       xmlns="http://www.w3.org/2005/Atom">
  <id>http://localhost:12345/OrderItems.svc/Orders(0)</id>
  <title type="text">0</title>
  <updated>2009-07-25T21:12:30Z</updated>
  <author>
    <name>Peter Franken</name>
  </author>
  <link rel="edit" title="Order" href="Orders(0)" />
  <link rel="http://schemas.microsoft.com/ado/2007/08/dataservices/related/Items"
        type="application/atom+xml;type=feed" title="Items" href="Orders(0)/Items">
    <m:inline>
      <feed>
        <title type="text">Items</title>
        <id>http://localhost:12345/OrderItems.svc/Orders(0)/Items</id>
        <updated>2009-07-25T21:12:30Z</updated>
        <link rel="self" title="Items" href="Orders(0)/Items" />
        <entry>
          <id>http://localhost:12345/OrderItems.svc/Items('Chai')</id>
          <title type="text" />
          <updated>2009-07-25T21:12:30Z</updated>
          <author>
            <name />
          </author>
          <link rel="edit" title="Item" href="Items('Chai')" />
          <category term="CustomDataService.Item"
                    scheme="http://schemas.microsoft.com/ado/2007/08/dataservices/scheme" />
          <content type="application/xml">
            <m:properties>
              <d:Product>Chai</d:Product>
              <d:Quantity m:type="Edm.Int32">10</d:Quantity>
            </m:properties>
          </content>
          <orders:productname
            xmlns:orders="http://schema.examples.microsoft.com/dataservices">Chai</orders:productname>
        </entry>
        <entry>
          <id>http://localhost:12345/OrderItems.svc/Items('Chang')</id>
          <title type="text" />
          <updated>2009-07-25T21:12:30Z</updated>
          <author>
            <name />
          </author>
          <link rel="edit" title="Item" href="Items('Chang')" />
          <category term="CustomDataService.Item"
```

```
                      scheme="http://schemas.microsoft.com/ado/2007/08/dataservices/scheme" />
            <content type="application/xml">
              <m:properties>
                <d:Product>Chang</d:Product>
                <d:Quantity m:type="Edm.Int32">25</d:Quantity>
              </m:properties>
            </content>
            <orders:productname
              xmlns:orders="http://schema.examples.microsoft.com/dataservices">Chang</orders:productname>
          </entry>
        </feed>
      </m:inline>
    </link>
    <category term="CustomDataService.Order"
              scheme="http://schemas.microsoft.com/ado/2007/08/dataservices/scheme" />
    <content type="application/xml">
      <m:properties>
        <d:Customer>Peter Franken</d:Customer>
      </m:properties>
    </content>
  </entry>
```

See Also

Reflection Provider

Interceptors (WCF Data Services)

7/23/2018 • 2 minutes to read • Edit Online

WCF Data Services enables an application to intercept request messages so that you can add custom logic to an operation. You can use this custom logic to validate data in incoming messages. You can also use it to further restrict the scope of a query request, such as to insert a custom authorization policy on a per request basis.

Interception is performed by specially attributed methods in the data service. These methods are called by WCF Data Services at the appropriate point in message processing. Interceptors are defined on a per-entity set basis, and interceptor methods cannot accept parameters from the request like service operations can. Query interceptor methods, which are called when processing an HTTP GET request, must return a lambda expression that determines whether an instance of the interceptor's entity set should be returned by the query results. This expression is used by the data service to further refine the requested operation. The following is an example definition of a query interceptor.

```
// Define a query interceptor for the Orders entity set.
[QueryInterceptor("Orders")]
public Expression<Func<Order, bool>> OnQueryOrders()
```

```
' Define a query interceptor for the Orders entity set.
<QueryInterceptor("Orders")> _
Public Function OnQueryOrders() As Expression(Of Func(Of Order, Boolean))
```

For more information, see How to: Intercept Data Service Messages.

Change interceptors, which are called when processing non-query operations, must return `void` (`Nothing` in Visual Basic). Change interceptor methods must accept the following two parameters:

1. A parameter of a type that is compatible with the entity type of the entity set. When the data service invokes the change interceptor, the value of this parameter will reflect the entity information that is sent by the request.

2. A parameter of type UpdateOperations. When the data service invokes the change interceptor, the value of this parameter will reflect the operation that the request is trying to perform.

The following is an example definition of a change interceptor.

```
// Define a change interceptor for the Products entity set.
[ChangeInterceptor("Products")]
public void OnChangeProducts(Product product, UpdateOperations operations)
```

```
' Define a change interceptor for the Products entity set.
<ChangeInterceptor("Products")> _
Public Sub OnChangeProducts(ByVal product As Product, _
                            ByVal operations As UpdateOperations)
```

For more information, see How to: Intercept Data Service Messages.

The following attributes are supported for interception.

[QueryInterceptor(*EntitySetName* **)]**

Methods with the QueryInterceptorAttribute attribute applied are called when an HTTP GET request is received for the targeted entity set resource. These methods must always return a lambda expression in the form of `Expression<Func<T,bool>>`.

[ChangeInterceptor(*EntitySetName* **)]**
Methods with the ChangeInterceptorAttribute attribute applied are called when an HTTP request other than HTTP GET request is received for the targeted entity set resource. These methods must always return `void` (`Nothing` in Visual Basic).

For more information, see How to: Intercept Data Service Messages.

See Also

Service Operations

How to: Intercept Data Service Messages (WCF Data Services)

5/2/2018 • 3 minutes to read • Edit Online

With WCF Data Services, you can intercept request messages so that you can add custom logic to an operation. To intercept a message, you use specially attributed methods in the data service. For more information, see Interceptors.

The example in this topic uses the Northwind sample data service. This service is created when you complete the WCF Data Services quickstart.

To define a query interceptor for the Orders entity set

1. In the Northwind data service project, open the Northwind.svc file.

2. In the code page for the `Northwind` class, add the following `using` statement (`Imports` in Visual Basic).

```
using System.Linq.Expressions;
```

```
Imports System.Linq.Expressions
```

3. In the `Northwind` class, define a service operation method named `OnQueryOrders` as follows:

```
// Define a query interceptor for the Orders entity set.
[QueryInterceptor("Orders")]
public Expression<Func<Order, bool>> OnQueryOrders()
```

```
' Define a query interceptor for the Orders entity set.
<QueryInterceptor("Orders")> _
Public Function OnQueryOrders() As Expression(Of Func(Of Order, Boolean))
```

To define a change interceptor for the Products entity set

1. In the Northwind data service project, open the Northwind.svc file.

2. In the `Northwind` class, define a service operation method named `OnChangeProducts` as follows:

```
// Define a change interceptor for the Products entity set.
[ChangeInterceptor("Products")]
public void OnChangeProducts(Product product, UpdateOperations operations)
```

```
' Define a change interceptor for the Products entity set.
<ChangeInterceptor("Products")> _
Public Sub OnChangeProducts(ByVal product As Product, _
                            ByVal operations As UpdateOperations)
```

Example

This example defines a query interceptor method for the `Orders` entity set that returns a lambda expression. This expression contains a delegate that filters the requested `Orders` based on related `Customers` that have a specific contact name. The name is in turn determined based on the requesting user. This example assumes that the data service is hosted within an ASP.NET Web application that uses WCF, and that authentication is enabled. The HttpContext class is used to retrieve the principle of the current request.

```
// Define a query interceptor for the Orders entity set.
[QueryInterceptor("Orders")]
public Expression<Func<Order, bool>> OnQueryOrders()
{
    // Filter the returned orders to only orders
    // that belong to a customer that is the current user.
    return o => o.Customer.ContactName ==
        HttpContext.Current.User.Identity.Name;
}
```

```
' Define a query interceptor for the Orders entity set.
<QueryInterceptor("Orders")> _
Public Function OnQueryOrders() As Expression(Of Func(Of Order, Boolean))
    ' Filter the returned orders to only orders
    ' that belong to a customer that is the current user.
    Return Function(o) o.Customer.ContactName = _
        HttpContext.Current.User.Identity.Name
End Function
```

Example

This example defines a change interceptor method for the `Products` entity set. This method validates input to the service for an Add or Change operation and raises an exception if a change is being made to a discontinued product. It also blocks the deletion of products as an unsupported operation.

```
// Define a change interceptor for the Products entity set.
[ChangeInterceptor("Products")]
public void OnChangeProducts(Product product, UpdateOperations operations)
{
    if (operations == UpdateOperations.Change)
    {
        System.Data.Objects.ObjectStateEntry entry;

        if (this.CurrentDataSource.ObjectStateManager
            .TryGetObjectStateEntry(product, out entry))
        {
            // Reject changes to a discontinued Product.
            // Because the update is already made to the entity by the time the
            // change interceptor in invoked, check the original value of the Discontinued
            // property in the state entry and reject the change if 'true'.
            if ((bool)entry.OriginalValues["Discontinued"])
            {
                throw new DataServiceException(400, string.Format(
                            "A discontinued {0} cannot be modified.", product.ToString()));
            }
        }
        else
        {
            throw new DataServiceException(string.Format(
                "The requested {0} could not be found in the data source.", product.ToString()));
        }
    }
    else if (operations == UpdateOperations.Delete)
    {
        // Block the delete and instead set the Discontinued flag.
        throw new DataServiceException(400,
            "Products cannot be deleted; instead set the Discontinued flag to 'true'");
    }
}
```

```
' Define a change interceptor for the Products entity set.
<ChangeInterceptor("Products")> _
Public Sub OnChangeProducts(ByVal product As Product, _
                            ByVal operations As UpdateOperations)
    If operations = UpdateOperations.Change Then
        Dim entry As System.Data.Objects.ObjectStateEntry

        If Me.CurrentDataSource.ObjectStateManager _
            .TryGetObjectStateEntry(product, entry) Then

            ' Reject changes to a discontinued Product.
            ' Because the update is already made to the entity by the time the
            ' change interceptor in invoked, check the original value of the Discontinued
            ' property in the state entry and reject the change if 'true'.
            If CType(entry.OriginalValues("Discontinued"), Boolean) Then
                Throw New DataServiceException(400, String.Format(
                            "A discontinued {0} cannot be modified.", product.ToString()))
            Else
                Throw New DataServiceException(String.Format( _
                    "The requested {0} could not be found in the data source.", product.ToString()))
            End If
        ElseIf (operations = UpdateOperations.Delete) Then
            ' Block the delete and instead set the Discontinued flag.
            Throw New DataServiceException(400, _
                "Products cannot be deleted; instead set the Discontinued flag to 'true'")
        End If
    End If
End Sub
```

See Also

How to: Define a Service Operation

Defining WCF Data Services

Develop and Deploy WCF Data Services

10/16/2018 • 7 minutes to read • Edit Online

This topic provides information about developing and deploying WCF Data Services. For more basic information about WCF Data Services, see Getting Started and Overview.

Develop WCF Data Services

When you use WCF Data Services to create a data service that supports the Open Data Protocol (OData), you must perform the following basic tasks during development:

1. **Define the data model**

   WCF Data Services supports a variety of data service providers that enable you to define a data model based on data from a variety of data sources, from relational databases to late-bound data types. For more information, see Data Services Providers.

2. **Create the data service**

   The most basic data service exposes a class that inherits from the DataService<T> class, with a type `T` that is the namespace-qualified name of the entity container. For more information, see Defining WCF Data Services.

3. **Configure the data service**

   By default, WCF Data Services disables access to resources that are exposed by an entity container. The DataServiceConfiguration interface enables you to configure access to resources and service operations, specify the supported version of OData, and to define other service-wide behaviors, such as batching behaviors or the maximum number of entities that can be returned in a single response feed. For more information, see Configuring the Data Service.

This topic covers primarily the development and deployment of data services by using Visual Studio. For information about the flexibility provided by WCF Data Services for exposing your data as OData feeds, see Defining WCF Data Services.

Choose a Development Web Server

When you develop a WCF Data Service as an ASP.NET application or ASP.NET Web site by using Visual Studio 2015, you have a choice of Web servers on which to run the data service during development. The following Web servers integrate with Visual Studio to make it easier to test and debug your data services on the local computer.

1. **Local IIS Server**

   When you create a data service that is an ASP.NET application or ASP.NET Web site that runs on Internet Information Services (IIS), we recommend that you develop and test your data service by using IIS on the local computer. Running the data service on IIS makes it easier to trace HTTP requests during debugging. This also enables you to pre-determine the necessary rights required by IIS to access files, databases, and other resources required by the data service. To run your data service on IIS, you must makes sure that both IIS and Windows Communication Foundation (WCF) are installed and configured correctly and grant access to IIS accounts in the file system and databases. For more information, see How to: Develop a WCF Data Service Running on IIS.

> **NOTE**
> You must run Visual Studio with administrator rights to enable the develop environment to configure the local IIS server.

2. **Visual Studio Development Server**

   Visual Studio includes a built-in Web server, the Visual Studio Development Server, which is the default Web server for ASP.NET projects. This Web server is designed to run ASP.NET projects on the local computer during development. The WCF Data Services quickstart shows how to create a data service that runs in the Visual Studio Development Server.

   You should be aware of the following limitations when you use the Visual Studio Development Server to develop the data service:

   - This server can only be accessed on the local computer.
   - This server listens on `localhost` and on a specific port, not on port 80, which is the default port for HTTP messages. For more information, see Web Servers in Visual Studio for ASP.NET Web Projects.
   - This server runs the data service in the context of your current user account. For example, if you are running as an administrator-level user, a data service running in the Visual Studio Development Server will have administrator-level privileges. This can cause the data service to be able to access resources that it does not have the rights to access when deployed to an IIS server.
   - This server does not include the extra facilities of IIS, such as authentication.
   - This server cannot handle chunked HTTP streams, which are sent be default by the WCF Data Services client when accessing large binary data from the data service. For more information, see Streaming Provider.
   - This server has issues with processing the period (`.`) character in a URL, even though this character is supported by WCF Data Services in key values.

   > **TIP**
   > Even though you can use the Visual Studio Development Server to test your data services during development, you should test them again after deploying to a Web server that is running IIS.

3. **Windows Azure Development Environment**

   Windows Azure Tools for Visual Studio includes an integrated set of tools for developing Windows Azure services in Visual Studio. With these tools, you can develop a data service that can be deployed to Windows Azure, and you can test the data service on the local computer before deployment. Use these tools when using Visual Studio to develop a data service that runs on the Windows Azure platform. You can download the Windows Azure Tools for Visual Studio from the Microsoft Download Center. For more information about developing a data service that runs on Windows Azure, see the post Deploying an OData Service in Windows Azure.

Development Tips

You should consider the following when you develop a data service:

- Determine the security requirements of your data service, if you plan authenticate users or restrict access for specific users. For more information, see Securing WCF Data Services.
- An HTTP inspection program can be very helpful when debugging a data service by enabling you to

inspect the contents of request and response messages. Any network packet analyzer that can display raw packets can be used to inspect HTTP requests to and responses from the data service.

- When debugging a data service, you may want to get more information about an error from the data service than during regular operation. You can get additional error information from the data service by setting the UseVerboseErrors property in the DataServiceConfiguration to `true` and by setting the IncludeExceptionDetailInFaults property of the ServiceDebugBehavior attribute on the data service class to `true`. For more information, see the post Debugging WCF Data Services. You can also enable tracing in WCF to view exceptions raised in the HTTP messaging layer. For more information, see Configuring Tracing.

- A data service is usually developed as an ASP.NET application project, but you can also create you data service as an ASP.NET Web site project in Visual Studio. For information about the differences between the two types of projects, see NIB: Web Application Projects versus Web Site Projects in Visual Studio.

- When you create a data service by using the **Add New Item** dialog box in Visual Studio, the data service is hosted by ASP.NET in IIS. While ASP.NET and IIS is the default host for a data service, other hosting options are supported. For more information, see Hosting the Data Service.

Deploy WCF Data Services

WCF Data Service provides flexibility in choosing the process that hosts the data service. You can use Visual Studio to deploy a data service to the following platforms:

- **IIS-Hosted Web Server**

  When a data service is developed as an ASP.NET project, it can be deployed to an IIS Web server by using the standard ASP.NET deployment processes. Visual Studio provides the following deployment technologies for ASP.NET, depending on the kind of ASP.NET project that hosts the data service that you are deploying.

  - **Deployment Technologies for ASP.NET Web Applications**

    - Web Deployment Package

    - One-Click Publishing

  - **Deployment Technologies for ASP.NET Web Sites**

    - Copy Web Site Tool

    - Publish Web Site Tool

    - XCopy

  For more information about the deployment options for an ASP.NET application, see Web Deployment Overview for Visual Studio and ASP.NET.

  > **TIP**
  >
  > Before you attempt to deploy the data service to IIS, make sure that you have tested the deployment to a Web server that is running IIS. For more information, see How to: Develop a WCF Data Service Running on IIS.

- **Windows Azure**

  You can deploy a data service to Windows Azure by using Windows Azure Tools for Visual Studio. You can download the Windows Azure Tools for Visual Studio from the Microsoft Download Center. For more information about deploying a data service to Windows Azure, see the post Deploying an OData Service in

Windows Azure.

Deployment Considerations

You should consider the following when deploying a data service:

- When you deploy a data service that uses the Entity Framework provider to access a SQL Server database, you might also have to propagate data structures, data, or both with your data service deployment. Visual Studio can automatically create scripts (.sql files) to do this in the destination database, and these scripts can be included in the Web deployment package of an ASP.NET application. For more information, see How to: Deploy a Database With a Web Application Project. For an ASP.NET Web site, you can do this by using the **Database Publishing Wizard** in Visual Studio. For more information, see Deploying a Database by Using the Database Publishing Wizard.

- Because WCF Data Services includes a basic WCF implementation, you can use Windows Server AppFabric to monitor a data service deployed to IIS running on Windows Server. For more information about using Windows Server AppFabric to monitor a data service, see the post Tracking WCF Data Services with Windows Server AppFabric.

See Also

- Hosting the Data Service
- Securing WCF Data Services
- Defining WCF Data Services

Securing WCF Data Services

8/31/2018 • 9 minutes to read • Edit Online

This topic describes security considerations that are specific to developing, deploying, and running WCF Data Services and applications that access services that support the Open Data Protocol (OData). You should also follow recommendations for creating secure .NET Framework applications.

When planning how to secure a WCF Data Services-based OData service, you must address both authentication, the process of discovering and verifying the identity of a principal, and authorization, the process of determining whether an authenticated principal is allowed to access the requested resources. You should also consider whether to encrypt the message by using SSL.

Authenticating Client Requests

WCF Data Services does not implement any kind of authentication of its own, but rather relies on the authentication provisions of the data service host. This means that the service assumes that any request that it receives has already been authenticated by the network host and that the host has correctly identified the principle for the request appropriately via the interfaces provided by WCF Data Services. These authentication options and approaches are detailed in OData and Authentication series.

Authentication Options for a WCF Data Service

The following table lists some of the authentication mechanisms that are available to help you authenticate requests to a WCF Data Service.

AUTHENTICATION OPTIONS	DESCRIPTION
Anonymous authentication	When HTTP anonymous authentication is enabled, any principle is able to connect to the data service. Credentials are not required for anonymous access. Use this option only when you want to allow anyone to access the data service.

AUTHENTICATION OPTIONS	DESCRIPTION
Basic and digest authentication	Credentials consisting of a user name and password are required for authentication. Supports authentication of non-Windows clients. **Security Note:** Basic authentication credentials (user name and password) are sent in the clear and can be intercepted. Digest authentication sends a hash based-on the supplied credentials, which makes it more secure than basic authentication. Both are susceptible to man-in-the-middle attacks. When using these authentication methods, you should consider encrypting communication between client and the data service by using the Secure Sockets Layer (SSL). Microsoft Internet Information Services (IIS) provides an implementation of both basic and digest authentication for HTTP requests in an ASP.NET application. This Windows Authentication Provider implementation enables a .NET Framework client application to supply credentials in the HTTP header of the request to the data service to seamlessly negotiate authentication of a Windows user. For more information, see Digest Authentication Technical Reference. When you want to have your data service use basic authentication based on some custom authentication service and not Windows credentials, you must implement a custom ASP.NET HTTP Module for authentication. For an example of how to use a custom basic authentication scheme with WCF Data Services, see the post on Custom Basic Authentication in the OData and authentication series.
Windows authentication	Windows-based credentials are exchanged by using NTLM or Kerberos. This mechanism is more secure than basic or digest authentication, but it requires that the client be a Windows-based application. IIS also provides an implementation of Windows authentication for HTTP requests in an ASP.NET application. For more information, see ASP.NET Forms Authentication Overview. For an example of how to use Windows authentication with WCF Data Services, see the post on Windows Authentication in the OData and authentication series.

AUTHENTICATION OPTIONS	DESCRIPTION
ASP.NET forms authentication	Forms authentication lets you authenticate users by using your own code and then maintain an authentication token in a cookie or in the page URL. You authenticate the user name and password of your users using a login form that you create. Unauthenticated requests are redirected to a login page, where the user provides credentials and submits the form. If the application authenticates the request, the system issues a ticket that contains a key for reestablishing the identity for subsequent requests. For more information, see Forms Authentication Provider. **Security Note:** By default, the cookie that contains the forms authentication ticket is not secured when you use forms authentication in a ASP.NET Web application. You should consider requiring SSL to protect both the authentication ticket and the initial login credentials. For an example of how to use forms authentication with WCF Data Services, see the post on Forms Authentication in the OData and authentication series.
Claims-based authentication	In claims-based authentication, the data service relies on a trusted "third-party" identity provider service to authenticate the user. The identity provider positively authenticates the user that is requesting access to data service resources and issues a token that grants access to the requested resources. This token is then presented to the data service, which then grants access to the user based on the trust relationship with the identity service that issued the access token. The benefit of using a claims-based authentication provider is that they can be used to authenticate various types of clients across trust domains. By employing such a third-party provider, a data service can offload the requirements of maintaining and authenticating users. OAuth 2.0 is a claims-based authentication protocol that is supported by Windows Azure AppFabric Access Control for federated authorization as a service. This protocol supports REST-based services. For an example of how to use OAuth 2.0 with WCF Data Services, see the post on OData and OAuth in the OData and authentication series.

Authentication in the Client Library

By default, the WCF Data Services client library does not supply credentials when making a request to an OData service. When login credentials are required by the data service to authenticate a user, these credentials can be supplied in a NetworkCredential accessed from the Credentials property of the DataServiceContext, as in the following example:

```
// Set the client authentication credentials.
context.Credentials =
    new NetworkCredential(userName, password, domain);
```

```
' Set the client authentication credentials.
context.Credentials = _
    New NetworkCredential(userName, password, domain)
```

For more information, see How to: Specify Client Credentials for a Data Service Request.

When the data service requires login credentials that cannot be specified by using a NetworkCredential object, such as a claims-based token or cookie, you must manually set headers in the HTTP request, usually the `Authorization` and `Cookie` headers. For more information about this kind of authentication scenario, see the post OData and Authentication: Client-Side Hooks. For an example of how to set HTTP headers in a request message, see How to: Set Headers in the Client Request.

Impersonation

Generally, the data service accesses required resources, such as files on the server or a database, by using the credentials of the worker process that is hosting the data service. When using impersonation, ASP.NET applications can execute with the Windows identity (user account) of the user making the request. Impersonation is commonly used in applications that rely on IIS to authenticate the user, and the credentials of this principle are used to access the required resources. For more information, see ASP.NET Impersonation.

Configuring Data Service Authorization

Authorization is the granting of access to application resources to a principle or process that is identified based on a previously successful authentication. As a general practice, you should only grant sufficient rights to users of the data service to perform the operations required by client applications.

Restrict Access to Data Service Resources

By default, WCF Data Services enables you to grant common read and write access against data service resources (entity set and service operations) to any user that is able to access the data service. Rules that define read and write access can be defined separately for each entity set exposed by the data service, as well as to any service operations. We recommend limiting both read and write access to only the resources required by the client application. For more information, see Minimum Resource Access Requirements.

Implement Role-Based Interceptors

Interceptors enable you to intercept requests against data service resources before they are acted on by the data service. For more information, see Interceptors. Interceptors enable you to make authorization decisions based the authenticated user that is making the request. For an example of how to restrict access to data service resources based on an authenticated user identity, see How to: Intercept Data Service Messages.

Restrict Access to the Persisted Data Store and Local Resources

The accounts that are used to access the persisted store should be granted only enough rights in a database or the file system to support the requirements of the data service. When anonymous authentication is used, this is the account used to run the hosting application. For more information, see How to: Develop a WCF Data Service Running on IIS. When impersonation is used, authenticated users must be granted access to these resources, usually as part of a Windows group.

Other Security Considerations

Secure the Data in the Payload

OData is based on the HTTP protocol. In an HTTP message, the header may contain valuable user credentials, depending on the authentication implemented by the data service. The message body may also contain valuable customer data that must be protected. In both of these cases, we recommend that you use SSL to protect this information over the wire.

Ignored Message Headers and Cookies

HTTP request headers, other than those that declare content types and resource locations, are ignored and are never set by the data service.

Cookies can be used as part of an authentication scheme, such as with ASP.NET Forms Authentication. However, any HTTP cookies set on an incoming request are ignored by WCF Data Services. The host of a data service may process the cookie, but the WCF Data Services runtime never analyzes or returns cookies. The WCF Data Services client library also does not process cookies sent in the response.

Custom Hosting Requirements

By default, WCF Data Services is created as an ASP.NET application hosted in IIS. This enables the data service to leverage the secure behaviors of this platform. You can define WCF Data Services that are hosted by a custom host. For more information, see Hosting the Data Service. The components and platform hosting a data service must ensure the following security behaviors to prevent attacks on the data service:

- Limit the length of the URI accepted in a data service request for all possible operations.
- Limit the size of both incoming and outgoing HTTP messages.
- Limit the total number of outstanding requests at any given time.
- Limit the size of HTTP headers and their values, and provide WCF Data Services access to header data.
- Detect and counter known attacks, such as TCP SYN and message replay attacks.

Values Are Not Further Encoded

Property values sent to the data service are not further encoded by the WCF Data Services runtime. For example, when a string property of an entity contains formatted HTML content, the tags are not HTML encoded by the data service. The data service also does not further encode property values in the response. The client library also does not perform any additional encoding.

Considerations for Client Applications

The following security considerations apply to applications that use the WCF Data Services client to access OData services:

- The client library assumes that the protocols used to access the data service provide an appropriate level of security.
- The client library uses all the default values for timeouts and parsing options of the underlying platform-provided transport stacks.
- The client library does not read any settings from application configuration files.
- The client library does not implement any cross domain access mechanisms. Instead, it relies on the mechanisms provided by the underlying HTTP stack.
- The client library has no user interface elements, and it never tries to display or render the data that it receives or sends.
- We recommend that client applications always validate user input as well as data accepted from untrusted services.

See Also

Defining WCF Data Services
WCF Data Services Client Library

Hosting the Data Service (WCF Data Services)

8/29/2018 • 3 minutes to read • Edit Online

By using WCF Data Services, you can create a service that exposes data as an Open Data Protocol (OData) feed. This data service is defined as a class that inherits from DataService<T>. This class provides the functionality required to process request messages, perform updates against the data source, and generate responses messages, as required by OData. However, a data service cannot bind to and listen on a network socket for incoming HTTP requests. For this required functionality, the data service relies on a hosting component.

The data service host performs the following tasks on behalf of the data service:

- Listens for requests and routes these requests to the data service.
- Creates an instance of the data service for each request.
- Requests that the data service process the incoming request.
- Sends the response on behalf of the data service.

To simplify hosting a data service, WCF Data Services is designed to integrate with Windows Communication Foundation (WCF). The data service provides a default WCF implementation that serves as the data service host in an ASP.NET application. Therefore, you can host a data service in one of the following ways:

- In an ASP.NET application.
- In a managed application that supports self-hosted WCF services.
- In some other custom data service host.

Hosting a Data Service in an ASP.NET Application

When you use the **Add New Item** dialog in Visual Studio 2015 to define a data service in an ASP.NET application, the tool generates two new files in the project. The first file has a `.svc` extension and instructs the WCF runtime how to instantiate the data service. The following is an example of this file for the Northwind sample data service created when you complete the quickstart:

```
<%@ ServiceHost Language="C#"
    Factory="System.Data.Services.DataServiceHostFactory,
            System.Data.Services, Version=4.0.0.0,
            Culture=neutral, PublicKeyToken=b77a5c561934e089"
    Service="NorthwindService.Northwind" %>
```

This directive tells the application to create the service host for the named data service class by using the DataServiceHostFactory class.

The code-behind page for the `.svc` file contains the class that is the implementation of the data service itself, which is defined as follows for the Northwind sample data service:

```
public class Northwind : DataService<NorthwindEntities>
```

```
Public Class Northwind
    Inherits DataService(Of NorthwindEntities)
```

Because a data service behaves like a WCF service, the data service integrates with ASP.NET and follows the WCF Web programming model. For more information, see WCF Services and ASP.NET and WCF Web HTTP Programming Model.

Self-Hosted WCF Services

Because it incorporates a WCF implementation, WCF Data Services support self-hosting a data service as a WCF service. A service can be self-hosted in any .NET Framework application, such as a console application. The DataServiceHost class, which inherits from WebServiceHost, is used to instantiate the data service at a specific address.

Self-hosting can be used for development and testing because it can make it easier to deploy and troubleshoot the service. However, this kind of hosting does not provide the advanced hosting and management features provided by ASP.NET or by Internet Information Services (IIS). For more information, see Hosting in a Managed Application.

Defining a Custom Data Service Host

For cases where the WCF host implementation is too restrictive, you can also define a custom host for a data service. Any class that implements IDataServiceHost interface can be used as the network host for a data service. A custom host must implement the IDataServiceHost interface and be able to handle the following basic responsibilities of the data service host:

- Provide the data service with the service root path.
- Process request and response headers information to the appropriate IDataServiceHost member implementation.
- Handle exceptions raised by the data service.
- Validate parameters in the query string.

See Also

- Defining WCF Data Services
- Exposing Your Data as a Service
- Configuring the Data Service

Data Service Versioning (WCF Data Services)

8/31/2018 • 6 minutes to read • Edit Online

The Open Data Protocol (OData) enables you to create data services so that clients can access data as resources using URIs that are based on a data model. OData also supports the definition of service operations. After initial deployment, and potentially several times during their lifetime, these data services may need to be changed for a variety of reasons, such as changing business needs, information technology requirements, or to address other issues. When you make changes to an existing data service, you must consider whether to define a new version of your data service and how best to minimize the impact on existing client applications. This topic provides guidance on when and how to create a new version of a data service. It also describes how WCF Data Services handles an exchange between clients and data services that support different versions of the OData protocol.

Versioning a WCF Data Service

Once a data service is deployed and the data is being consumed, changes to the data service have the potential to cause compatibility issues with existing client applications. However, because changes are often required by the overall business needs of the service, you must consider when and how to create a new version of your data service with the least impact on client applications.

Data Model Changes that Recommend a New Data Service Version

When considering whether to publish a new version of a data service, it is important to understand how the different kinds of changes may affect client applications. Changes to a data service that might require you to create a new version of a data service can be divided into the following two categories:

- Changes to the service contract—which include updates to service operations, changes to the accessibility of entity sets (feeds), version changes, and other changes to service behaviors.
- Changes to the data contract—which include changes to the data model, feed formats, or feed customizations.

The following table details for which kinds of changes you should consider publishing a new version of the data service:

TYPE OF CHANGE	REQUIRES A NEW VERSION	NEW VERSION NOT NEEDED
Service operations	- Add new parameter - Change return type - Remove service operation	- Delete existing parameter - Add new service operation
Service behaviors	- Disable count requests - Disable projection support - Increase the required data service version	- Enable count requests - Enable projection support - Decrease the required data service version
Entity set permissions	- Restrict entity set permissions - Change response code (new first digit value) [1]	- Relax entity set permissions - Change response code (same first digit value)
Entity properties	- Remove existing property or relationship - Add non-nullable property - Change existing property	- Add nullable property[2]

TYPE OF CHANGE	REQUIRES A NEW VERSION	NEW VERSION NOT NEEDED
Entity sets	- Remove entity set	- Add derived type - Change base type - Add entity set
Feed customization	- Change entity-property mapping	

[1] This may depend on how strictly a client application relies on receiving a specific error code.

[2] You can set the IgnoreMissingProperties property to `true` to have the client ignore any new properties sent by the data service that are not defined on the client. However, when inserts are made, the properties not included by the client in the POST request are set to their default values. For updates, any existing data in a property unknown to the client might be overwritten with default values. In this case, you should send the update as a MERGE request, which is the default. For more information, see Managing the Data Service Context.

How to Version a Data Service

When required, a new data service version is defined by creating a new instance of the service with an updated service contract or data model. This new service is then exposed by using a new URI endpoint, which differentiates it from the previous version. For example:

- Old version: `http://services.odata.org/Northwind/v1/Northwind.svc/`
- New version: `http://services.odata.org/Northwind/v2/Northwind.svc/`

When upgrading a data service, clients will need to also be updated based on the new data service metadata and to use the new root URI. When possible, you should maintain the previous version of the data service to support clients that have not yet been upgraded to use the new version. Older versions of a data service can be removed when they are no longer needed. You should consider maintaining the data service endpoint URI in an external configuration file.

OData Protocol Versions

As new versions of OData are released, client applications may not be using the same version of the OData protocol that is supported by the data service. An older client application may access a data service that supports a newer version of OData. A client application may also be using a newer version of the WCF Data Services client library, which supports a newer version of OData than does the data service that is being accessed.

WCF Data Services leverages the support provided by OData to handle such versioning scenarios. There is also support for generating and using data model metadata to create client data service classes when the client uses a different version of OData than the data service uses. For more information, see OData: Protocol Versioning.

Version Negotiation

The data service can be configured to define the highest version of the OData protocol that will be used by the service, regardless of the version requested by the client. You can do this by specifying a DataServiceProtocolVersion value for the MaxProtocolVersion property of the DataServiceBehavior used by the data service. For more information, see Configuring the Data Service.

When an application uses the WCF Data Services client libraries to access a data service, the libraries automatically set these headers to the correct values, depending on the version of OData and the features that are used in your application. By default, WCF Data Services uses the lowest protocol version that supports the requested operation.

The following table details the versions of .NET Framework and Silverlight that include WCF Data Services support for specific versions of the OData protocol.

ODATA PROTOCOL VERSION	SUPPORT INTRODUCED IN...
Version 1	- .NET Framework version 3.5 Service Pack 1 (SP1) - Silverlight version 3
Version 2	- .NET Framework version 4 - An update to .NET Framework version 3.5 SP1. You can download and install the update from the Microsoft Download Center. - Silverlight version 4
Version 3	- You can download and install a pre-release version that supports OData version 3 from the Microsoft Download Center.

Metadata Versions

By default, WCF Data Services uses version 1.1 of CSDL to represent a data model. This is always the case for data models that are based on either a reflection provider or a custom data service provider. However, when the data model is defined by using the Entity Framework, the version of CSDL returned is the same as the version that is used by the Entity Framework. The version of the CSDL is determined by the namespace of the Schema element. For more information, see the specification [MC-CSDL]: Conceptual Schema Definition File Format.

The `DataServices` element of the returned metadata also contains a `DataServiceVersion` attribute, which is the same value as the `DataServiceVersion` header in the response message. Client applications, such as the **Add Service Reference** dialog box in Visual Studio, use this information to generate client data service classes that work correctly with the version of WCF Data Services that host the data service. For more information, see OData: Protocol Versioning.

See Also

- Data Services Providers
- Defining WCF Data Services

WCF Data Services Protocol Implementation Details

8/31/2018 • 2 minutes to read • Edit Online

OData Protocol Implementation Details

The Open Data Protocol (OData) requires that a data service that implements the protocol provide a certain minimum set of functionalities. These functionalities are described in the protocol documents in terms of "should" and "must." Other optional functionality is described in terms of "may." This topic describes these optional functionalities that are not currently implemented by WCF Data Services. For more information, see OData Protocol Documentation.

Support for the $format Query Option

The OData protocol supports both JavaScript Notation (JSON) and Atom feeds, and OData provides the `$format` system query option to allow a client to request the format of the response feed. This system query option, if supported by the data service, must override the value of the Accept header of the request. WCF Data Services supports returning both JSON and Atom feeds. However, the default implementation does not support the `$format` query option and uses only the value of the Accept header to determine the format of the response. There are cases when your data service may need to support the `$format` query option, such as when clients cannot set the Accept header. To support these scenarios, you must extend your data service to handle this option in the URI. You can add this functionality to your data service by downloading the JSONP and URL-controlled format support for ADO.NET Data Services sample project from the MSDN Code Gallery web site and adding it to your data service project. This sample removes the `$format` query option and changes the Accept header to `application/json`. When you include the sample project and adding the `JSONPSupportBehaviorAttribute` to your data service class enables the service to handle the `$format` query option `$format=json`. Further customization of this sample project is required to also handle `$format=atom` or other custom formats.

WCF Data Services Behaviors

The following WCF Data Services behaviors are not explicitly defined by the OData protocol:

Default Sorting Behavior

When a query request that is sent to the data service includes a `$top` or `$skip` system query option and does not include the `$orderby` system query option, the returned feed is sorted by the key properties, in ascending order. This is because ordering is required to ensure the correct paging of results. To do this, the data service adds an ordering expression to the query. This behavior also occurs when server-driven paging is enabled in the data service. For more information, see Configuring the Data Service.To control the ordering of the returned feed, you should include `$orderby` in the query URI.

See Also

Defining WCF Data Services
WCF Data Services Client Library

Using actions to implement server-side behavior

9/19/2018 • 5 minutes to read • Edit Online

OData Actions provide a way to implement a behavior that acts upon a resource retrieved from an OData service. For example consider a digital movie as a resource, there are many things you may do with a digital movie: check-out, rate/comment, or check-in. These are all examples of Actions that may be implemented by a WCF Data Service that manages digital movies. Actions are described in an OData response that contains a resource on which the Action can be invoked. When a user requests a resource that represents a digital movie the response returned from the WCF Data Service contains information about the Actions that are available for that resource. The availability of an Action can depend on the state of the data service or resource. For example once a digital movie is checked out it cannot be checked out by another user. Clients can invoke an action simply by specifying a URL. For example, `http://MyServer/MovieService.svc/Movies(6)` would identify a specific digital movie and `http://MyServer/MovieService.svc/Movies(6)/Checkout` would invoke the action on the specific movie. Actions allow you to expose you service model without exposing your data model. Continuing with the movie service example, you may wish to allow a user to rate a movie, but not directly expose the rating data as a resource. You could implement a Rate Action to allow the user to rate a movie but not directly access the rating data as a resource.

Implementing an action

To implement a service action you must implement the IServiceProvider, IDataServiceActionProvider, and IDataServiceInvokable interfaces. IServiceProvider allows WCF Data Services to get your implementation of IDataServiceActionProvider. IDataServiceActionProvider allows WCF Data Services to create, find, describe and invoke service actions. IDataServiceInvokable allows you to invoke the code that implements the service actions' behavior and get the results, if any. Keep in mind that WCF Data Services are Per-Call WCF Services, a new instance of the service will be created each time the service is called. Make sure no unnecessary work is done when the service is created.

IServiceProvider

IServiceProvider contains a method called GetService. This method is called by WCF Data Services to retrieve a number of service providers, including metadata service providers and data service action providers. When asked for a data service action provider, return your IDataServiceActionProvider implementation.

IDataServiceActionProvider

IDataServiceActionProvider contains methods that allow you to retrieve information about the available actions. When you implement IDataServiceActionProvider you are augmenting the metadata for your service which is defined by your service's implementation of IDataServiceActionProvider with Actions and handling dispatch to those actions as appropriate.

AdvertiseServiceAction

IDataServiceActionProvider is called to determine what actions are available for the specified resource. This method is only called for actions that are not always available. It is used to check if the action should be included in the OData response based upon the state of the resource being requested or the state of the service. How this check is accomplished is completely up to you. If it is an expensive to calculate availability and the current resource is in a feed, it is acceptable to skip the check and advertise the action. The `inFeed` parameter is set to `true` if the current resource being returned is part of a feed.

CreateInvokable

CreateInvokable is called to create a IDataServiceInvokable that contains a delegate that encapsulates the code that implements the action's behavior. This creates the IDataServiceInvokable instance but does not invoke the action. WCF Data Service Actions have side effects and need to work in conjunction with the Update Provider to save those changes to disk. The IDataServiceInvokable.Invoke method is called from the Update Provider's SaveChanges() method is called.

GetServiceActions

This method returns a collection of ServiceAction instances that represent all of the actions a WCF Data Service exposes. ServiceAction is the metadata representation of an Action, that includes information like the Action name, its parameters, and its return type.

GetServiceActionsByBindingParameterType

This method returns a collection of all ServiceAction instances that can be bound to the specified binding parameter type. In other words, all ServiceActions that can act on the specified resource type (also called binding parameter type).This is used when the service returns a resource in order to include information about Actions that can be invoked against that resource. This method should only return actions that can bind to the exact binding parameter type (no derived types). This method is called once per request per type encountered and the result is cached by WCF Data Services.

TryResolveServiceAction

This method searches for a specified ServiceAction and returns `true` if the ServiceAction is found. If found, the ServiceAction is returned in the `serviceAction` `out` parameter.

IDataServiceInvokable

This interface provides a way to execute a WCF Data Service Action. When implementing IDataServiceInvokable you are responsible for 3 things:

1. Capturing and potentially marshaling the parameters

2. Dispatching the parameters to the code that actually implements the Action when Invoke() is called

3. Storing any results from Invoke() so they can be retrieved using GetResult()

The parameters may be passed as tokens. This is because it is possible to write a Data Service Provider that works with tokens that represent resources, if this is the case you may need to convert (marshal) these tokens into actual resources before dispatching to the actual action. After the parameter has been marshalled, it must be in an editable state so that any changes to the resource that occur when the action is invoked will be saved and written to disk.

This interface requires two methods: Invoke and GetResult. Invoke invokes the delegate that implements the action's behavior and GetResult returns the result of the action.

Invoking a WCF Data Service Action

Actions are invoked using an HTTP POST request. The URL specifies the resource followed by the action name. Parameters are passed in the body of the request. For example if there was a service called MovieService which exposed an action called Rate. You could use the following URL to invoke the Rate action on a specific movie:

`http://MovieServer/MovieService.svc/Movies(1)/Rate`

Movies(1) specifies the movie you wish to rate and Rate specifies the Rate action. The actual value of the rating will be in the body of the HTTP request as shown in the following example:

```
POST http://MovieServer/MoviesService.svc/Movies(1)/Rate HTTP/1.1
Content-Type: application/json
Content-Length: 20
Host: localhost:15238
{
   "rating": 4
}
```

WARNING

The above sample code will work only with WCF Data Services 5.2 and later which has support for JSON light. If using an earlier version of WCF Data Services, you must specify the json verbose content-type as follows: `application/json;odata=verbose`.

Alternatively you can invoke an action using the WCF Data Services Client as shown in the following code snippet.

```
MoviesModel context = new MoviesModel (new Uri("http://MyServer/MoviesService.svc/"));
//...
context.Execute(new Uri("http://MyServer/MoviesService.svc/Movies(1)/Rate"), "POST", new
BodyOperationParameter("rating",4) );
```

In the code snippet above, the `MoviesModel` class was generated by using Visual Studio to Add Service Reference to a WCF Data Service.

See Also

WCF Data Services 4.5
Defining WCF Data Services
Developing and Deploying WCF Data Services
Custom Data Service Providers

WCF Data Services Client Library

8/31/2018 • 2 minutes to read • Edit Online

Any application can interact with an Open Data Protocol (OData)-based data service if it can send an HTTP request and process the OData feed that a data service returns. This interoperability enables you to access OData-based services from a broad range of Web-enabled applications. WCF Data Services includes client libraries that provide a richer programming experience when you consume OData feeds from .NET Framework or Silverlight-based applications.

The two main classes of the client library are the DataServiceContext class and the DataServiceQuery<TElement> class. The DataServiceContext class encapsulates operations that are supported against a specified data service. Although OData services are stateless, the context is not. Therefore, you can use the DataServiceContext class to maintain state on the client between interactions with the data service in order to support features such as change management. This class also manages identities and tracks changes. The DataServiceQuery<TElement> class represents a query against a specific entity set.

This section describes how to use client libraries to access and change data from a .NET Framework client application. For more information about how to use the WCF Data Services client library with a Silverlight-based application, see WCF Data Services (Silverlight). Other client libraries are available that enable you to consume an OData feed in other kinds of applications. For more information, see the OData SDK.

In This Section

Generating the Data Service Client Library
Describes how to generate a client library and client data service classes that are based on OData feeds.

Querying the Data Service
Describes how to query a data service from a .NET Framework-based application by using client libraries.

Loading Deferred Content
Describes how to load additional content not included in the initial query response.

Updating the Data Service
Describes how to create, modify, and delete entities and relationships by using the client libraries.

Asynchronous Operations
Describes the facilities provided by the client libraries for working with a data service in an asynchronous manner.

Batching Operations
Describes how to send multiple requests to the data service in a single batch by using the client libraries.

Binding Data to Controls
Describes how to bind controls to a OData feed returned by a data service.

Calling Service Operations
Describes how to use the client library to call service operations.

Managing the Data Service Context
Describes options for managing the behavior of the client library.

Working with Binary Data
Describes how to access and change binary data returned by the data service as a data stream.

See Also

Defining WCF Data Services

Getting Started

Generating the Data Service Client Library (WCF Data Services)

8/31/2018 • 2 minutes to read • Edit Online

A data service that implements the Open Data Protocol (OData) can return a service metadata document that describes the data model exposed by the OData feed. For more information, see OData: Service Metadata Document. You can use the **Add Service Reference** dialog in Visual Studio to add a reference to an OData-based service. When you use this tool to add a reference to the metadata returned by an OData feed in a client project, it performs the following actions:

- Requests the service metadata document from the data service and interprets the returned metadata.

  > **NOTE**
  >
  > The returned metadata is stored in the client project as an .edmx file. This .edmx file cannot be opened by using the Entity Data Model designer because it does not have the same format an .edmx file used by the Entity Framework. You can view this metadata file by using the XML editor or any text editor. For more information, see the [MC-EDMX]: Entity Data Model for Data Services Packaging Format specification

- Generates a representation of the service as an entity container class that inherits from DataServiceContext. This generated entity container class resembles the entity container that the Entity Data Model tools generate. For more information, see Object Services Overview (Entity Framework).
- Generates data classes for the data model types that it discovers in the service metadata.
- Adds a reference to the `System.Data.Services.Client` assembly to the project.

For more information, see How to: Add a Data Service Reference.

The client data service classes can also be generated by using the DataSvcUtil.exe tool at the command prompt. For more information, see How to: Manually Generate Client Data Service Classes.

Client Data Type Mapping

When you use the **Add Service Reference** dialog in Visual Studio or the `DataSvcUtil.exe` tool to generate client data classes that are based on an OData feed, the .NET Framework data types are mapped to the primitive types from the data model as follows:

DATA MODEL TYPE	.NET FRAMEWORK DATA TYPE
`Edm.Binary`	Byte `[]`
`Edm.Boolean`	Boolean
`Edm.Byte`	Byte
`Edm.DateTime`	DateTime
`Edm.Decimal`	Decimal

DATA MODEL TYPE	.NET FRAMEWORK DATA TYPE
`Edm.Double`	Double
`Edm.Guid`	Guid
`Edm.Int16`	Int16
`Edm.Int32`	Int32
`Edm.Int64`	Int64
`Edm.SByte`	SByte
`Edm.Single`	Single
`Edm.String`	String

For more information, see OData: Primitive Data Types.

See Also

WCF Data Services Client Library
Quickstart

WCF Data Service Client Utility (DataSvcUtil.exe)

8/31/2018 • 2 minutes to read • Edit Online

DataSvcUtil.exe is a command-line tool provided by WCF Data Services that consumes an Open Data Protocol (OData) feed and generates the client data service classes that are needed to access a data service from a .NET Framework client application. This utility can generate data classes by using the following metadata sources:

- The root URI of a data service. The utility requests the service metadata document, which describes the data model exposed by the data service. For more information, see OData: Service Metadata Document.
- A data model file (.csdl) defined by using the conceptual schema definition language (CSDL), as defined in the [MC-CSDL]: Conceptual Schema Definition File Format specification.
- An .edmx file created by using the Entity Data Model tools that are provided with the Entity Framework. For more information, see the [MC-EDMX]: Entity Data Model for Data Services Packaging Format specification.

For more information, see How to: Manually Generate Client Data Service Classes.

The DataSvcUtil.exe tool is installed in the .NET Framework directory. In many cases, this is located in *C:\Windows\Microsoft.NET\Framework\v4.0*. For 64-bit systems, this is located in *C:\Windows\Microsoft.NET\Framework64\v4.0*. You can also access the DataSvcUtil.exe tool from Developer Command Prompt for Visual Studio.

Syntax

```
datasvcutil /out:file [/in:file | /uri:serviceuri] [/dataservicecollection] [/language:devlang] [/nologo]
[/version:ver] [/help]
```

Parameters

OPTION	DESCRIPTION
`/dataservicecollection`	Specifies that the code required to bind objects to controls is also generated.

`/help`
 	
-or-
 	
`/?`
	Displays command syntax and options for the tool.

`/in:` *<file>*	Specifies the .csdl or .edmx file or a directory where the file is located.
`/language:` [VB|CSharp]	Specifies the language for the generated source code files. The language defaults to C#.
`/nologo`	Suppresses the copyright message from displaying.

OPTION	DESCRIPTION
`/out:` *<file>*	Specifies the name of the source code file that contains the generated client data service classes.
`/uri:` *<string>*	The URI of the OData feed.
`/version:` [1.0|2.0]	Specifies the highest accepted version of OData. The version is determined based on the `DataServiceVersion` attribute of the DataService element in the returned data service metadata. For more information, see Data Service Versioning. When you specify the `/dataservicecollection` parameter, you must also specify `/version:2.0` to enable data binding.

See Also

- Generating the Data Service Client Library
- How to: Add a Data Service Reference

How to: Add a data service reference (WCF Data Services)

8/29/2018 • 2 minutes to read • Edit Online

You can use the **Add Service Reference** dialog in Visual Studio to add a reference to WCF Data Services. This enables you to more easily access a data service in a client application that you develop in Visual Studio. When you complete this procedure, data classes are generated based on metadata that is obtained from the data service. For more information, see Generating the Data Service Client Library.

Add a data service reference

1. (Optional) If the data service is not part of the solution and is not already running, start the data service and note the URI of the data service.
2. In Visual Studio, 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**.

   -or-

   In the **Address** text box, type the base URL of the data service, such as `http://localhost:1234/Northwind.svc`, and then click **Go**.
4. Select **OK**.

   A new code file that contains the data classes that can access and interact with data service resources is added to the project.

See also

- Quickstart

How to: Manually Generate Client Data Service Classes (WCF Data Services)

8/31/2018 • 2 minutes to read • Edit Online

WCF Data Services integrates with Visual Studio to enable you to automatically generate client data service classes when you use the **Add Service Reference** dialog box to add a reference to a data service in a Visual Studio project. For more information, see How to: Add a Data Service Reference. You can also manually generate the same client data service classes by using the code-generation tool, `DataSvcUtil.exe`. This tool, which is included with WCF Data Services, generates .NET Framework classes from the data service definition. It can also be used to generate data service classes from the conceptual model (.csdl) file and from the .edmx file that represents an Entity Framework model in a Visual Studio project.

The example in this topic creates client data service classes based on the Northwind sample data service. This service is created when you complete the WCF Data Services quickstart. Some examples in this topic require the conceptual model file for the Northwind model. For more information, see How to: Use EdmGen.exe to Generate the Model and Mapping Files. Some examples in this topic require the .edmx file for the Northwind model. For more information, see .edmx File Overview.

To generate C# classes that support data binding

- At the command prompt, execute the following command without line breaks:

```
"%windir%\Microsoft.NET\Framework\v3.5\DataSvcUtil.exe" /dataservicecollection /version:2.0
/language:CSharp /out:Northwind.cs /uri:http://localhost:12345/Northwind.svc
```

> **NOTE**
> You must replace the value supplied to the `/uri:` parameter with the URI of your instance of the Northwind sample data service.

To generate Visual Basic classes that support data binding

- At the command prompt, execute the following command without line breaks:

```
"%windir%\Microsoft.NET\Framework\v3.5\DataSvcUtil.exe" /dataservicecollection /version:2.0
/language:VB /out:Northwind.vb /uri:http://localhost:12345/Northwind.svc
```

> **NOTE**
> You must replace value supplied to the `/uri:` parameter with the URI of your instance of the Northwind sample data service.

To generate C# classes based on the service URI

- At the command prompt, execute the following command without line breaks:

```
"%windir%\Microsoft.NET\Framework\v3.5\DataSvcUtil.exe" /language:CSharp /out:northwind.cs
/uri:http://localhost:12345/Northwind.svc
```

NOTE

You must replace the value supplied to the `/uri:` parameter with the URI of your instance of the Northwind sample data service.

To generate Visual Basic classes based on the service URI

- At the command prompt, execute the following command without line breaks:

```
"%windir%\Microsoft.NET\Framework\v3.5\datasvcutil.exe" /language:VB /out:Northwind.vb
/uri:http://localhost:12345/Northwind.svc
```

NOTE

You must replace value supplied to the `/uri:` parameter with the URI of your instance of the Northwind sample data service.

To generate C# classes based on the conceptual model file (CSDL)

- At the command prompt, execute the following command without line breaks:

```
"%windir%\Microsoft.NET\Framework\v3.5\datasvcutil.exe" /language:CSharp /in:Northwind.csdl
/out:Northwind.cs
```

To generate Visual Basic classes based on the conceptual model file (CSDL)

- At the command prompt, execute the following command without line breaks:

```
"%windir%\Microsoft.NET\Framework\v3.5\datasvcutil.exe" /language:VB /in:Northwind.csdl
/out:Northwind.vb
```

To generate C# classes based on the .edmx file

- At the command prompt, execute the following command without line breaks:

```
"%windir%\Microsoft.NET\Framework\v3.5\datasvcutil.exe" /language:CSharp /in:Northwind.edmx
/out:c:\northwind.cs
```

To generate Visual Basic classes based on the .edmx file

- At the command prompt, execute the following command without line breaks:

```
"%windir%\Microsoft.NET\Framework\v3.5\datasvcutil.exe" /language:VB /in:Northwind.edmx
/out:c:\northwind.vb
```

See Also

- Generating the Data Service Client Library
- How to: Add a Data Service Reference
- WCF Data Service Client Utility (DataSvcUtil.exe)

Querying the Data Service (WCF Data Services)

5/2/2018 • 8 minutes to read • Edit Online

The WCF Data Services client library enables you to execute queries against a data service by using familiar .NET Framework programming patterns, including using language integrated query (LINQ). The client library translates a query, which is defined on the client as an instance of the DataServiceQuery<TElement> class, into an HTTP GET request message. The library receives the response message and translates it into instances of client data service classes. These classes are tracked by the DataServiceContext to which the DataServiceQuery<TElement> belongs.

Data Service Queries

The DataServiceQuery<TElement> generic class represents a query that returns a collection of zero or more entity type instances. A data service query always belongs to an existing data service context. This context maintains the service URI and metadata information that is required to compose and execute the query.

When you use the **Add Service Reference** dialog to add a data service to a .NET Framework-based client application, an entity container class is created that inherits from the DataServiceContext class. This class includes properties that return typed DataServiceQuery<TElement> instances. There is one property for each entity set that the data service exposes. These properties make it easier to create an instance of a typed DataServiceQuery<TElement>.

A query is executed in the following scenarios:

- When results are enumerated implicitly, such as:
  - When a property on the DataServiceContext that represents and entity set is enumerated, such as during a `foreach` (C#) or `For Each` (Visual Basic) loop.
  - When the query is assigned to a `List` collection.
- When the Execute or BeginExecute method is explicitly called.
- When a LINQ query execution operator, such as First or Single is called.

The following query, when it is executed, returns all `Customers` entities in the Northwind data service:

```
// Define a new query for Customers.
DataServiceQuery<Customer> query = context.Customers;
```

```
' Define a new query for Customers.
Dim query As DataServiceQuery(Of Customer) = context.Customers
```

For more information, see How to: Execute Data Service Queries.

The WCF Data Services client supports queries for late-bound objects, such as when you use the *dynamic* type in C#. However, for performance reasons you should always compose strongly-typed queries against the data service. The Tuple type and dynamic objects are not supported by the client.

LINQ Queries

Because the DataServiceQuery<TElement> class implements the IQueryable<T> interface defined by LINQ, the

WCF Data Services client library is able to transform LINQ queries against entity set data into a URI that represents a query expression evaluated against a data service resource. The following example is a LINQ query that is equivalent to the previous DataServiceQuery<TElement> that returns `Orders` that have a freight cost of more than $30 and orders the results by the freight cost:

```
var selectedOrders = from o in context.Orders
                     where o.Freight > 30
                     orderby o.ShippedDate descending
                     select o;
```

```
Dim selectedOrders = From o In context.Orders _
        Where (o.Freight > 30) _
        Order By o.ShippedDate Descending _
        Select o
```

This LINQ query is translated into the following query URI that is executed against the Northwind-based quickstart data service:

```
http://localhost:12345/Northwind.svc/Orders?Orderby=ShippedDate&?filter=Freight gt 30
```

NOTE

The set of queries expressible in the LINQ syntax is broader than those enabled in the representational state transfer (REST)-based URI syntax that is used by data services. A NotSupportedException is raised when the query cannot be mapped to a URI in the target data service.

For more information, see LINQ Considerations.

Adding Query Options

Data service queries support all the query options that WCF Data Servicess provides. You call the AddQueryOption method to append query options to a DataServiceQuery<TElement> instance. AddQueryOption returns a new DataServiceQuery<TElement> instance that is equivalent to the original query but with the new query option set. The following query, when executed, returns `Orders` that are filtered by the `Freight` value and ordered by the `OrderID`, descending:

```
// Define a query for orders with a Freight value greater than 30
// and that is ordered by the ship date, descending.
DataServiceQuery<Order> selectedOrders = context.Orders
    .AddQueryOption("$filter", "Freight gt 30")
    .AddQueryOption("$orderby", "OrderID desc");
```

```
' Define a query for orders with a Freight value greater than 30
' and that is ordered by the ship date, descending.
Dim selectedOrders As DataServiceQuery(Of Order) = context.Orders _
.AddQueryOption("$filter", "Freight gt 30") _
.AddQueryOption("$orderby", "OrderID desc")
```

You can use the `$orderby` query option to both order and filter a query based on a single property, as in the following example that filters and orders the returned `Orders` objects based on the value of the `Freight` property:

```
// Create the DataServiceContext using the service URI.
NorthwindEntities context = new NorthwindEntities(svcUri);

// Define a query for orders with a Freight value greater than 30
// that also orders the result by the Freight value, descending.
DataServiceQuery<Order> selectedOrders = context.Orders
    .AddQueryOption("$orderby", "Freight gt 30 desc");

try
{
    // Enumerate over the results of the query.
    foreach (Order order in selectedOrders)
    {
        Console.WriteLine("Order ID: {0} - Freight: {1}",
            order.OrderID, order.Freight);
    }
}
catch (DataServiceQueryException ex)
{
    throw new ApplicationException(
        "An error occurred during query execution.", ex);
}
```

```
' Create the DataServiceContext using the service URI.
Dim context = New NorthwindEntities(svcUri)

' Define a query for orders with a Freight value greater than 30
' that also orders the result by the Freight value, descending.
Dim selectedOrders As DataServiceQuery(Of Order) = _
context.Orders.AddQueryOption("$orderby", "Freight gt 30 desc")

Try
    ' Enumerate over the results of the query.
    For Each order As Order In selectedOrders
        Console.WriteLine("Order ID: {0} - Freight: {1}", _
                order.OrderID, order.Freight)
    Next
Catch ex As DataServiceQueryException
    Throw New ApplicationException( _
            "An error occurred during query execution.", ex)
End Try
```

You can call the AddQueryOption method consecutively to construct complex query expressions. For more information, see How to: Add Query Options to a Data Service Query.

Query options give you another way to express the syntactic components of a LINQ query. For more information, see LINQ Considerations.

NOTE

The `$select` query option cannot be added to a query URI by using the AddQueryOption method. We recommend that you use the LINQ Select method to have the client generate the `$select` query option in the request URI.

Client versus Server Execution

The client executes a query in two parts. Whenever possible, expressions in a query are first evaluated on the client, and then a URI-based query is generated and sent to the data service for evaluation against data in the service. Consider the following LINQ query:

```
int basePrice = 100;
decimal discount = .10M;

// Define a query that returns products based on a
// calculation that is determined on the client.
var productsQuery = from p in context.Products
                  where p.UnitPrice >
                  (basePrice - (basePrice * discount)) &&
                  p.ProductName.Contains("bike")
                  select p;
```

```
Dim basePrice As Integer = 100
Dim discount As Decimal = Convert.ToDecimal(0.1)

' Define a query that returns products based on a
' calculation that is determined on the client.
Dim productsQuery = From p In context.Products
                  Where p.UnitPrice >
                  (basePrice - (basePrice * discount)) AndAlso
                  p.ProductName.Contains("bike")
                  Select p
```

In this example, the expression `(basePrice – (basePrice * discount))` is evaluated on the client. Because of this, the actual query URI

```
http://localhost:12345/northwind.svc/Products()?$filter=(UnitPrice gt 90.00M) and
substringof('bike',ProductName)
```

that is sent to the data service contains the already calculated decimal value of `90` in the filter clause. The other parts of the filtering expression, including the substring expression, are evaluated by the data service. Expressions that are evaluated on the client follow common language runtime (CLR) semantics, while expressions sent to the data service rely on the data service implementation of the OData Protocol. You should also be aware of scenarios where this separate evaluation may cause unexpected results, such as when both the client and service perform time-based evaluations in different time zones.

Query Responses

When executed, the DataServiceQuery<TElement> returns an IEnumerable<T> of the requested entity type. This query result can be cast to a QueryOperationResponse<T> object, as in the following example:

```
// Execute the query for all customers and get the response object.
QueryOperationResponse<Customer> response =
    query.Execute() as QueryOperationResponse<Customer>;
```

```
' Execute the query for all customers and get the response object.
Dim response As QueryOperationResponse(Of Customer) = _
    CType(query.Execute(), QueryOperationResponse(Of Customer))
```

The entity type instances that represent entities in the data service are created on the client by a process called object materialization. For more information, see Object Materialization. The QueryOperationResponse<T> object implements IEnumerable<T> to provide access to the results of the query.

The QueryOperationResponse<T> also has the following members that enable you to access additional information about a query result:

- Error - gets an error thrown by the operation, if any has occurred.
- Headers - contains the collection of HTTP response headers associated with the query response.

- Query - gets the original DataServiceQuery<TElement> that generated the QueryOperationResponse<T>.
- StatusCode - gets the HTTP response code for the query response.
- TotalCount - gets the total number of entities in the entity set when the IncludeTotalCount method was called on the DataServiceQuery<TElement>.
- GetContinuation - returns a DataServiceQueryContinuation object that contains the URI of the next page of results.

By default, WCF Data Services only returns data that is explicitly selected by the query URI. This gives you the option to explicitly load additional data from the data service when it is needed. A request is sent to the data service each time you explicitly load data from the data service. Data that can be explicitly loaded includes related entities, paged response data, and binary data streams.

NOTE

Because a data service may return a paged response, we recommend that your application use the programming pattern to handle a paged data service response. For more information, see Loading Deferred Content.

The amount of data returned by a query can also be reduced by specifying that only certain properties of an entity are returned in the response. For more information, see Query Projections.

Getting a Count of the Total Number of Entities in the Set

In some scenarios, it is helpful to know the total number of entities in an entity set and not merely the number returned by the query. Call the IncludeTotalCount method on the DataServiceQuery<TElement> to request that this total count of entities in the set be included with the query result. In this case, the TotalCount property of the returned QueryOperationResponse<T> returns the total number of entities in the set.

You can also get only the total count of entities in the set either as an Int32 or as a Int64 value by calling the Count or LongCount methods respectively. When these methods are called, a QueryOperationResponse<T> is not returned; only the count value is returned. For more information, see How to: Determine the Number of Entities Returned by a Query.

In This Section

Query Projections

Object Materialization

LINQ Considerations

How to: Execute Data Service Queries

How to: Add Query Options to a Data Service Query

How to: Determine the Number of Entities Returned by a Query

How to: Specify Client Credentials for a Data Service Request

How to: Set Headers in the Client Request

How to: Project Query Results

See Also

Query Projections (WCF Data Services)

8/31/2018 • 6 minutes to read • Edit Online

Projection provides a mechanism in the Open Data Protocol (OData) to reduce the amount of data in the feed returned by a query by specifying that only certain properties of an entity are returned in the response. For more information, see OData: Select System Query Option ($select).

This topic describes how to define a query projection, what the requirements are for entity and non-entity types, making updates to projected results, creating projected types, and lists some projection considerations.

Defining a Query Projection

You can add a projection clause to a query either by using the `$select` query option in a URI or by using the select clause (Select in Visual Basic) in a LINQ query. Returned entity data can be projected into either entity types or non-entity types on the client. Examples in this topic demonstrate how to use the `select` clause in a LINQ query.

IMPORTANT

Data loss might occur in the data service when you save updates that were made to projected types. For more information, see Projection Considerations.

Requirements for Entity and Non-Entity Types

Entity types must have one or more identity properties that make up the entity key. Entity types are defined on clients in one of the following ways:

- By applying the DataServiceKeyAttribute or DataServiceEntityAttribute to the type.
- When the type has a property named `ID`.
- When the type has a property named *type* `ID`, where *type* is the name of the type.

By default, when you project query results into a type defined at the client, the properties requested in the projection must exist in the client type. However, when you specify a value of `true` for the IgnoreMissingProperties property of the DataServiceContext, properties specified in the projection are not required to occur in the client type.

Making Updates to Projected Results

When you project query results into entity types on the client, the DataServiceContext can track those objects with updates to be sent back to the data service when the SaveChanges method is called. However, updates that are made to data projected into non-entity types on the client cannot be sent back to the data service. This is because without a key to identify the entity instance, the data service cannot update the correct entity in the data source. Non-entity types are not attached to the DataServiceContext.

When one or more properties of an entity type defined in the data service do not occur in the client type into which the entity is projected, inserts of new entities will not contain these missing properties. In this case, updates that are made to existing entities will **also** not include these missing properties. When a value exists for such a property, the update resets it to the default value for the property, as defined in the data source.

Creating Projected Types

The following example uses an anonymous LINQ query that projects the address-related properties of the `Customers` type into a new `CustomerAddress` type, which is defined on the client and is attributed as an entity type:

```
// Define an anonymous LINQ query that projects the Customers type into
// a CustomerAddress type that contains only address properties.
var query = from c in context.Customers
            where c.Country == "Germany"
            select new CustomerAddress {
                CustomerID = c.CustomerID,
                Address = c.Address,
                City = c.City,
                Region = c.Region,
                PostalCode = c.PostalCode,
                Country = c.Country};
```

```
' Define an anonymous LINQ query that projects the Customers type into
' a CustomerAddress type that contains only address properties.
Dim query = From c In context.Customers _
                Where c.Country = "Germany" _
                Select New CustomerAddress With { _
                    .CustomerID = c.CustomerID, _
                    .Address = c.Address, _
                    .City = c.City, _
                    .Region = c.Region, _
                    .PostalCode = c.PostalCode, _
                    .Country = c.Country}
```

In this example, the object initializer pattern is used to create a new instance of the `CustmerAddress` type instead of calling a constructor. Constructors are not supported when projecting into entity types, but they can be used when projecting into non-entity and anonymous types. Because `CustomerAddress` is an entity type, changes can be made and sent back to the data service.

Also, the data from the `Customer` type is projected into an instance of the `CustomerAddress` entity type instead of an anonymous type. Projection into anonymous types is supported, but the data is read-only because anonymous types are treated as non-entity types.

The MergeOption settings of the DataServiceContext are used for identity resolution during query projection. This means that if an instance of the `Customer` type already exists in the DataServiceContext, an instance of `CustomerAddress` with the same identity will follow the identity resolution rules set by the MergeOption

The following describes the behaviors when projecting results into entity and non-entity types:

Creating a new projected instance by using initializers

- Example:

  ```
  var query = from c in context.Customers
              where c.Country == "Germany"
              select new CustomerAddress {
                  CustomerID = c.CustomerID,
                  Address = c.Address,
                  City = c.City,
                  Region = c.Region,
                  PostalCode = c.PostalCode,
                  Country = c.Country};
  ```

```
Dim query = From c In context.Customers _
                Where c.Country = "Germany" _
                Select New CustomerAddress With { _
                    .CustomerID = c.CustomerID, _
                    .Address = c.Address, _
                    .City = c.City, _
                    .Region = c.Region, _
                    .PostalCode = c.PostalCode, _
                    .Country = c.Country}
```

- Entity type: Supported
- Non-entity type: Supported

Creating a new projected instance by using constructors

- Example:

```
var query = from c in context.Customers
            where c.Country == "Germany"
            select new CustomerAddress(
            c.CustomerID,
            c.Address,
            c.City,
            c.Region,
            c.PostalCode,
            c.Country);
```

```
Dim query = From c In context.Customers _
                Where c.Country = "Germany" _
                Select New CustomerAddress( _
                c.CustomerID, _
                c.Address, _
                c.City, _
                c.Region, _
                c.PostalCode, _
                c.Country)
```

- Entity type: A NotSupportedException is raised.
- Non-entity type: Supported

Using projection to transform a property value

- Example:

```
var query = from c in context.Customers
            where c.Country == "Germany"
            select new CustomerAddress
            {
                CustomerID = c.CustomerID,
                Address = "Full address:" + c.Address + ", " +
                c.City + ", " + c.Region + " " + c.PostalCode,
                City = c.City,
                Region = c.Region,
                PostalCode = c.PostalCode,
                Country = c.Country
            };
```

```
Dim query = From c In context.Customers _
                Where c.Country = "Germany" _
                Select New CustomerAddress With _
                {.CustomerID = c.CustomerID, _
                    .Address = "Full address: " & c.Address & ", " & _
                    c.City & "," & c.Region & " " & c.PostalCode, _
                    .City = c.City, _
                    .Region = c.Region, _
                    .PostalCode = c.PostalCode, _
                    .Country = c.Country}
```

- Entity type: This transformation is not supported for entity types because it can lead to confusion and potentially overwriting the data in the data source that belongs to another entity. A NotSupportedException is raised.

- Non-entity type: Supported

Projection Considerations

The following additional considerations apply when defining a query projection.

- When you define custom feeds for the Atom format, you must make sure that all entity properties that have custom mappings defined are included in the projection. When a mapped entity property is not included in the projection, data loss might occur. For more information, see Feed Customization.

- When inserts are made to a projected type that does not contain all of the properties of the entity in the data model of the data service, the properties not included in the projection at the client are set to their default values.

- When updates are made to a projected type that does not contain all of the properties of the entity in the data model of the data service, existing values not included in the projection on the client will be overwritten with uninitialized default values.

- When a projection includes a complex property, the entire complex object must be returned.

- When a projection includes a navigation property, the related objects are loaded implicitly without having to call the Expand method. The Expand method is not supported for use in a projected query.

- Query projections queries on the client are translated to use the `$select` query option in the request URI. When a query with projection is executed against a previous version of WCF Data Services that does not support the `$select` query option, an error is returned. This can also happen when the MaxProtocolVersion of the DataServiceBehavior for the data service is set to a value of V1. For more information, see Data Service Versioning.

For more information, see How to: Project Query Results.

See Also

Querying the Data Service

How to: Project Query Results (WCF Data Services)

7/18/2018 • 6 minutes to read • Edit Online

Projection provides a mechanism to reduce the amount of data returned by a query by specifying that only certain properties of an entity are returned in the response. You can perform projections on the results of an WCF Data Services query either by using the `$select` query option or by using the select clause (Select in Visual Basic) in a LINQ query. For more information, see Querying the Data Service.

The example in this topic uses the Northwind sample data service and autogenerated client data service classes. This service and the client data classes are created when you complete the WCF Data Services quickstart.

Example

The following example shows a LINQ query that projects Customers entities into a new CustomerAddress type, which contains only address-specific properties plus the identity property. This `CustomerAddress` class is defined on the client and is attributed so that the client library can recognize it as an entity type.

```
// Create the DataServiceContext using the service URI.
NorthwindEntities context = new NorthwindEntities(svcUri);

// Define an anonymous LINQ query that projects the Customers type into
// a CustomerAddress type that contains only address properties.
var query = from c in context.Customers
            where c.Country == "Germany"
            select new CustomerAddress {
                CustomerID = c.CustomerID,
                Address = c.Address,
                City = c.City,
                Region = c.Region,
                PostalCode = c.PostalCode,
                Country = c.Country};

try
{
    // Enumerate over the query result, which is executed implicitly.
    foreach (var item in query)
    {
        // Modify the address and mark the object as updated.
        item.Address += " #101";
        context.UpdateObject(item);

        // Write out the current values.
        Console.WriteLine("Customer ID: {0} \r\nStreet: {1} "
            + "\r\nCity: {2} \r\nState: {3} \r\nZip Code: {4} \r\nCountry: {5}",
            item.CustomerID, item.Address, item.City, item.Region,
            item.PostalCode, item.Country);
    }

    // Save changes to the data service.
    context.SaveChanges();
}
catch (DataServiceQueryException ex)
{
    throw new ApplicationException(
        "An error occurred during query execution.", ex);
}
```

```
' Create the DataServiceContext using the service URI.
Dim context = New NorthwindEntities(svcUri)

' Define an anonymous LINQ query that projects the Customers type into
' a CustomerAddress type that contains only address properties.
Dim query = From c In context.Customers _
                Where c.Country = "Germany" _
                Select New CustomerAddress With { _
                    .CustomerID = c.CustomerID, _
                    .Address = c.Address, _
                    .City = c.City, _
                    .Region = c.Region, _
                    .PostalCode = c.PostalCode, _
                    .Country = c.Country}

Try
    ' Enumerate over the query result, which is executed implicitly.
    For Each item In query
        ' Modify the address and mark the object as updated.
        item.Address += " #101"
        context.UpdateObject(item)

        ' Write out the current values.
        Console.WriteLine("Customer ID: {0} " & vbCrLf & "Street: {1} " _
                & vbCrLf & "City: {2} " & vbCrLf & "State: {3} " & vbCrLf & "Zip Code: {4}" _
                & vbCrLf & "Country: {5}", _
                item.CustomerID, item.Address, item.City, item.Region, _
                item.PostalCode, item.Country)
    Next

    ' Save changes to the data service.
    context.SaveChanges()
Catch ex As DataServiceQueryException
    Throw New ApplicationException( _
            "An error occurred during query execution.", ex)
End Try
```

Example

The following example shows a LINQ query that projects returned Customers entities into a new CustomerAddressNonEntity type, which contains only address-specific properties and no identity property. This `CustomerAddressNonEntity` class is defined on the client and is not attributed as an entity type.

```
// Create the DataServiceContext using the service URI.
NorthwindEntities context = new NorthwindEntities(svcUri);

// Define an anonymous LINQ query that projects the Customers type into
// a CustomerAddress type that contains only address properties.
var query = from c in context.Customers
            where c.Country == "Germany"
            select new CustomerAddressNonEntity
            {
                CompanyName = c.CompanyName,
                Address = c.Address,
                City = c.City,
                Region = c.Region,
                PostalCode = c.PostalCode,
                Country = c.Country
            };

try
{
    // Enumerate over the query result, which is executed implicitly.
    foreach (var item in query)
    {
        item.Address += "Street";

        Console.WriteLine("Company name: {0} \nStreet: {1} "
            + "\nCity: {2} \nState: {3} \nZip Code: {4} \nCountry: {5}",
            item.CompanyName, item.Address, item.City, item.Region,
            item.PostalCode, item.Country);
    }
}
catch (DataServiceQueryException ex)
{
    throw new ApplicationException(
        "An error occurred during query execution.", ex);
}
```

```
' Create the DataServiceContext using the service URI.
Dim context = New NorthwindEntities(svcUri)

' Define an anonymous LINQ query that projects the Customers type into
' a CustomerAddress type that contains only address properties.
Dim query = From c In context.Customers _
                Where c.Country = "Germany" _
                Select New CustomerAddressNonEntity With _
                {.CompanyName = c.CompanyName, _
                    .Address = c.Address, _
                    .City = c.City, _
                    .Region = c.Region, _
                    .PostalCode = c.PostalCode, _
                    .Country = c.Country}

Try
    ' Enumerate over the query result, which is executed implicitly.
    For Each item In query
        item.Address += "Street"

        Console.WriteLine("Company name: {0} " & vbNewLine & "Street: {1} " _
            & "" & vbNewLine & "City: {2} " & vbNewLine & "State: {3} " & vbNewLine & _
            "Zip Code: {4} " & vbNewLine & "Country: {5}", _
            item.CompanyName, item.Address, item.City, item.Region, _
            item.PostalCode, item.Country)
    Next
Catch ex As DataServiceQueryException
    Throw New ApplicationException( _
            "An error occurred during query execution.", ex)
End Try
```

Example

The following example shows the definitions of the `CustomerAddress` and `CustomerAddressNonEntity` types that are used in the previous examples.

```
[DataServiceKey("CustomerID")]
public partial class CustomerAddress
{
    private string _customerID;
    private string _address;
    private string _city;
    private string _region;
    private string _postalCode;
    private string _country;

    public string CustomerID
    {
        get
        {
            return this._customerID;
        }

        set
        {
            this._customerID = value;
        }
    }
    public string Address
    {
        get
        {
            return this._address;
        }
        set
        set
        {
            this._address = value;
        }
    }
    public string City
    {
        get
        {
            return this._city;
        }
        set
        {
            this._city = value;
        }
    }
    public string Region
    {
        get
        {
            return this._region;
        }
        set
        {
            this._region = value;
        }
    }
    public string PostalCode
    {
        get
        {
            return this._postalCode;
        }
        set
        {
            this._postalCode = value;
        }
    }
    public string Country
    {
        get
        {
            return this._country;
        }
        set
        {
            this._country = value;
        }
    }
}

public class CustomerAddressNonEntity
{
    private string _companyName;
    private string _address;
    private string _city;
    private string _region;
    private string _postalCode;
    private string _country;

    public string CompanyName
    {
        get
        {
            return this._companyName;
        }
        set
        {
            this._companyName = value;
        }
        }
    }
    public string Address
    {
        get
        {
            return this._address;
        }
        set
        {
            this._address = value;
        }
    }
    public string City
    {
        get
        {
            return this._city;
        }
        set
        {
            this._city = value;
        }
    }
    public string Region
    {
        get
        {
            return this._region;
        }
        set
        {
            this._region = value;
        }
    }
    public string PostalCode
    {
        get
        {
            return this._postalCode;
        }
        set
        {
            this._postalCode = value;
        }
    }
    public string Country
    {
        get
        {
            return this._country;
        }
        set
        {
            this._country = value;
        }
    }
}
```

```
<DataServiceKey("CustomerID")> _
Partial Public Class CustomerAddress
    Private _customerID As String
    Private _address As String
    Private _city As String
    Private _region As String
    Private _postalCode As String
    Private _country As String
    Public Property CustomerID() As String
        Get
            Return Me._customerID
        End Get
        Set(ByVal value As String)
            Me._customerID = Value
        End Set
    End Property
    Public Property Address() As String
        Get
            Return Me._address
        End Get
        Set(ByVal value As String)
            Me._address = Value
        End Set
    End Property
    Public Property City() As String
        Get
            Return Me._city
        End Get
        Set(ByVal value As String)
            Me._city = Value
        End Set
    End Property
    Public Property Region() As String
        Get
            Return Me._region
        End Get
        Set(ByVal value As String)
            Me._region = Value
        End Set
    End Property
    Public Property PostalCode() As String
        Get
            Return Me._postalCode
        End Get
        Set(ByVal value As String)
            Me._postalCode = Value
        End Set
    End Property
    Public Property Country() As String
        Get
            Return Me._country
        End Get
        Set(ByVal value As String)
            Me._country = value
        End Set
    End Property
End Class
Public Class CustomerAddressNonEntity
    Private _companyName As String
    Private _address As String
    Private _city As String
    Private _region As String
    Private _postalCode As String
    Private _country As String

    Public Property CompanyName() As String
        Get
            Return Me._companyName
        End Get
        Set(ByVal value As String)
            Me._companyName = value
        End Set
    End Property
    Public Property Address() As String
        Get
            Return Me._address
        End Get
        Set(ByVal value As String)
            Me._address = Value
        End Set
    End Property
    Public Property City() As String
        Get
            Return Me._city
        End Get
        Set(ByVal value As String)
            Me._city = Value
        End Set
    End Property
    Public Property Region() As String
        Get
            Return Me._region
        End Get
        Set(ByVal value As String)
            Me._region = Value
        End Set
    End Property
    Public Property PostalCode() As String
        Get
            Return Me._postalCode
        End Get
        Set(ByVal value As String)
            Me._postalCode = Value
        End Set
    End Property
    Public Property Country() As String
        Get
            Return Me._country
        End Get
        Set(ByVal value As String)
            Me._country = value
        End Set
    End Property
End Class
```

LINQ Considerations (WCF Data Services)

8/31/2018 • 8 minutes to read • Edit Online

This topic provides information about the way in which LINQ queries are composed and executed when you are using the WCF Data Services client and limitations of using LINQ to query a data service that implements the Open Data Protocol (OData). For more information about composing and executing queries against an OData-based data service, see Querying the Data Service.

Composing LINQ Queries

LINQ enables you to compose queries against a collection of objects that implements IEnumerable<T>. Both the **Add Service Reference** dialog box in Visual Studio and the DataSvcUtil.exe tool are used to generate a representation of an OData service as an entity container class that inherits from DataServiceContext, as well as objects that represent the entities returned in feeds. These tools also generate properties on the entity container class for the collections that are exposed as feeds by the service. Each of these properties of the class that encapsulates the data service return a DataServiceQuery<TElement>. Because the DataServiceQuery<TElement> class implements the IQueryable<T> interface defined by LINQ, you can compose a LINQ query against feeds exposed by the data service, which are translated by the client library into a query request URI that is sent to the data service on execution.

IMPORTANT

The set of queries expressible in the LINQ syntax is broader than those enabled in the URI syntax that is used by OData data services. A NotSupportedException is raised when the query cannot be mapped to a URI in the target data service. For more information, see the Unsupported LINQ Methods in this topic.

The following example is a LINQ query that returns `Orders` that have a freight cost of more than $30 and sorts the results by the shipping date, starting with the latest ship date:

```
var selectedOrders = from o in context.Orders
                     where o.Freight > 30
                     orderby o.ShippedDate descending
                     select o;
```

```
Dim selectedOrders = From o In context.Orders _
        Where (o.Freight > 30) _
        Order By o.ShippedDate Descending _
        Select o
```

This LINQ query is translated into the following query URI that is executed against the Northwind-based quickstart data service:

```
http://localhost:12345/Northwind.svc/Orders?Orderby=ShippedDate&?filter=Freight gt 30
```

For more general information about LINQ, see LINQ (Language-Integrated Query).

LINQ enables you to compose queries by using both the language-specific declarative query syntax, shown in the previous example, as well as a set of query methods known as standard query operators. An equivalent query to the previous example can be composed by using only the method-based syntax, as shown the following example:

```
var selectedOrders = context.Orders
                    .Where(o => o.Freight > 30)
                    .OrderByDescending(o => o.ShippedDate);
```

```
Dim selectedOrders = context.Orders _
                     .Where(Function(o) o.Freight.Value > 30) _
                     .OrderByDescending(Function(o) o.ShippedDate)
```

The WCF Data Services client is able to translate both kinds of composed queries into a query URI, and you can extend a LINQ query by appending query methods to a query expression. When you compose LINQ queries by appending method syntax to a query expression or a DataServiceQuery<TElement>, the operations are added to the query URI in the order in which methods are called. This is equivalent to calling the AddQueryOption method to add each query option to the query URI.

Executing LINQ Queries

Certain LINQ query methods, such as First<TSource>(IEnumerable<TSource>) or Single<TSource>(IEnumerable<TSource>), when appended to the query, cause the query to be executed. A query is also executed when results are enumerated implicitly, such as during a `foreach` loop or when the query is assigned to a `List` collection. For more information, see Querying the Data Service.

The client executes a LINQ query in two parts. Whenever possible, LINQ expressions in a query are first evaluated on the client, and then a URI-based query is generated and sent to the data service for evaluation against data in the service. For more information, see the section Client versus Server Execution in Querying the Data Service.

When a LINQ query cannot be translated in an OData-compliant query URI, an exception is raised when execution is attempted. For more information, see Querying the Data Service.

LINQ Query Examples

The examples in the following sections demonstrate the kinds of LINQ queries that can be executed against an OData service.

Filtering

The LINQ query examples in this section filter data in the feed returned by the service.

The following examples are equivalent queries that filter the returned `Orders` entities so that only orders with a freight cost greater than $30 are returned:

- Using LINQ query syntax:

```
var filteredOrders = from o in context.Orders
                        where o.Freight > 30
                        select o;
```

```
Dim filteredOrders = From o In context.Orders
                        Where o.Freight.Value > 30
                        Select o
```

- Using LINQ query methods:

```
var filteredOrders = context.Orders
    .Where(o => o.Freight > 30);
```

```
Dim filteredOrders = context.Orders.Where(Function(o) o.Freight.Value > 0)
```

- The URI query string `$filter` option:

```
// Define a query for orders with a Freight value greater than 30.
var filteredOrders
    = context.Orders.AddQueryOption("$filter", "Freight gt 30M");
```

```
' Define a query for orders with a Freight value greater than 30.
Dim filteredOrders _
           = context.Orders.AddQueryOption("$filter", "Freight gt 30M")
```

All of the previous examples are translated to the query URI: `http://localhost:12345/northwind.svc/Orders()?$filter=Freight gt 30M`.

Sorting

The following examples show equivalent queries that sort returned data both by the company name and by postal code, descending:

- Using LINQ query syntax:

```
var sortedCustomers = from c in context.Customers
                     orderby c.CompanyName ascending,
                     c.PostalCode descending
                     select c;
```

```
Dim sortedCustomers = From c In context.Customers
                             Order By c.CompanyName Ascending,
                             c.PostalCode Descending
                             Select c
```

- Using LINQ query methods:

```
var sortedCustomers = context.Customers.OrderBy(c => c.CompanyName)
    .ThenByDescending(c => c.PostalCode);
```

```
Dim sortedCustomers = context.Customers.OrderBy(Function(c) c.CompanyName) _
.ThenByDescending(Function(c) c.PostalCode)
```

- URI query string `$orderby` option):

```
var sortedCustomers = context.Customers
    .AddQueryOption("$orderby", "CompanyName, PostalCode desc");
```

```
Dim sortedCustomers = context.Customers _
                      .AddQueryOption("$orderby", "CompanyName, PostalCode desc")
```

All of the previous examples are translated to the query URI:
`http://localhost:12345/northwind.svc/Customers()?$orderby=CompanyName,PostalCode desc`.

Projection

The following examples show equivalent queries that project returned data into the narrower `CustomerAddress` type:

- Using LINQ query syntax:

```
var projectedQuery = from c in context.Customers
            select new CustomerAddress
            {
                CustomerID = c.CustomerID,
                Address = c.Address,
                City = c.City,
                Region = c.Region,
                PostalCode = c.PostalCode,
                Country = c.Country
            };
```

```
Dim projectedQuery = From c In context.Customers
                     Select New CustomerAddress With
                    {
                        .CustomerID = c.CustomerID,
                        .Address = c.Address,
                        .City = c.City,
                        .Region = c.Region,
                        .PostalCode = c.PostalCode,
                        .Country = c.Country
                    }
```

- Using LINQ query methods:

```
var projectedQuery = context.Customers.Where(c => c.Country == "Germany")
    .Select(c => new CustomerAddress
    {
        CustomerID = c.CustomerID,
        Address = c.Address,
        City = c.City,
        Region = c.Region,
        PostalCode = c.PostalCode,
        Country = c.Country});
```

```
Dim projectedQuery = context.Customers.Where(Function(c) c.Country = "Germany") _
            .Select(Function(c) New CustomerAddress With
            {
                .CustomerID = c.CustomerID,
                .Address = c.Address,
                .City = c.City,
                .Region = c.Region,
                .PostalCode = c.PostalCode,
                .Country = c.Country
            })
```

NOTE

The `$select` query option cannot be added to a query URI by using the AddQueryOption method. We recommend that you use the LINQ Select method to have the client generate the `$select` query option in the request URI.

Both of the previous examples are translated to the query URI:
`"http://localhost:12345/northwind.svc/Customers()?$filter=Country eq 'GerGerm'&$select=CustomerID,Address,City,Region,PostalCode,Country"`
.

Client Paging

The following examples show equivalent queries that request a page of sorted order entities that includes 25 orders, skipping the first 50 orders:

- Applying query methods to a LINQ query:

```
var pagedOrders = (from o in context.Orders
                      orderby o.OrderDate descending
                     select o).Skip(50).Take(25);
```

```
Dim pagedOrders = (From o In context.Orders
                   Order By o.OrderDate Descending
                   Select o) _
               .Skip(50).Take(25)
```

- URI query string `$skip` and `$top` options):

```
var pagedOrders = context.Orders
    .AddQueryOption("$orderby", "OrderDate desc")
    .AddQueryOption("$skip", 50)
    .AddQueryOption("$top", 25);
```

```
Dim pagedOrders = context.Orders _
                  .AddQueryOption("$orderby", "OrderDate desc") _
                  .AddQueryOption("$skip", 50) _
                  .AddQueryOption("$top", 25) _
```

Both of the previous examples are translated to the query URI:
`http://localhost:12345/northwind.svc/Orders()?$orderby=OrderDate desc&$skip=50&$top=25`.

Expand

When you query an OData data service, you can request that entities related to the entity targeted by the query be included the returned feed. The Expand method is called on the DataServiceQuery<TElement> for the entity set targeted by the LINQ query, with the related entity set name supplied as the `path` parameter. For more information, see Loading Deferred Content.

The following examples show equivalent ways to use the Expand method in a query:

- In LINQ query syntax:

```
var ordersQuery = from o in context.Orders.Expand("Order_Details")
                     where o.CustomerID == "ALFKI"
                     select o;
```

```
Dim ordersQuery = From o In context.Orders.Expand("Order_Details")
                     Where o.CustomerID = "ALFKI"
                     Select o
```

- With LINQ query methods:

```
var ordersQuery = context.Orders.Expand("Order_Details")
                  .Where(o => o.CustomerID == "ALFKI");
```

```
Dim ordersQuery = context.Orders.Expand("Order_Details") _
                          .Where(Function(o) o.CustomerID = "ALFKI")
```

Both of the previous examples are translated to the query URI: `http://localhost:12345/northwind.svc/Orders()?$filter=CustomerID eq 'ALFKI'&$expand=Order_Details`.

Unsupported LINQ Methods

The following table contains the classes of LINQ methods are not supported and cannot be included in a query executed against an OData service:

OPERATION TYPE	UNSUPPORTED METHOD
Set operators	All set operators are unsupported against a DataServiceQuery<TElement>, which included the following: - All - Any - Concat - DefaultIfEmpty - Distinct - Except - Intersect - Union - Zip
Ordering operators	The following ordering operators that require IComparer<T> are unsupported against a DataServiceQuery<TElement>: - OrderBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IComparer<TKey>) - OrderByDescending<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IComparer<TKey>) - ThenBy<TSource,TKey>(IOrderedEnumerable<TSource>, Func<TSource,TKey>, IComparer<TKey>) - ThenByDescending<TSource,TKey>(IOrderedEnumerable<TSource>, Func<TSource,TKey>, IComparer<TKey>)

OPERATION TYPE	UNSUPPORTED METHOD
Projection and filtering operators	The following projection and filtering operators that accept a positional argument are unsupported against a DataServiceQuery<TElement>: - Join<TOuter,TInner,TKey,TResult>(IEnumerable<TOuter>, IEnumerable<TInner>, Func<TOuter,TKey>, Func<TInner,TKey>, Func<TOuter,TInner,TResult>, IEqualityComparer<TKey>) - Select<TSource,TResult>(IEnumerable<TSource>, Func<TSource,Int32,TResult>) - SelectMany<TSource,TResult>(IEnumerable<TSource>, Func<TSource,IEnumerable<TResult>>) - SelectMany<TSource,TResult>(IEnumerable<TSource>, Func<TSource,Int32,IEnumerable<TResult>>) - SelectMany<TSource,TCollection,TResult>(IEnumerable<TSource>, Func<TSource,IEnumerable<TCollection>>, Func<TSource,TCollection,TResult>) - SelectMany<TSource,TCollection,TResult>(IEnumerable<TSource>, Func<TSource,Int32,IEnumerable<TCollection>>, Func<TSource,TCollection,TResult>) - Where<TSource>(IEnumerable<TSource>, Func<TSource,Int32,Boolean>)
Grouping operators	All grouping operators are unsupported against a DataServiceQuery<TElement>, including the following: - GroupBy - GroupJoin Grouping operations must be performed on the client.
Aggregate operators	All aggregate operations are unsupported against a DataServiceQuery<TElement>, including the following: - Aggregate - Average - Count - LongCount - Max - Min - Sum Aggregate operations must either be performed on the client or be encapsulated by a service operation.
Paging operators	The following paging operators are not supported against a DataServiceQuery<TElement>: - ElementAt - Last - LastOrDefault - SkipWhile - TakeWhile **Note:** Paging operators that are executed on an empty sequence return null.

OPERATION TYPE	UNSUPPORTED METHOD
Other operators	The following other operators are not supported against a DataServiceQuery<TElement>: 1. Empty 2. Range 3. Repeat 4. ToDictionary 5. ToLookup

Supported Expression Functions

The following common-language runtime (CLR) methods and properties are supported because they can be translated in a query expression for inclusion in the request URI to an OData service:

STRING MEMBER	SUPPORTED ODATA FUNCTION
Concat(String, String)	`string concat(string p0, string p1)`
Contains(String)	`bool substringof(string p0, string p1)`
EndsWith(String)	`bool endswith(string p0, string p1)`
IndexOf(String, Int32)	`int indexof(string p0, string p1)`
Length	`int length(string p0)`
Replace(String, String)	`string replace(string p0, string find, string replace)`
Substring(Int32)	`string substring(string p0, int pos)`
Substring(Int32, Int32)	`string substring(string p0, int pos, int length)`
ToLower()	`string tolower(string p0)`
ToUpper()	`string toupper(string p0)`
Trim()	`string trim(string p0)`

DATETIME MEMBER[1]	SUPPORTED ODATA FUNCTION
Day	`int day(DateTime p0)`
Hour	`int hour(DateTime p0)`
Minute	`int minute(DateTime p0)`
Month	`int month(DateTime p0)`
Second	`int second(DateTime p0)`

DATETIME MEMBER	SUPPORTED ODATA FUNCTION
Year	`int year(DateTime p0)`

[1]The equivalent date and time properties of Microsoft.VisualBasic.DateAndTime, as well as the DatePart method in Visual Basic are also supported.

MATH MEMBER	SUPPORTED ODATA FUNCTION
Ceiling(Decimal)	`decimal ceiling(decimal p0)`
Ceiling(Double)	`double ceiling(double p0)`
Floor(Decimal)	`decimal floor(decimal p0)`
Floor(Double)	`double floor(double p0)`
Round(Decimal)	`decimal round(decimal p0)`
Round(Double)	`double round(double p0)`

EXPRESSION MEMBER	SUPPORTED ODATA FUNCTION
TypeIs(Expression, Type)	`bool isof(type p0)`

The client may also be able to evaluate additional CLR functions on the client. A NotSupportedException is raised for any expression that cannot be evaluated on the client and cannot be translated into a valid request URI for evaluation on the server.

See Also

Querying the Data Service
Query Projections
Object Materialization
OData: URI Conventions

Object Materialization (WCF Data Services)

5/2/2018 • 3 minutes to read • Edit Online

When you use the **Add Service Reference** dialog to consume an Open Data Protocol (OData) feed in a .NET Framework-based client application, equivalent data classes are generated for each entity type in the data model exposed by the feed. For more information, see Generating the Data Service Client Library. Entity data that is returned by a query is materialized into an instance of one of these generated client data service classes. For information about merge options and identity resolution for tracked objects, see Managing the Data Service Context.

WCF Data Services also enables you to define your own client data service classes rather than using the tool-generated data classes. This enables you to use your own data classes, also known as "plain-old CLR object" (POCO) data classes. When using these types of custom data classes, you should attribute the data class with either DataServiceKeyAttribute or DataServiceEntityAttribute and ensure that type names on the client match type names in the data model of the data service.

After the library receives the query response message, it materializes the returned data from the OData feed into instances of client data service classes that are of the type of the query. The general process for materializing these objects is as follows:

1. The client library reads the serialized type from the `entry` element in the response message feed and attempts to create a new instance of the correct type, in one of the following ways:

   - When the type declared in the feed has the same name as the type of the DataServiceQuery<TElement>, a new instance of this type is created by using the empty constructor.

   - When the type declared in the feed has the same name as a type that is derived from the type of the DataServiceQuery<TElement>, a new instance of this derived type is created by using the empty constructor.

   - When the type declared in the feed cannot be matched to the type of the DataServiceQuery<TElement> or any derived types, a new instance of the queried type is created by using the empty constructor.

   - When the ResolveType property is set, the supplied delegate is called to override the default name-based type mapping and a new instance of the type returned by the Func<T,TResult> is created instead. If this delegate returns a null value, a new instance of the queried type is created instead. It may be required to override the default name-based type name mapping to support inheritance scenarios.

2. The client library reads the URI value from the `id` element of the `entry`, which is the identity value of the entity. Unless a MergeOption value of NoTracking is used, the identity value is used to track the object in the DataServiceContext. The identity value is also used to guarantee that only a single entity instance is created, even when an entity is returned multiple times in the query response.

3. The client library reads properties from the feed entry and set the corresponding properties on the newly created object. When an object that has the same identity value already occurs in the DataServiceContext, the properties are set based on the MergeOption setting of the DataServiceContext. The response might contain property values for which a corresponding property does not occur in the client type. When this occurs, the action depends on the value of the IgnoreMissingProperties property of the DataServiceContext. When this property is set to `true`, the missing property is ignored. Otherwise, an

error is raised. Properties are set as follows:

- Scalar properties are set to the corresponding value in the entry in the response message.
- Complex properties are set to a new complex type instance, which are set with the properties of the complex type from the response.
- Navigation properties that return a collection of related entities are set to a new or existing instance of ICollection<T>, where `T` is the type of the related entity. This collection is empty unless the related objects have been loaded into the DataServiceContext. For more information, see Loading Deferred Content.

  > **NOTE**
  >
  > When the generated client data classes support data binding, navigation properties return instances of the DataServiceCollection<T> class instead. For more information, see Binding Data to Controls.

4. The ReadingEntity event is raised.
5. The client library attaches the object to the DataServiceContext. The object is not attached when the MergeOption is NoTracking.

See Also

Querying the Data Service
Query Projections

How to: Execute Data Service Queries (WCF Data Services)

5/2/2018 • 3 minutes to read • Edit Online

WCF Data Services enables you to query a data service from a .NET Framework-based client application by using the generated client data service classes. You can execute queries by using one of these methods:

- Executing a LINQ query against the named DataServiceQuery<TElement> that you obtain from the DataServiceContext that the `Add Data Service Reference` tool generates.
- Implicitly, by enumerating over the named DataServiceQuery<TElement> that you obtain from the DataServiceContext that the `Add Data Service Reference` tool generates.
- Explicitly, by calling the Execute method on the DataServiceQuery<TElement>, or the BeginExecute method for asynchronous execution.

For more information, see Querying the Data Service.

The example in this topic uses the Northwind sample data service and autogenerated client data service classes. This service and the client data classes are created when you complete the WCF Data Services quickstart.

Example

The following example shows how to define and execute a LINQ query that returns all `Customers` against the Northwind data service.

```
// Create the DataServiceContext using the service URI.
NorthwindEntities context = new NorthwindEntities(svcUri);

try
{
    // Define a LINQ query that returns all customers.
    var allCustomers = from cust in context.Customers
                       select cust;

    // Enumerate over the query obtained from the context.
    foreach (Customer customer in allCustomers)
    {
        Console.WriteLine("Customer Name: {0}", customer.CompanyName);
    }
}
catch (DataServiceQueryException ex)
{
    throw new ApplicationException(
        "An error occurred during query execution.", ex);
}
```

```
' Create the DataServiceContext using the service URI.
Dim context = New NorthwindEntities(svcUri)

Try
    ' Define a LINQ query that returns all customers.
    Dim allCustomers = From cust In context.Customers _
                           Select cust

    ' Enumerate over the query obtained from the context.
    For Each customer As Customer In allCustomers
        Console.WriteLine("Customer Name: {0}", customer.CompanyName)
    Next
Catch ex As DataServiceQueryException
    Throw New ApplicationException( _
            "An error occurred during query execution.", ex)
End Try
```

Example

The following example shows how to use the context that the `Add Data Service Reference` tool generates to implicitly execute a query that returns all `Customers` against the Northwind data service. The URI of the requested `Customers` entity set is determined automatically by the context. The query is executed implicitly when the enumeration occurs.

```
// Create the DataServiceContext using the service URI.
NorthwindEntities context = new NorthwindEntities(svcUri);

// Define a new query for Customers.
DataServiceQuery<Customer> query = context.Customers;

try
{
    // Enumerate over the query result, which is executed implicitly.
    foreach (Customer customer in query)
    {
        Console.WriteLine("Customer Name: {0}", customer.CompanyName);
    }
}
catch (DataServiceQueryException ex)
{
    throw new ApplicationException(
        "An error occurred during query execution.", ex);
}
```

```
' Create the DataServiceContext using the service URI.
Dim context = New NorthwindEntities(svcUri)

' Define a new query for Customers.
Dim query As DataServiceQuery(Of Customer) = context.Customers

Try
    ' Enumerate over the query result, which is executed implicitly.
    For Each customer As Customer In query
        Console.WriteLine("Customer Name: {0}", customer.CompanyName)
    Next
Catch ex As DataServiceQueryException
    Throw New ApplicationException( _
            "An error occurred during query execution.", ex)
End Try
```

Example

The following example shows how to use the DataServiceContext to explicitly execute a query that returns all `Customers` against the Northwind data service.

```
// Define a request URI that returns Customers.
Uri customersUri = new Uri("Customers", UriKind.Relative);

// Create the DataServiceContext using the service URI.
DataServiceContext context = new DataServiceContext(svcUri);

try
{
    // Enumerate over the query result.
    foreach (Customer customer in context.Execute<Customer>(customersUri))
    {
        Console.WriteLine("Customer Name: {0}", customer.CompanyName);
    }
}
catch (DataServiceQueryException ex)
{
    throw new ApplicationException(
        "An error occurred during query execution.", ex);
}
```

```
' Define a request URI that returns Customers.
Dim customersUri = New Uri(svcUri, "Northwind.svc/Customers")

' Create the DataServiceContext using the service URI.
Dim context = New DataServiceContext(svcUri)

Try
    ' Enumerate over the query result.
    For Each customer As Customer In context.Execute(Of Customer)(customersUri)
        Console.WriteLine("Customer Name: {0}", customer.CompanyName)
    Next

Catch ex As DataServiceQueryException
    Throw New ApplicationException( _
            "An error occurred during query execution.", ex)
End Try
```

See Also

How to: Add Query Options to a Data Service Query

How to: Add Query Options to a Data Service Query (WCF Data Services)

5/2/2018 • 5 minutes to read • Edit Online

WCF Data Services enables you to query a data service from a .NET Framework-based client application by using the generated client data service classes. The easiest to do this is to compose a Language Integrated Query (LINQ) query expression that includes the desired query options. You can also call a series of LINQ query methods to compose an equivalent query. Finally, you can use the AddQueryOption method to add query options to a query. In each of these cases, the URI that is generated by the client includes the requested entity set with the selected query options applied. For more information, see Querying the Data Service.

The example in this topic uses the Northwind sample data service and autogenerated client data service classes. This service and the client data classes are created when you complete the WCF Data Services quickstart.

Example

The following example shows how to compose a LINQ query expression that returns only orders with a freight cost of more than $30 and that orders the results by the ship date in descending order.

```
// Create the DataServiceContext using the service URI.
NorthwindEntities context = new NorthwindEntities(svcUri);

// Define a query for orders with a Freight value greater than 30
// and that is ordered by the ship date, descending.
var selectedOrders = from o in context.Orders
                     where o.Freight > 30
                     orderby o.ShippedDate descending
                     select o;

try
{
    // Enumerate over the results of the query.
    foreach (Order order in selectedOrders)
    {
        Console.WriteLine("Order ID: {0} - Ship Date: {1} - Freight: {2}",
            order.OrderID, order.ShippedDate, order.Freight);
    }
}
catch (DataServiceQueryException ex)
{
    throw new ApplicationException(
        "An error occurred during query execution.", ex);
}
```

```
' Create the DataServiceContext using the service URI.
Dim context = New NorthwindEntities(svcUri)

' Define a query for orders with a Freight value greater than 30
' and that is ordered by the ship date, descending.
Dim selectedOrders = From o In context.Orders _
        Where (o.Freight > 30) _
        Order By o.ShippedDate Descending _
        Select o

Try
    ' Enumerate over the results of the query.
    For Each order As Order In selectedOrders
        Console.WriteLine("Order ID: {0} - Ship Date: {1} - Freight: {2}", _
                order.OrderID, order.ShippedDate, order.Freight)
    Next
Catch ex As DataServiceQueryException
    Throw New ApplicationException( _
            "An error occurred during query execution.", ex)
End Try
```

Example

The following example shows how to compose a LINQ query by using LINQ query methods that is equivalent to the previous query.

```
// Create the DataServiceContext using the service URI.
NorthwindEntities context = new NorthwindEntities(svcUri);

// Define a query for orders with a Freight value greater than 30
// and that is ordered by the ship date, descending.
var selectedOrders = context.Orders
                    .Where(o => o.Freight > 30)
                    .OrderByDescending(o => o.ShippedDate);

try
{
    // Enumerate over the results of the query.
    foreach (Order currentOrder in selectedOrders)
    {
        Console.WriteLine("Order ID: {0} - Ship Date: {1} - Freight: {2}",
            currentOrder.OrderID, currentOrder.ShippedDate,
            currentOrder.Freight);
    }
}
catch (DataServiceQueryException ex)
{
    throw new ApplicationException(
        "An error occurred during query execution.", ex);
}
```

```
' Create the DataServiceContext using the service URI.
Dim context = New NorthwindEntities(svcUri)

' Define a query for orders with a Freight value greater than 30
' and that is ordered by the ship date, descending.
Dim selectedOrders = context.Orders _
                     .Where(Function(o) o.Freight.Value > 30) _
                     .OrderByDescending(Function(o) o.ShippedDate)

Try
    ' Enumerate over the results of the query.
    For Each order As Order In selectedOrders

        Console.WriteLine("Order ID: {0} - Ship Date: {1} - Freight: {2}", _
                order.OrderID, order.ShippedDate, order.Freight)
    Next
Catch ex As DataServiceQueryException
    Throw New ApplicationException( _
        "An error occurred during query execution.", ex)
End Try
```

Example

The following example shows how to use to the AddQueryOption method to create a DataServiceQuery<TElement> that is equivalent to the previous examples.

```
// Create the DataServiceContext using the service URI.
NorthwindEntities context = new NorthwindEntities(svcUri);

// Define a query for orders with a Freight value greater than 30
// and that is ordered by the ship date, descending.
DataServiceQuery<Order> selectedOrders = context.Orders
    .AddQueryOption("$filter", "Freight gt 30")
    .AddQueryOption("$orderby", "OrderID desc");

try
{
    // Enumerate over the results of the query.
    foreach (Order order in selectedOrders)
    {
        Console.WriteLine("Order ID: {0} - Ship Date: {1} - Freight: {2}",
            order.OrderID, order.ShippedDate, order.Freight);
    }
}
catch (DataServiceQueryException ex)
{
    throw new ApplicationException(
        "An error occurred during query execution.", ex);
}
```

```
' Create the DataServiceContext using the service URI.
Dim context = New NorthwindEntities(svcUri)

' Define a query for orders with a Freight value greater than 30
' and that is ordered by the ship date, descending.
Dim selectedOrders As DataServiceQuery(Of Order) = context.Orders _
.AddQueryOption("$filter", "Freight gt 30") _
.AddQueryOption("$orderby", "OrderID desc")

Try
    ' Enumerate over the results of the query.
    For Each order As Order In selectedOrders
        Console.WriteLine("Order ID: {0} - Ship Date: {1} - Freight: {2}", _
                order.OrderID, order.ShippedDate, order.Freight)
    Next
Catch ex As DataServiceQueryException
    Throw New ApplicationException( _
            "An error occurred during query execution.", ex)
End Try
```

Example

The following example shows how to use the `$orderby` query option to both filter and order returned Orders objects by the Freight property.

```
// Create the DataServiceContext using the service URI.
NorthwindEntities context = new NorthwindEntities(svcUri);

// Define a query for orders with a Freight value greater than 30
// that also orders the result by the Freight value, descending.
DataServiceQuery<Order> selectedOrders = context.Orders
    .AddQueryOption("$orderby", "Freight gt 30 desc");

try
{
    // Enumerate over the results of the query.
    foreach (Order order in selectedOrders)
    {
        Console.WriteLine("Order ID: {0} - Freight: {1}",
            order.OrderID, order.Freight);
    }
}
catch (DataServiceQueryException ex)
{
    throw new ApplicationException(
        "An error occurred during query execution.", ex);
}
```

```
' Create the DataServiceContext using the service URI.
Dim context = New NorthwindEntities(svcUri)

' Define a query for orders with a Freight value greater than 30
' that also orders the result by the Freight value, descending.
Dim selectedOrders As DataServiceQuery(Of Order) = _
context.Orders.AddQueryOption("$orderby", "Freight gt 30 desc")

Try
    ' Enumerate over the results of the query.
    For Each order As Order In selectedOrders
        Console.WriteLine("Order ID: {0} - Freight: {1}", _
                order.OrderID, order.Freight)
    Next
Catch ex As DataServiceQueryException
    Throw New ApplicationException( _
            "An error occurred during query execution.", ex)
End Try
```

See Also

Querying the Data Service
How to: Project Query Results

How to: Determine the Number of Entities Returned by a Query (WCF Data Services)

5/2/2018 • 2 minutes to read • Edit Online

With WCF Data Services, you can determine the number of entities that are in the entity set specified by a query URI. This count can be included either along with the query result or as an integer value. For more information, see Querying the Data Service.

The example in this topic uses the Northwind sample data service and autogenerated client data service classes. This service and the client data classes are created when you complete the WCF Data Services quickstart.

Example

This example executes a query after calling the IncludeTotalCount method. The TotalCount property returns the number of entities in the `Customers` entity set.

```
// Create the DataServiceContext using the service URI.
NorthwindEntities context = new NorthwindEntities(svcUri);

// Define a new query for Customers that includes the total count.
DataServiceQuery<Customer> query = context.Customers.IncludeTotalCount();

try
{
    // Execute the query for all customers and get the response object.
    QueryOperationResponse<Customer> response =
        query.Execute() as QueryOperationResponse<Customer>;

    // Retrieve the total count from the response.
    Console.WriteLine("There are a total of {0} customers.", response.TotalCount);

    // Enumerate the customers in the response.
    foreach (Customer customer in response)
    {
        Console.WriteLine("\tCustomer Name: {0}", customer.CompanyName);
    }
}
catch (DataServiceQueryException ex)
{
    throw new ApplicationException(
        "An error occurred during query execution.", ex);
}
```

```
' Create the DataServiceContext using the service URI.
Dim context = New NorthwindEntities(svcUri)

' Define a new query for Customers that includes the total count.
Dim query As DataServiceQuery(Of Customer) = _
context.Customers.IncludeTotalCount()

Try
    ' Execute the query for all customers and get the response object.
    Dim response As QueryOperationResponse(Of Customer) = _
        CType(query.Execute(), QueryOperationResponse(Of Customer))

    ' Retrieve the total count from the response.
    Console.WriteLine("There are a total of {0} customers.", response.TotalCount)

    ' Enumerate the customers in the response.
    For Each customer As Customer In response
        Console.WriteLine(vbTab & "Customer Name: {0}", customer.CompanyName)
    Next
Catch ex As DataServiceQueryException
    Throw New ApplicationException( _
            "An error occurred during query execution.", ex)
End Try
```

Example

This example calls the Count method to return only an integer value that is the number of entities in the `Customers` entity set.

```
   // Create the DataServiceContext using the service URI.
   NorthwindEntities context = new NorthwindEntities(svcUri);

   // Define a new query for Customers.
   DataServiceQuery<Customer> query = context.Customers;

   try
   {
       // Execute the query to just return the value of all customers in the set.
       int totalCount = query.Count();

       // Retrieve the total count from the response.
       Console.WriteLine("There are {0} customers in total.", totalCount);
   }
   catch (DataServiceQueryException ex)
   {
       throw new ApplicationException(
           "An error occurred during query execution.", ex);
   }
```

```
' Create the DataServiceContext using the service URI.
Dim context = New NorthwindEntities(svcUri)

' Define a new query for Customers.
Dim query As DataServiceQuery(Of Customer) = context.Customers

Try
    ' Execute the query to just return the value of all customers in the set.
    Dim totalCount = query.Count()

    ' Retrieve the total count from the response.
    Console.WriteLine("There are {0} customers in total.", totalCount)
Catch ex As DataServiceQueryException
    Throw New ApplicationException( _
            "An error occurred during query execution.", ex)
End Try
```

See Also

Querying the Data Service

How to: Specify Client Credentials for a Data Service Request (WCF Data Services)

8/31/2018 • 6 minutes to read • Edit Online

By default, the client library does not supply credentials when sending a request to an OData service. However, you can specify that credentials be sent to authenticate requests to the data service by supplying a NetworkCredential for the Credentials property of the DataServiceContext. For more information, see Securing WCF Data Services. The example in this topic shows how to explicitly provide credentials that are used by the WCF Data Services client when requesting data from the data service.

The example in this topic uses the Northwind sample data service and autogenerated client data service classes. This service and the client data classes are created when you complete the WCF Data Services quickstart. You can also use the Northwind sample data service that is published on the OData Web site; this sample data service is read-only and attempting to save changes returns an error. The sample data services on the OData Web site allow anonymous authentication.

Example

The following example is from the code-behind page for an Extensible Application Markup Language (XAML) file that is the main page of the Windows Presentation Framework application. This example displays a `LoginWindow` instance to collect authentication credentials from the user, and then uses these credentials when making a request to the data service.

```
using System;
using System.Linq;
using System.Net;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Security;
using NorthwindClient.Northwind;
using System.Data.Services.Client;

namespace NorthwindClient
{
    public partial class ClientCredentials : Window
    {
        // Create the binding collections and the data service context.
        private DataServiceCollection<Customer> binding;
        NorthwindEntities context;
        CollectionViewSource customerAddressViewSource;

        // Instantiate the service URI and credentials.
        Uri serviceUri = new Uri("http://localhost:12345/Northwind.svc/");
        NetworkCredential credentials = new NetworkCredential();

        public ClientCredentials()
        {
            InitializeComponent();
        }

        private void ClientCredentials_Loaded(object sender, RoutedEventArgs e)
        {
            string userName = string.Empty;
            string domain = string.Empty;
            SecureString password = new SecureString();
            // Get credentials for authentication.
            LoginWindow login = new LoginWindow();
            login.ShowDialog();

            if (login.DialogResult == true
                && login.userNameBox.Text != string.Empty
                && login.passwordBox.SecurePassword.Length != 0)
            {
                // Instantiate the context.
                context =
                    new NorthwindEntities(serviceUri);

                // Get the user name and domain from the login.
                string[] qualifiedUserName = login.userNameBox.Text.Split(new char[] { '\\' });
                if (qualifiedUserName.Length == 2)
                {
                    domain = qualifiedUserName[0];
                    userName = qualifiedUserName[1];
                }
                else
                {
                    userName = login.userNameBox.Text;
                }
                password = login.passwordBox.SecurePassword;

                // Set the client authentication credentials.
                context.Credentials =
                    new NetworkCredential(userName, password, domain);


                // Define an anonymous LINQ query that returns a collection of Customer types.
                var query = from c in context.Customers
                            where c.Country == "Germany"
                            select c;

                try
                {
                    // Instantiate the binding collection, which executes the query.
                    binding = new DataServiceCollection<Customer>(query);

                    while (binding.Continuation != null)
                    {
                        // Continue to execute the query until all pages are loaded.
                        binding.Load(context.Execute<Customer>(binding.Continuation.NextLinkUri));
                    }

                    // Assign the binding collection to the CollectionViewSource.
                    customerAddressViewSource =
                        (CollectionViewSource)this.Resources["customerViewSource"];
                    customerAddressViewSource.Source = binding;
                }
                catch (Exception ex)
                {
                    MessageBox.Show(ex.Message);
                }
            }
            else if (login.DialogResult == false)
            {
                MessageBox.Show("Login cancelled.");
            }
        }
    }
}
```

```
Imports NorthwindClient.Northwind
Imports System.Data.Services.Client
Imports System.Windows.Data
Imports System.Net
Imports System.Windows
Imports System.Security

Partial Public Class ClientCredentials
    Inherits Window

    ' Create the binding collections and the data service context.
    Private binding As DataServiceCollection(Of Customer)
    Private context As NorthwindEntities
    Private customerAddressViewSource As CollectionViewSource

    ' Instantiate the service URI and credentials.
    Dim serviceUri As Uri = New Uri("http://localhost:54321/Northwind.svc/")
    Private credentials As NetworkCredential = New NetworkCredential()

    Public Sub Main()
        InitializeComponent()
    End Sub

    Private Sub ClientCredentials_Loaded(ByVal sender As Object, ByVal e As RoutedEventArgs)

        Dim userName = String.Empty
        Dim domain = String.Empty
        Dim password = New SecureString()

        ' Get credentials for authentication.
        Dim login As New LoginWindow()
        login.ShowDialog()

        If login.DialogResult = True _
            AndAlso Not login.userNameBox.Text Is String.Empty _
            AndAlso login.passwordBox.SecurePassword.Length <> 0 Then

            ' Instantiate the context.
            context = New NorthwindEntities(serviceUri)

            ' Get the user name and domain from the login.
            Dim qualifiedUserName As String() = login.userNameBox.Text.Split(New [Char]() {"\"c})
            If qualifiedUserName.Length = 2 Then
                domain = qualifiedUserName(0)
                userName = qualifiedUserName(1)
            Else
                userName = login.userNameBox.Text
            End If
            password = login.passwordBox.SecurePassword

            ' Set the client authentication credentials.
            context.Credentials = _
                New NetworkCredential(userName, password, domain)


            ' Define an anonymous LINQ query that returns a collection of Customer types.
            Dim query = From c In context.Customers
                        Where c.Country = "Germany"
                        Select c

            Try
                ' Instantiate the binding collection, which executes the query.
                binding = New DataServiceCollection(Of Customer)(query)

                ' Load result pages into the binding collection.
                While Not binding.Continuation Is Nothing
                    ' Continue to execute the query until all pages are loaded.
                    binding.Load(context.Execute(Of Customer)(binding.Continuation.NextLinkUri))
                End While

                ' Assign the binding collection to the CollectionViewSource.
                customerAddressViewSource = _
                    CType(Me.Resources("customerViewSource"), CollectionViewSource)
                customerAddressViewSource.Source = binding
            Catch ex As Exception
                MessageBox.Show(ex.Message)
            End Try
        ElseIf login.DialogResult = False Then
            MessageBox.Show("Login cancelled.")
        End If
    End Sub
End Class
```

Example

The following XAML defines the main page of the WPF application.

```
<Window x:Class="ClientCredentials"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d"
    d:DesignHeight="312" d:DesignWidth="577"
             Loaded="ClientCredentials_Loaded">
    <Window.Resources>
        <CollectionViewSource x:Key="customerViewSource" />
    </Window.Resources>
    <Grid x:Name="LayoutRoot" Background="White" DataContext="" Height="312" Width="577"
          VerticalAlignment="Top" HorizontalAlignment="Left">
        <Grid.RowDefinitions>
            <RowDefinition Height="203*" />
            <RowDefinition Height="119*" />
        </Grid.RowDefinitions>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="336*" />
        </Grid.ColumnDefinitions>
        <DataGrid AutoGenerateColumns="False" Height="213" HorizontalAlignment="Left"
                      ItemsSource="{Binding Source={StaticResource customerViewSource}}"
                      Name="customerDataGrid" RowDetailsVisibilityMode="VisibleWhenSelected"
                      VerticalAlignment="Top" Width="553" Margin="12,44,0,0"
                      Grid.RowSpan="2" Grid.ColumnSpan="1">
            <DataGrid.Columns>
                <DataGridTextColumn x:Name="customerIDColumn" Binding="{Binding Path=CustomerID}"
                                        Header="Customer" Width="80" />
                <DataGridTextColumn x:Name="addressColumn" Binding="{Binding Path=Address}"
                                        Header="Address" Width="180" />
                <DataGridTextColumn x:Name="cityColumn" Binding="{Binding Path=City}"
                                        Header="City" Width="120" />
                <DataGridTextColumn x:Name="countryColumn" Binding="{Binding Path=Country}"
                                        Header="Country" Width="80" />
                <DataGridTextColumn x:Name="postalCodeColumn" Binding="{Binding Path=PostalCode}"
                                        Header="Postal Code" Width="90" />
                <DataGridTextColumn Binding="{Binding Path=CompanyName}" Header="CompanyName" />
                <DataGridTextColumn Binding="{Binding Path=ContactName}" Header="ContactName" />
                <DataGridTextColumn Binding="{Binding Path=Phone}" Header="Phone" />
            </DataGrid.Columns>
        </DataGrid>
        <Label Grid.Row="0" Grid.Column="0" Height="26" HorizontalAlignment="Left" Margin="16,12,0,0"
                   Name="serviceUriLabel" VerticalAlignment="Top" Width="550"  />
    </Grid>
</Window>
```

Example

The following example is from the code-behind page for the window that is used to collect the authentication credentials from the user before making a request to the data service.

```
using System;
using System.Windows;
using System.Windows.Controls;
using System.ComponentModel;

namespace NorthwindClient
{
    public partial class LoginWindow : Window
    {
        public LoginWindow()
        {
            InitializeComponent();
        }

        private void OKButton_Click(object sender, RoutedEventArgs e)
        {
            this.DialogResult = true;
            e.Handled = true;
        }

        private void CancelButton_Click(object sender, RoutedEventArgs e)
        {
             this.DialogResult = false;
             e.Handled = true;
        }

        private void LoginWindow_Closing(object sender, CancelEventArgs e)
        {
            if (this.DialogResult == true &&
                    (this.userNameBox.Text == string.Empty || this.passwordBox.SecurePassword.Length == 0))
            {
                e.Cancel = true;
                MessageBox.Show("Please enter name and password or click Cancel.");
            }
        }
    }
}
```

```
Imports System.ComponentModel
Imports System.Windows
Imports System.Security

Partial Public Class LoginWindow
    Inherits Window

    Public Sub New()
        InitializeComponent()
    End Sub

    Private Sub OKButton_Click(ByVal sender As Object, ByVal e As RoutedEventArgs) Handles OKButton.Click
        Me.DialogResult = True
        e.Handled = True
    End Sub

    Private Sub CancelButton_Click(ByVal sender As Object, ByVal e As RoutedEventArgs) Handles
CancelButton.Click
        Me.DialogResult = False
        e.Handled = True
    End Sub

    Private Sub LoginWindow_Closing(ByVal sender As System.Object, ByVal e As CancelEventArgs)
        If Me.DialogResult = True AndAlso _
                    (Me.userNameBox.Text = String.Empty OrElse Me.passwordBox.SecurePassword.Length = 0) Then
            e.Cancel = True
            MessageBox.Show("Please enter name and password or click Cancel.")
        End If
    End Sub
End Class
```

Example

The following XAML defines the login of the WPF application.

```
<Window x:Class="LoginWindow"
           xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
           xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
           Width="400" Height="200"
           Title="LoginWindow" xmlns:sdk="http://schemas.microsoft.com/winfx/2006/xaml/presentation/sdk"
Closing="LoginWindow_Closing">
    <StackPanel Name="LayoutRoot" Orientation="Vertical" VerticalAlignment="Top">
        <StackPanel Orientation="Horizontal">
            <TextBlock Height="25" HorizontalAlignment="Left" Margin="10,20,0,0" Name="userNameLabel"
VerticalAlignment="Top"
                       Width="80" Text="User name:"/>
            <TextBox Height="23" HorizontalAlignment="Left" Margin="10,20,0,0"  Name="userNameBox"
VerticalAlignment="Top"
                     Width="150" Text="DOMAIN\login"/>
        </StackPanel>
        <StackPanel Orientation="Horizontal" VerticalAlignment="Top">
            <TextBlock Height="25" HorizontalAlignment="Left" Margin="10,20,0,0" Name="pwdLabel" Width="80"
Text="Password:"/>
            <PasswordBox Height="23" HorizontalAlignment="Left" Margin="10,20,0,0" Name="passwordBox"
Width="150" />
        </StackPanel>
        <StackPanel Orientation="Horizontal" HorizontalAlignment="Right" Height="80" VerticalAlignment="Top">
            <Button x:Name="CancelButton" Content="Cancel" Click="CancelButton_Click" Width="75" Height="23"
                HorizontalAlignment="Right" Margin="8" IsCancel="True" />
            <Button x:Name="OKButton" Content="OK" Click="OKButton_Click" Width="75" Height="23"
                HorizontalAlignment="Right" Margin="8" IsDefault="True" />
        </StackPanel>
    </StackPanel>
</Window>
```

.NET Framework Security

The following security considerations apply to the example in this topic:

- To verify that the credentials supplied in this sample work, the Northwind data service must use an authentication scheme other than anonymous access. Otherwise, the Web site hosting the data service will not request credentials.
- User credentials should only be requested during execution and should not be cached. Credentials must always be stored securely.
- Data sent with Basic and Digest Authentication is not encrypted, so the data can be seen by an adversary. Additionally, basic authentication credentials (user name and password) are sent in cleartext and can be intercepted.

For more information, see Securing WCF Data Services.

See Also

Securing WCF Data Services
WCF Data Services Client Library

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

8/31/2018 • 3 minutes to read • Edit Online

When you use the WCF Data Services client library to access a data service that supports the Open Data Protocol (OData), the client library automatically sets the required HTTP headers in request messages sent to the data service. However, the client library does not know to set message headers that are required in certain cases, such as when the data service requires claims-based authentication or cookies. For more information, see Securing WCF Data Services. In these cases, you must manually set message headers in the request message before it is sent. The example in this topic shows how to handle the SendingRequest event to add a new header to the request message before it is sent to the data service.

The example in this topic uses the Northwind sample data service and autogenerated client data service classes. This service and the client data classes are created when you complete the WCF Data Services quickstart. You can also use the Northwind sample data service that is published on the OData Web site; this sample data service is read-only and attempting to save changes returns an error. The sample data services on the OData Web site allow anonymous authentication.

Example

The following example registers a handler for the SendingRequest event and then executes a query against the data service.

NOTE

When a data service requires you to manually set the message header for every request, consider registering the handler for the SendingRequest event by overriding the `OnContextCreated` partial method in the entity container that represents the data service, which in this case is `NorthwindEntities`.

```
// Create the DataServiceContext using the service URI.
NorthwindEntities context = new NorthwindEntities(svcUri);

// Register to handle the SendingRequest event.
// Note: If this must be done for every request to the data service, consider
// registering for this event by overriding the OnContextCreated partial method in
// the entity container, in this case NorthwindEntities.
context.SendingRequest += new EventHandler<SendingRequestEventArgs>(OnSendingRequest);

// Define a query for orders with a Freight value greater than 30.
var query = from cust in context.Customers
    where cust.Country == "Germany"
    select cust;

try
{
    // Enumerate to execute the query.
    foreach (Customer cust in query)
    {
        Console.WriteLine("Name: {0}\nAddress:\n{1}\n{2}, {3}",
            cust.CompanyName, cust.Address, cust.City, cust.Region);
    }
}
catch (DataServiceQueryException ex)
{
    throw new ApplicationException(
        "An error occurred during query execution.", ex);
}
```

```
' Create the DataServiceContext using the service URI.
Dim context As NorthwindEntities = New NorthwindEntities(svcUri)

' Register to handle the SendingRequest event.
' Note: If this must be done for every request to the data service, consider
' registering for this event by overriding the OnContextCreated partial method in
' the entity container, in this case NorthwindEntities.
AddHandler context.SendingRequest, AddressOf OnSendingRequest

' Define a query for orders with a Freight value greater than 30.
Dim query = From cust In context.Customers _
    Where cust.Country = "Germany" _
    Select cust

Try
    ' Enumerate to execute the query.
    For Each cust As Customer In query
        Console.WriteLine("Name: {0}" & vbNewLine & "Address:" & vbNewLine & "{1}" _
                          & vbNewLine & "{2}, {3}", _
            cust.CompanyName, cust.Address, cust.City, cust.Region)
    Next
Catch ex As DataServiceQueryException
    Throw New ApplicationException( _
        "An error occurred during query execution.", ex)
End Try
```

Example

The following method handles the SendingRequest event and adds an Authentication header to the request.

```
private static void OnSendingRequest(object sender, SendingRequestEventArgs e)
{
    // Add an Authorization header that contains an OAuth WRAP access token to the request.
    e.RequestHeaders.Add("Authorization", "WRAP access_token=\"123456789\"");
}
```

```
Private Shared Sub OnSendingRequest(ByVal sender As Object, ByVal e As SendingRequestEventArgs)
    ' Add an Authorization header that contains an OAuth WRAP access token to the request.
    e.RequestHeaders.Add("Authorization", "WRAP access_token=""123456789""")
End Sub
```

See Also

Securing WCF Data Services
WCF Data Services Client Library

Loading Deferred Content (WCF Data Services)

5/2/2018 • 6 minutes to read • Edit Online

By default, WCF Data Services limits the amount of data that a query returns. However, you can explicitly load additional data, including related entities, paged response data, and binary data streams, from the data service when it is needed. This topic describes how to load such deferred content into your application.

Related Entities

When you execute a query, only entities in the addressed entity set are returned. For example, when a query against the Northwind data service returns `Customers` entities, by default the related `Orders` entities are not returned, even though there is a relationship between `Customers` and `Orders`. Also, when paging is enabled in the data service, you must explicitly load subsequent data pages from the service. There are two ways to load related entities:

- **Eager loading**: You can use the `$expand` query option to request that the query return entities that are related by an association to the entity set that the query requested. Use the Expand method on the DataServiceQuery<TElement> to add the `$expand` option to the query that is sent to the data service. You can request multiple related entity sets by separating them by a comma, as in the following example. All entities requested by the query are returned in a single response. The following example returns `Order_Details` and `Customers` together with the `Orders` entity set:

  ```
  // Define a query for orders that also returns items and customers.
  DataServiceQuery<Order> query =
      context.Orders.Expand("Order_Details,Customer");
  ```

  ```
  ' Define a query for orders that also returns items and customers.
  Dim query As DataServiceQuery(Of Order) = _
  context.Orders.Expand("Order_Details,Customer")
  ```

  WCF Data Services limits to 12 the number of entity sets that can be included in a single query by using the `$expand` query option.

- **Explicit loading**: You can call the LoadProperty method on the DataServiceContext instance to explicitly load related entities. Each call to the LoadProperty method creates a separate request to the data service. The following example explicitly loads `Order_Details` for an `Orders` entity:

  ```
  // Explicitly load the order details for each order.
  context.LoadProperty(order, "Order_Details");
  ```

  ```
  ' Explicitly load the order details for each order.
  context.LoadProperty(order, "Order_Details")
  ```

When you consider which option to use, realize that there is a tradeoff between the number of requests to the data service and the amount of data that is returned in a single response. Use eager loading when your application requires associated objects and you want to avoid the added latency of additional requests to explicitly retrieve them. However, if there are cases when the application only needs the data for specific related entity instances, you should consider explicitly loading those entities by calling the LoadProperty method. For more information, see How to: Load Related Entities.

Paged Content

When paging is enabled in the data service, the number of entries in the feed that the data service returns is limited by the configuration of the data service. Page limits can be set separately for each entity set. For more information, see Configuring the Data Service. When paging is enabled, the final entry in the feed contains a link to the next page of data. This link is contained in a DataServiceQueryContinuation<T> object. You obtain the URI to the next page of data by calling the GetContinuation method on the QueryOperationResponse<T> returned when the DataServiceQuery<TElement> is executed. The returned DataServiceQueryContinuation<T> object is then used to load the next page of results. You must enumerate the query result before you call the GetContinuation method. Consider using a `do…while` loop to first enumerate the query result and then check for a `non-null` next link value. When the GetContinuation method returns `null` (`Nothing` in Visual Basic), there are no additional result pages for the original query. The following example shows a `do…while` loop that loads paged customer data from the Northwind sample data service.

```
// With a paged response from the service, use a do...while loop
// to enumerate the results before getting the next link.
do
{
    // Write the page number.
    Console.WriteLine("Page {0}:", pageCount++);

    // If nextLink is not null, then there is a new page to load.
    if (token != null)
    {
        // Load the new page from the next link URI.
        response = context.Execute<Customer>(token)
            as QueryOperationResponse<Customer>;
    }

    // Enumerate the customers in the response.
    foreach (Customer customer in response)
    {
        Console.WriteLine("\tCustomer Name: {0}", customer.CompanyName);
    }
}

// Get the next link, and continue while there is a next link.
while ((token = response.GetContinuation()) != null);
```

```
' With a paged response from the service, use a do...while loop
' to enumerate the results before getting the next link.
Do
    ' Write the page number.
    Console.WriteLine("Page {0}:", pageCount + 1)

    ' If nextLink is not null, then there is a new page to load.
    If token IsNot Nothing Then
        ' Load the new page from the next link URI.
        response = CType(context.Execute(Of Customer)(token),  _
        QueryOperationResponse(Of Customer))
    End If

    ' Enumerate the customers in the response.
    For Each customer As Customer In response
        Console.WriteLine(vbTab & "Customer Name: {0}", customer.CompanyName)
    Next

    ' Get the next link, and continue while there is a next link.
    token = response.GetContinuation()
Loop While token IsNot Nothing
```

When a query requests that related entities are returned in a single response together with the requested entity set, paging limits may affect nested feeds that are included inline with the response. For example, when a paging limit is set in the Northwind sample data service for the `Customers` entity set, an independent paging limit can also be set for the related `Orders` entity set, as in the following example from the Northwind.svc.cs file that defines the Northwind sample data service.

```
// Set page size defaults for the data service.
config.SetEntitySetPageSize("Orders", 20);
config.SetEntitySetPageSize("Order_Details", 50);
config.SetEntitySetPageSize("Products", 50);

// Paging requires v2 of the OData protocol.
config.DataServiceBehavior.MaxProtocolVersion =
    System.Data.Services.Common.DataServiceProtocolVersion.V2;
```

```
' Set page size defaults for the data service.
config.SetEntitySetPageSize("Orders", 20)
config.SetEntitySetPageSize("Order_Details", 50)
config.SetEntitySetPageSize("Products", 50)

' Paging requires v2 of the OData protocol.
config.DataServiceBehavior.MaxProtocolVersion = _
    System.Data.Services.Common.DataServiceProtocolVersion.V2
```

In this case, you must implement paging for both the top-level `Customers` and the nested `Orders` entity feeds. The following example shows the `while` loop used to load pages of `Orders` entities related to a selected `Customers` entity.

```
while (nextOrdersLink != null)
{
    foreach (Order o in c.Orders)
    {
        // Print out the orders.
        Console.WriteLine("\t\tOrderID: {0} - Freight: ${1}",
            o.OrderID, o.Freight);
    }

    // Load the next page of Orders.
    var ordersResponse = context.LoadProperty(c, "Orders", nextOrdersLink);
    nextOrdersLink = ordersResponse.GetContinuation();
}
```

```
While nextOrdersLink IsNot Nothing
    For Each o As Order In c.Orders
        ' Print out the orders.
        Console.WriteLine(vbTab & vbTab & "OrderID: {0} - Freight: ${1}", _
                o.OrderID, o.Freight)
    Next
    ' Load the next page of Orders.
    Dim ordersResponse = _
    context.LoadProperty(c, "Orders", nextOrdersLink)
    nextOrdersLink = ordersResponse.GetContinuation()
End While
```

For more information, see How to: Load Paged Results.

Binary Data Streams

WCF Data Services enables you to access binary large object (BLOB) data as a data stream. Streaming defers the loading of binary data until it is needed, and the client can more efficiently process this data. In order to take advantage of this functionality, the data service must implement the IDataServiceStreamProvider provider. For more information, see Streaming Provider. When streaming is enabled, entity types are returned without the related binary data. In this case, you must use the GetReadStream method of the DataServiceContext class to access the data stream for the binary data from the service. Similarly, use the SetSaveStream method to add or change binary data for an entity as a stream. For more information, see Working with Binary Data.

See Also

WCF Data Services Client Library
Querying the Data Service

How to: Load Related Entities (WCF Data Services)

5/2/2018 • 2 minutes to read • Edit Online

When you need to load associated entities in WCF Data Services, you can use the LoadProperty method on the DataServiceContext class. You can also use the Expand method on the DataServiceQuery<TElement> to require that related entities be eagerly loaded in the same query response.

The example in this topic uses the Northwind sample data service and autogenerated client data service classes. This service and the client data classes are created when you complete the WCF Data Services quickstart.

Example

The following example shows how to explicitly load the `Customer` that is related to each returned `Orders` instance.

```
// Create the DataServiceContext using the service URI.
NorthwindEntities context = new NorthwindEntities(svcUri);

try
{
    // Enumerate over the top 10 orders obtained from the context.
    foreach (Order order in context.Orders.Take(10))
    {
        // Explicitly load the customer for each order.
        context.LoadProperty(order, "Customer");

        // Write out customer and order information.
        Console.WriteLine("Customer: {0} - Order ID: {1}",
            order.Customer.CompanyName, order.OrderID);
    }
}
catch (DataServiceQueryException ex)
{
    throw new ApplicationException(
        "An error occurred during query execution.", ex);
}
```

```
' Create the DataServiceContext using the service URI.
Dim context = New NorthwindEntities(svcUri)

Try
    ' Enumerate over the top 10 orders obtained from the context.
    For Each order As Order In context.Orders.Take(10)
        ' Explicitly load the customer for each order.
        context.LoadProperty(order, "Customer")

        ' Write out customer and order information.
        Console.WriteLine("Customer: {0} - Order ID: {1}", _
                order.Customer.CompanyName, order.OrderID)
    Next
Catch ex As DataServiceQueryException
    Throw New ApplicationException( _
            "An error occurred during query execution.", ex)
End Try
```

Example

The following example shows how to use the Expand method to return `Order Details` that belong to the `Orders` returned by the query.

```
// Create the DataServiceContext using the service URI.
NorthwindEntities context = new NorthwindEntities(svcUri);

// Define a query for orders that also returns items and customers.
DataServiceQuery<Order> query =
    context.Orders.Expand("Order_Details,Customer");

try
{
    // Enumerate over the first 10 results of the query.
    foreach (Order order in query.Take(10))
    {
        Console.WriteLine("Customer: {0}", order.Customer.CompanyName);
        Console.WriteLine("Order ID: {0}", order.OrderID);

        foreach (Order_Detail item in order.Order_Details)
        {
            Console.WriteLine("\tProduct: {0} - Quantity: {1}",
                item.ProductID, item.Quantity);
        }
    }
}
catch (DataServiceQueryException ex)
{
    throw new ApplicationException(
        "An error occurred during query execution.", ex);
}
```

```
' Create the DataServiceContext using the service URI.
Dim context = New NorthwindEntities(svcUri)

' Define a query for orders that also returns items and customers.
Dim query As DataServiceQuery(Of Order) = _
context.Orders.Expand("Order_Details,Customer")

Try
    ' Enumerate over the first 10 results of the query.
    For Each order As Order In query.Take(10)
        Console.WriteLine("Customer: {0}", order.Customer.CompanyName)
        Console.WriteLine("Order ID: {0}", order.OrderID)

        For Each item As Order_Detail In order.Order_Details
            Console.WriteLine(vbTab & "Product: {0} - Quantity: {1}", _
                    item.ProductID, item.Quantity)
        Next
    Next
Catch ex As DataServiceQueryException
    Throw New ApplicationException( _
            "An error occurred during query execution.", ex)
End Try
```

See Also

Querying the Data Service

How to: Load Paged Results (WCF Data Services)

5/2/2018 • 5 minutes to read • Edit Online

WCF Data Services enables the data service to limit the number of entities that are returned in a single response feed. When this happens, the final entry in the feed contains a link to the next page of data. The URI for the next page of data is obtained by calling the GetContinuation method of the QueryOperationResponse<T>, which is returned when the DataServiceQuery<TElement> is executed. The URI represented by this object is then used to load the next page of results. For more information, see Loading Deferred Content.

The example in this topic uses the Northwind sample data service and autogenerated client data service classes. This service and the client data classes are created when you complete the WCF Data Services quickstart.

Example

This example uses a `do…while` loop to load `Customers` entities from a paged results from the data service.

```
// Create the DataServiceContext using the service URI.
NorthwindEntities context = new NorthwindEntities(svcUri);
DataServiceQueryContinuation<Customer> token = null;
int pageCount = 0;

try
{
    // Execute the query for all customers and get the response object.
    QueryOperationResponse<Customer> response =
        context.Customers.Execute() as QueryOperationResponse<Customer>;

    // With a paged response from the service, use a do...while loop
    // to enumerate the results before getting the next link.
    do
    {
        // Write the page number.
        Console.WriteLine("Page {0}:", pageCount++);

        // If nextLink is not null, then there is a new page to load.
        if (token != null)
        {
            // Load the new page from the next link URI.
            response = context.Execute<Customer>(token)
                as QueryOperationResponse<Customer>;
        }

        // Enumerate the customers in the response.
        foreach (Customer customer in response)
        {
            Console.WriteLine("\tCustomer Name: {0}", customer.CompanyName);
        }
    }

    // Get the next link, and continue while there is a next link.
    while ((token = response.GetContinuation()) != null);
}
catch (DataServiceQueryException ex)
{
    throw new ApplicationException(
        "An error occurred during query execution.", ex);
}
```

```
' Create the DataServiceContext using the service URI.
Dim context = New NorthwindEntities(svcUri)
Dim token As DataServiceQueryContinuation(Of Customer) = Nothing
Dim pageCount = 0

Try
    ' Execute the query for all customers and get the response object.
    Dim response As QueryOperationResponse(Of Customer) = _
        CType(context.Customers.Execute(), QueryOperationResponse(Of Customer))

    ' With a paged response from the service, use a do...while loop
    ' to enumerate the results before getting the next link.
    Do
        ' Write the page number.
        Console.WriteLine("Page {0}:", pageCount + 1)

        ' If nextLink is not null, then there is a new page to load.
        If token IsNot Nothing Then
            ' Load the new page from the next link URI.
            response = CType(context.Execute(Of Customer)(token),  _
            QueryOperationResponse(Of Customer))
        End If

        ' Enumerate the customers in the response.
        For Each customer As Customer In response
            Console.WriteLine(vbTab & "Customer Name: {0}", customer.CompanyName)
        Next

        ' Get the next link, and continue while there is a next link.
        token = response.GetContinuation()
    Loop While token IsNot Nothing
Catch ex As DataServiceQueryException
    Throw New ApplicationException( _
            "An error occurred during query execution.", ex)
End Try
```

Example

This example returns related `Orders` entities with each `Customers` entity and uses a `do…while` loop to load `Customers` entities pages and a nested `while` loop to load pages of related `Orders` entities from the data service.

```
// Create the DataServiceContext using the service URI.
NorthwindEntities context = new NorthwindEntities(svcUri);
DataServiceQueryContinuation<Customer> nextLink = null;
int pageCount = 0;
int innerPageCount = 0;

try
{
    // Execute the query for all customers and related orders,
    // and get the response object.
    var response =
        context.Customers.AddQueryOption("$expand", "Orders")
        .Execute() as QueryOperationResponse<Customer>;

    // With a paged response from the service, use a do...while loop
    // to enumerate the results before getting the next link.
    do
    {
        // Write the page number.
        Console.WriteLine("Customers Page {0}:", ++pageCount);

        // If nextLink is not null, then there is a new page to load.
        if (nextLink != null)
        {
            // Load the new page from the next link URI.
            response = context.Execute<Customer>(nextLink)
                as QueryOperationResponse<Customer>;
        }

        // Enumerate the customers in the response.
        foreach (Customer c in response)
        {
            Console.WriteLine("\tCustomer Name: {0}", c.CompanyName);
            Console.WriteLine("\tOrders Page {0}:", ++innerPageCount);
            // Get the next link for the collection of related Orders.
            DataServiceQueryContinuation<Order> nextOrdersLink =
                response.GetContinuation(c.Orders);

            while (nextOrdersLink != null)
            {
                foreach (Order o in c.Orders)
                {
                    // Print out the orders.
                    Console.WriteLine("\t\tOrderID: {0} - Freight: ${1}",
                        o.OrderID, o.Freight);
                }

                // Load the next page of Orders.
                var ordersResponse = context.LoadProperty(c, "Orders", nextOrdersLink);
                nextOrdersLink = ordersResponse.GetContinuation();
            }
        }
    }

    // Get the next link, and continue while there is a next link.
    while ((nextLink = response.GetContinuation()) != null);
}
catch (DataServiceQueryException ex)
{
    throw new ApplicationException(
        "An error occurred during query execution.", ex);
}
```

```
' Create the DataServiceContext using the service URI.
Dim context = New NorthwindEntities(svcUri)
Dim nextLink As DataServiceQueryContinuation(Of Customer) = Nothing
Dim pageCount = 0
Dim innerPageCount = 0

Try
    ' Execute the query for all customers and related orders,
    ' and get the response object.
    Dim response = _
    CType(context.Customers.AddQueryOption("$expand", "Orders") _
            .Execute(), QueryOperationResponse(Of Customer))

    ' With a paged response from the service, use a do...while loop
    ' to enumerate the results before getting the next link.
    Do
        ' Write the page number.
        Console.WriteLine("Customers Page {0}:", ++pageCount)

        ' If nextLink is not null, then there is a new page to load.
        If nextLink IsNot Nothing Then
            ' Load the new page from the next link URI.
            response = CType(context.Execute(Of Customer)(nextLink),  _
                    QueryOperationResponse(Of Customer))
        End If

        ' Enumerate the customers in the response.
        For Each c As Customer In response
            Console.WriteLine(vbTab & "Customer Name: {0}", c.CompanyName)
            Console.WriteLine(vbTab & "Orders Page {0}:", innerPageCount + 1)

            ' Get the next link for the collection of related Orders.
            Dim nextOrdersLink As DataServiceQueryContinuation(Of Order) = _
            response.GetContinuation(c.Orders)

            While nextOrdersLink IsNot Nothing
                For Each o As Order In c.Orders
                    ' Print out the orders.
                    Console.WriteLine(vbTab & vbTab & "OrderID: {0} - Freight: ${1}", _
                            o.OrderID, o.Freight)
                Next
                ' Load the next page of Orders.
                Dim ordersResponse = _
                context.LoadProperty(c, "Orders", nextOrdersLink)
                nextOrdersLink = ordersResponse.GetContinuation()
            End While
        Next
        ' Get the next link, and continue while there is a next link.
        nextLink = response.GetContinuation()
    Loop While nextLink IsNot Nothing
Catch ex As DataServiceQueryException
    Throw New ApplicationException( _
            "An error occurred during query execution.", ex)
End Try
```

See Also

Loading Deferred Content
How to: Load Related Entities

Updating the Data Service (WCF Data Services)

5/2/2018 • 6 minutes to read • Edit Online

When you use the WCF Data Services client library to consume an Open Data Protocol (OData) feed, the library translates the entries in the feed into instances of client data service classes. These data service classes are tracked by using the DataServiceContext to which the DataServiceQuery<TElement> belongs. The client tracks changes to entities that you report by using methods on DataServiceContext. These methods enable the client to track added and deleted entities and also changes that you make to property values or to relationships between entity instances. Those tracked changes are sent back to the data service as REST-based operations when you call the SaveChanges method.

NOTE

When you use an instance of DataServiceCollection<T> to bind data to controls, changes made to data in the bound control are automatically reported to the DataServiceContext. For more information, see Binding Data to Controls.

Adding, Modifying, and Changing Entities

When you use the **Add Service Reference** dialog in Visual Studio to add a reference to an OData feed, the resulting client data service classes each have a static *Create* method that takes one parameter for each non-nullable entity property. You can use this method to create instances of entity type classes, as in the following example:

```
// Create the new product.
Product newProduct =
    Product.CreateProduct(0, "White Tea - loose", false);
```

```
' Create the new product.
Dim newProduct = _
    Product.CreateProduct(0, "White Tea - loose", False)
```

To add an entity instance, call the appropriate *AddTo* method on the DataServiceContext class generated by the **Add Service Reference** dialog box, as in the following example:

```
// Add the new product to the Products entity set.
context.AddToProducts(newProduct);
```

```
' Add the new product to the Products entity set.
context.AddToProducts(newProduct)
```

This adds the object to the context and into the correct entity set. You can also call AddObject, but you must instead supply the entity set name. If the added entity has one or more relationships to other entities, you can either use the AddRelatedObject method or use one of the previous methods and also explicitly define those links. These operations are discussed later in this topic.

To modify an existing entity instance, first query for that entity, make the desired changes to its properties, and then call the UpdateObject method on the DataServiceContext to indicate to the client library that it needs to send an update for that object, as shown in the following example:

```
// Mark the customer as updated.
context.UpdateObject(customerToChange);
```

```
' Mark the customer as updated.
context.UpdateObject(customerToChange)
```

To delete an entity instance, call the DeleteObject method on the DataServiceContext, as shown in the following example:

```
// Mark the product for deletion.
context.DeleteObject(deletedProduct);
```

```
' Mark the product for deletion.
context.DeleteObject(deletedProduct)
```

For more information, see How to: Add, Modify, and Delete Entities.

Attaching Entities

The client library enables you to save updates that you made to an entity without first executing a query to load the entity into the DataServiceContext. Use the AttachTo method to attach an existing object to a specific entity set in the DataServiceContext. You can then modify the object and save the changes to the data service. In the following example, a customer object that has been changed is attached to the context and then UpdateObject is called to mark the attached object as Modified before SaveChanges is called:

```
// Attach the existing customer to the context and mark it as updated.
context.AttachTo("Customers", customer);
context.UpdateObject(customer);

// Send updates to the data service.
context.SaveChanges();
```

```
' Attach the existing customer to the context and mark it as updated.
context.AttachTo("Customers", customer)
context.UpdateObject(customer)

' Send updates to the data service.
context.SaveChanges()
```

The following considerations apply when attaching objects:

- An object is attached in the Unchanged state.
- When an object is attached, objects that are related to the attached object are not also attached.
- An object cannot be attached if the entity is already being tracked by the context.
- The AttachTo(String, Object, String) method overload that takes an `etag` parameter is used when you attach an entity object that was received along with an eTag value. This eTag value is then used to check for concurrency when changes to the attached object are saved.

For more information, see How to: Attach an Existing Entity to the DataServiceContext.

Creating and Modifying Relationship Links

When you add a new entity by using either the AddObject method or the appropriate *AddTo* method of the DataServiceContext class that the **Add Service Reference** dialog generates, any relationships between the new entity and related entities are not automatically defined.

You can create and change relationships between entity instances and have the client library reflect those changes in the data service. Relationships between entities are defined as associations in the model, and the DataServiceContext tracks each relationship as a link object in the context. WCF Data Services provides the following methods on the DataServiceContext class to create, modify, and delete these links:

METHOD	DESCRIPTION
AddRelatedObject	Creates a new link between two related entity objects. Calling this method is equivalent to calling AddObject and AddLink to both create the new object and define the relationship to an existing object.
AddLink	Creates a new link between two related entity objects.
SetLink	Updates an existing link between two related entity objects. SetLink is also used to delete links with a cardinality of zero-or-one-to-one (`0..1:1`) and one-to-one (`1:1`). You can do this by setting the related object to `null`.
DeleteLink	Marks a link that the context is tracking for deletion when the SaveChanges method is called. Use this method when you delete a related object or change a relationship by first deleting the link to an existing object and then adding a link to the new related object.
AttachLink	Notifies the context of an existing link between two entity objects. The context assumes that this relationship already exists in the data service and does not try to create the link when you call the SaveChanges method. Use this method when you attach objects to a context and need to also attach the link between the two. If you are defining a new relationship, you should instead use AddLink.
DetachLink	Stops tracking the specified link in the context. This method is used to delete one-to-many (`*:*`) relationships. For relationship links with a cardinality of one, you must instead use SetLink.

The following example shows how to use the AddRelatedObject method to add a new `Order_Detail` that is related to an existing `Orders` entity. Because the new `Order_Details` object is now tracked by the DataServiceContext, the relationship of the added `Order_Details` object to the existing `Products` entity is defined by calling the AddLink method:

```
// Add the new item with a link to the related order.
context.AddRelatedObject(order, "Order_Details", newItem);
```

```
' Add the new item with a link to the related order.
context.AddRelatedObject(order, "Order_Details", newItem)
```

While the AddLink method defines links that must be created in the data service, to have these links reflected in the objects that are in the context, you must also set the navigation properties on the objects themselves. In the previous example, you should set the navigation properties as follows:

```
// Add the new order detail to the collection, and
// set the reference to the product.
order.Order_Details.Add(newItem);
newItem.Order = order;
newItem.Product = selectedProduct;
```

```
' Add the new order detail to the collection, and
' set the reference to the product.
order.Order_Details.Add(newItem)
newItem.Order = order
newItem.Product = selectedProduct
```

For more information, see How to: Define Entity Relationships.

Saving Changes

Changes are tracked in the DataServiceContext instance but not sent to the server immediately. After you are finished with the required changes for a specified activity, call SaveChanges to submit all the changes to the data service. For more information, see Managing the Data Service Context. You can also save changes asynchronously by using the BeginSaveChanges and EndSaveChanges methods. For more information, see Asynchronous Operations.

See Also

WCF Data Services Client Library
Querying the Data Service
Asynchronous Operations
Batching Operations
Object Materialization
Managing the Data Service Context

How to: Add, Modify, and Delete Entities (WCF Data Services)

5/2/2018 • 6 minutes to read • Edit Online

With the WCF Data Services client libraries, you can create, update, and delete entity data in a data service by performing equivalent actions on objects in the DataServiceContext. For more information, see Updating the Data Service.

The example in this topic uses the Northwind sample data service and autogenerated client data service classes. This service and the client data classes are created when you complete the WCF Data Services quickstart.

Example

The following example creates a new object instance and then calls the AddObject method on the DataServiceContext to create the item in the context. An HTTP POST message is sent to the data service when the SaveChanges method is called.

```
// Create the DataServiceContext using the service URI.
NorthwindEntities context = new NorthwindEntities(svcUri);

// Create the new product.
Product newProduct =
    Product.CreateProduct(0, "White Tea - loose", false);

// Set property values.
newProduct.QuantityPerUnit = "120gm bags";
newProduct.ReorderLevel = 5;
newProduct.UnitPrice = 5.2M;

try
{
    // Add the new product to the Products entity set.
    context.AddToProducts(newProduct);

    // Send the insert to the data service.
    DataServiceResponse response = context.SaveChanges();

    // Enumerate the returned responses.
    foreach (ChangeOperationResponse change in response)
    {
        // Get the descriptor for the entity.
        EntityDescriptor descriptor = change.Descriptor as EntityDescriptor;

        if (descriptor != null)
        {
            Product addedProduct = descriptor.Entity as Product;

            if (addedProduct != null)
            {
                Console.WriteLine("New product added with ID {0}.",
                    addedProduct.ProductID);
            }
        }
    }
}
catch (DataServiceRequestException ex)
{
    throw new ApplicationException(
        "An error occurred when saving changes.", ex);
}
```

```
' Create the DataServiceContext using the service URI.
Dim context = New NorthwindEntities(svcUri)

' Create the new product.
Dim newProduct = _
    Product.CreateProduct(0, "White Tea - loose", False)

' Set property values.
newProduct.QuantityPerUnit = "120gm bags"
newProduct.ReorderLevel = 5
newProduct.UnitPrice = 5.2D

Try
    ' Add the new product to the Products entity set.
    context.AddToProducts(newProduct)

    ' Send the insert to the data service.
    context.SaveChanges()

    Console.WriteLine("New product added with ID {0}.", newProduct.ProductID)
Catch ex As DataServiceRequestException
    Throw New ApplicationException( _
            "An error occurred when saving changes.", ex)
```

Example

The following example retrieves and modifies an existing object and then calls the UpdateObject method on the DataServiceContext to mark the item in the context as updated. An HTTP MERGE message is sent to the data service when the SaveChanges method is called.

```
string customerId = "ALFKI";

// Create the DataServiceContext using the service URI.
NorthwindEntities context = new NorthwindEntities(svcUri);

// Get a customer to modify using the supplied ID.
var customerToChange = (from customer in context.Customers
                        where customer.CustomerID == customerId
                        select customer).Single();

// Change some property values.
customerToChange.CompanyName = "Alfreds Futterkiste";
customerToChange.ContactName = "Maria Anders";
customerToChange.ContactTitle = "Sales Representative";

try
{
    // Mark the customer as updated.
    context.UpdateObject(customerToChange);

    // Send the update to the data service.
    context.SaveChanges();
}
catch (DataServiceRequestException  ex)
{
    throw new ApplicationException(
        "An error occurred when saving changes.", ex);
}
```

```
Dim customerId = "ALFKI"

' Create the DataServiceContext using the service URI.
Dim context = New NorthwindEntities(svcUri)

' Get a customer to modify using the supplied ID.
Dim customerToChange = (From customer In context.Customers _
                        Where customer.CustomerID = customerId _
                        Select customer).Single()

' Change some property values.
customerToChange.CompanyName = "Alfreds Futterkiste"
customerToChange.ContactName = "Maria Anders"
customerToChange.ContactTitle = "Sales Representative"

Try
    ' Mark the customer as updated.
    context.UpdateObject(customerToChange)

    ' Send the update to the data service.
    context.SaveChanges()
Catch ex As DataServiceRequestException
    Throw New ApplicationException( _
            "An error occurred when saving changes.", ex)
End Try
```

Example

The following example calls the DeleteObject method on the DataServiceContext to mark the item in the context as deleted. An HTTP DELETE message is sent to the data service when the SaveChanges method is called.

```
// Create the DataServiceContext using the service URI.
NorthwindEntities context = new NorthwindEntities(svcUri);

try
{
    // Get the product to delete, by product ID.
    var deletedProduct = (from product in context.Products
                          where product.ProductID == productID
                          select product).Single();

    // Mark the product for deletion.
    context.DeleteObject(deletedProduct);

    // Send the delete to the data service.
    context.SaveChanges();
}
// Handle the error that occurs when the delete operation fails,
// which can happen when there are entities with existing
// relationships to the product being deleted.
catch (DataServiceRequestException ex)
{
    throw new ApplicationException(
        "An error occurred when saving changes.", ex);
}
```

```
' Create the DataServiceContext using the service URI.
Dim context = New NorthwindEntities(svcUri)

Try
    ' Get the product to delete, by product ID.
    Dim deletedProduct = (From product In context.Products _
                          Where product.ProductID = productID _
                          Select product).Single()


    ' Mark the product for deletion.
    context.DeleteObject(deletedProduct)

    ' Send the delete to the data service.
    context.SaveChanges()

    ' Handle the error that occurs when the delete operation fails,
    ' which can happen when there are entities with existing
    ' relationships to the product being deleted.
Catch ex As DataServiceRequestException
    Throw New ApplicationException( _
            "An error occurred when saving changes.", ex)
End Try
```

Example

The following example creates a new object instance and then calls the AddRelatedObject method on the DataServiceContext to create the item in the context along with the link to the related order. An HTTP POST message is sent to the data service when the SaveChanges method is called.

```
int productId = 25;
string customerId = "ALFKI";

Order_Detail newItem = null;

// Create the DataServiceContext using the service URI.
NorthwindEntities context = new NorthwindEntities(svcUri);

try
{
    // Get the specific product.
    var selectedProduct = (from product in context.Products
                           where product.ProductID == productId
                           select product).Single();

    // Get the specific customer.
    var cust = (from customer in context.Customers.Expand("Orders")
                where customer.CustomerID == customerId
                select customer).Single();

    // Get the first order.
    Order order = cust.Orders.FirstOrDefault();

    // Create a new order detail for the specific product.
    newItem = Order_Detail.CreateOrder_Detail(
        order.OrderID, selectedProduct.ProductID, 10, 5, 0);

    // Add the new item with a link to the related order.
    context.AddRelatedObject(order, "Order_Details", newItem);

    // Add the new order detail to the collection, and
    // set the reference to the product.
    order.Order_Details.Add(newItem);
    newItem.Order = order;
    newItem.Product = selectedProduct;

    // Send the changes to the data service.
    DataServiceResponse response = context.SaveChanges();

    // Enumerate the returned responses.
    foreach (ChangeOperationResponse change in response)
    {
        // Get the descriptor for the entity.
        EntityDescriptor descriptor = change.Descriptor as EntityDescriptor;

        if (descriptor != null)
        {
            if (descriptor.Entity.GetType() == typeof(Order_Detail))
            {
                Order_Detail addedItem = descriptor.Entity as Order_Detail;

                if (addedItem != null)
                {
                    Console.WriteLine("New {0} item added to order {1}.",
                        addedItem.Product.ProductName, addedItem.OrderID.ToString());
                }
            }
        }
    }
}
catch (DataServiceQueryException ex)
{
    throw new ApplicationException(
        "An error occurred when saving changes.", ex);
}

// Handle any errors that may occur during insert, such as
// a constraint violation.
catch (DataServiceRequestException ex)
{
    throw new ApplicationException(
        "An error occurred when saving changes.", ex);
}
```

```
Dim productId = 25
Dim customerId = "ALFKI"

Dim newItem As Order_Detail = Nothing

' Create the DataServiceContext using the service URI.
Dim context = New NorthwindEntities(svcUri)

Try
    ' Get the specific product.
    Dim selectedProduct = (From product In context.Products _
                           Where product.ProductID = productId _
                           Select product).Single()

    ' Get the specific customer.
    Dim cust = (From customer In context.Customers.Expand("Orders") _
                Where customer.CustomerID = customerId _
                Select customer).Single()

    ' Get the first order.
    Dim order = cust.Orders.FirstOrDefault()

    ' Create a new order detail for the specific product.
    newItem = Order_Detail.CreateOrder_Detail( _
            order.OrderID, selectedProduct.ProductID, 10, 5, 0)

    ' Add the new item with a link to the related order.
    context.AddRelatedObject(order, "Order_Details", newItem)

    ' Add the new order detail to the collection, and
    ' set the reference to the product.
    order.Order_Details.Add(newItem)
    newItem.Order = order
    newItem.Product = selectedProduct

    ' Send the changes to the data service.
    Dim response As DataServiceResponse = context.SaveChanges()

    ' Enumerate the returned responses.
    For Each change As ChangeOperationResponse In response
        ' Get the descriptor for the entity.
        Dim descriptor = TryCast(change.Descriptor, EntityDescriptor)

        If Not descriptor Is Nothing Then
            If TypeOf descriptor.Entity Is Order_Detail Then
                Dim addedItem = TryCast(descriptor.Entity, Order_Detail)

                If Not addedItem Is Nothing Then
                    Console.WriteLine("New {0} item added to order {1}.", _
                        addedItem.Product.ProductName, addedItem.OrderID.ToString())
                End If
            End If
        End If
    Next
Catch ex As DataServiceQueryException
    Throw New ApplicationException( _
            "An error occurred when saving changes.", ex)

    ' Handle any errors that may occur during insert, such as
    ' a constraint violation.
Catch ex As DataServiceRequestException
    Throw New ApplicationException( _
            "An error occurred when saving changes.", ex)
```

See Also

WCF Data Services Client Library

How to: Attach an Existing Entity to the DataServiceContext

How to: Define Entity Relationships

Batching Operations

How to: Define Entity Relationships (WCF Data Services)

5/2/2018 • 6 minutes to read • Edit Online

When you add a new entity in WCF Data Services, any relationships between the new entity and related entities are not automatically defined. You can create and change relationships between entity instances and have the client library reflect those changes in the data service. For more information, see Updating the Data Service.

The example in this topic uses the Northwind sample data service and autogenerated client data service classes. This service and the client data classes are created when you complete the WCF Data Services quickstart.

Example

The following example creates a new object instance and then calls the AddRelatedObject method on the DataServiceContext to create the item in the context along with the link to the related order. An HTTP POST message is sent to the data service when the SaveChanges method is called.

```
int productId = 25;
string customerId = "ALFKI";

Order_Detail newItem = null;

// Create the DataServiceContext using the service URI.
NorthwindEntities context = new NorthwindEntities(svcUri);

try
{
    // Get the specific product.
    var selectedProduct = (from product in context.Products
                           where product.ProductID == productId
                           select product).Single();

    // Get the specific customer.
    var cust = (from customer in context.Customers.Expand("Orders")
                where customer.CustomerID == customerId
                select customer).Single();

    // Get the first order.
    Order order = cust.Orders.FirstOrDefault();

    // Create a new order detail for the specific product.
    newItem = Order_Detail.CreateOrder_Detail(
        order.OrderID, selectedProduct.ProductID, 10, 5, 0);

    // Add the new item with a link to the related order.
    context.AddRelatedObject(order, "Order_Details", newItem);

    // Add the new order detail to the collection, and
    // set the reference to the product.
    order.Order_Details.Add(newItem);
    newItem.Order = order;
    newItem.Product = selectedProduct;

    // Send the changes to the data service.
    DataServiceResponse response = context.SaveChanges();

    // Enumerate the returned responses.
    foreach (ChangeOperationResponse change in response)
    {
        // Get the descriptor for the entity.
        EntityDescriptor descriptor = change.Descriptor as EntityDescriptor;

        if (descriptor != null)
        {
            if (descriptor.Entity.GetType() == typeof(Order_Detail))
            {
                Order_Detail addedItem = descriptor.Entity as Order_Detail;

                if (addedItem != null)
                {
                    Console.WriteLine("New {0} item added to order {1}.",
                        addedItem.Product.ProductName, addedItem.OrderID.ToString());
                }
            }
        }
    }
}
catch (DataServiceQueryException ex)
{
    throw new ApplicationException(
        "An error occurred when saving changes.", ex);
}

// Handle any errors that may occur during insert, such as
// a constraint violation.
catch (DataServiceRequestException ex)
{
    throw new ApplicationException(
        "An error occurred when saving changes.", ex);
}
```

```
Dim productId = 25
Dim customerId = "ALFKI"

Dim newItem As Order_Detail = Nothing

' Create the DataServiceContext using the service URI.
Dim context = New NorthwindEntities(svcUri)

Try
    ' Get the specific product.
    Dim selectedProduct = (From product In context.Products _
                           Where product.ProductID = productId _
                           Select product).Single()

    ' Get the specific customer.
    Dim cust = (From customer In context.Customers.Expand("Orders") _
                Where customer.CustomerID = customerId _
                Select customer).Single()

    ' Get the first order.
    Dim order = cust.Orders.FirstOrDefault()

    ' Create a new order detail for the specific product.
    newItem = Order_Detail.CreateOrder_Detail( _
            order.OrderID, selectedProduct.ProductID, 10, 5, 0)

    ' Add the new item with a link to the related order.
    context.AddRelatedObject(order, "Order_Details", newItem)

    ' Add the new order detail to the collection, and
    ' set the reference to the product.
    order.Order_Details.Add(newItem)
    newItem.Order = order
    newItem.Product = selectedProduct

    ' Send the changes to the data service.
    Dim response As DataServiceResponse = context.SaveChanges()

    ' Enumerate the returned responses.
    For Each change As ChangeOperationResponse In response
        ' Get the descriptor for the entity.
        Dim descriptor = TryCast(change.Descriptor, EntityDescriptor)

        If Not descriptor Is Nothing Then
            If TypeOf descriptor.Entity Is Order_Detail Then
                Dim addedItem = TryCast(descriptor.Entity, Order_Detail)

                If Not addedItem Is Nothing Then
                    Console.WriteLine("New {0} item added to order {1}.", _
                        addedItem.Product.ProductName, addedItem.OrderID.ToString())
                End If
            End If
        End If
    Next
Catch ex As DataServiceQueryException
    Throw New ApplicationException( _
            "An error occurred when saving changes.", ex)

    ' Handle any errors that may occur during insert, such as
    ' a constraint violation.
Catch ex As DataServiceRequestException
    Throw New ApplicationException( _
            "An error occurred when saving changes.", ex)
```

Example

The following example shows how to use the AddObject method to add an `Order_Details` object to a related `Orders` object with a reference to a specific `Products` object. The AddLink and SetLink methods define the relationships. In this example, the navigation properties on the `Order_Details` object are also explicitly set.

```
int productId = 25;
string customerId = "ALFKI";

Order_Detail newItem = null;

// Create the DataServiceContext using the service URI.
NorthwindEntities context = new NorthwindEntities(svcUri);

try
{
    // Get the specific product.
    var selectedProduct = (from product in context.Products
                           where product.ProductID == productId
                           select product).Single();

    // Get the specific customer.
    var cust = (from customer in context.Customers.Expand("Orders")
                where customer.CustomerID == customerId
                select customer).Single();

    // Get the first order.
    Order order = cust.Orders.FirstOrDefault();

    // Create a new order detail for the specific product.
    newItem = Order_Detail.CreateOrder_Detail(
        order.OrderID, selectedProduct.ProductID, 10, 5, 0);

    // Add the new order detail to the context.
    context.AddToOrder_Details(newItem);

    // Add links for the one-to-many relationships.
    context.AddLink(order, "Order_Details", newItem);
    context.AddLink(selectedProduct, "Order_Details", newItem);

    // Add the new order detail to the collection, and
    // set the reference to the product.
    order.Order_Details.Add(newItem);
    newItem.Product = selectedProduct;

    // Send the changes to the data service.
    DataServiceResponse response = context.SaveChanges();

    // Enumerate the returned responses.
    foreach (ChangeOperationResponse change in response)
    {
        // Get the descriptor for the entity.
        EntityDescriptor descriptor = change.Descriptor as EntityDescriptor;

        if (descriptor != null)
        {
            if (descriptor.Entity.GetType() == typeof(Order_Detail))
            {
                Order_Detail addedItem = descriptor.Entity as Order_Detail;

                if (addedItem != null)
                {
                    Console.WriteLine("New {0} item added to order {1}.",
                        addedItem.Product.ProductName, addedItem.OrderID.ToString());
                }
            }
        }
    }
}
```

```
catch (DataServiceQueryException ex)
{
    throw new ApplicationException(
        "An error occurred when saving changes.", ex);
}

// Handle any errors that may occur during insert, such as
// a constraint violation.
catch (DataServiceRequestException ex)
{
    throw new ApplicationException(
        "An error occurred when saving changes.", ex);
}
```

```
Dim productId = 25
Dim customerId = "ALFKI"

Dim newItem As Order_Detail = Nothing

' Create the DataServiceContext using the service URI.
Dim context = New NorthwindEntities(svcUri)

Try
    ' Get the specific product.
    Dim selectedProduct = (From product In context.Products _
                                           Where product.ProductID = productId _
                                           Select product).Single()

    ' Get the specific customer.
    Dim cust = (From customer In context.Customers.Expand("Orders") _
                Where customer.CustomerID = customerId _
                Select customer).Single()

    ' Get the first order.
    Dim order = cust.Orders.FirstOrDefault()

    ' Create a new order detail for the specific product.
    newItem = Order_Detail.CreateOrder_Detail( _
    order.OrderID, selectedProduct.ProductID, 10, 5, 0)

    ' Add the new order detail to the context.
    context.AddToOrder_Details(newItem)

    ' Add links for the one-to-many relationships.
    context.AddLink(order, "Order_Details", newItem)
    context.AddLink(selectedProduct, "Order_Details", newItem)

    ' Add the new order detail to the collection, and
    ' set the reference to the product.
    order.Order_Details.Add(newItem)
    newItem.Product = selectedProduct

    ' Send the insert to the data service.
    Dim response As DataServiceResponse = context.SaveChanges()

    ' Enumerate the returned responses.
    For Each change As ChangeOperationResponse In response
        ' Get the descriptor for the entity.
        Dim descriptor = TryCast(change.Descriptor, EntityDescriptor)

        If Not descriptor Is Nothing Then

            Dim addedProduct = TryCast(descriptor.Entity, Product)

            If Not addedProduct Is Nothing Then
                Console.WriteLine("New product added with ID {0}.", _
                    addedProduct.ProductID)
            End If
        End If
    Next
Catch ex As DataServiceQueryException
    Throw New ApplicationException( _
            "An error occurred when saving changes.", ex)

    ' Handle any errors that may occur during insert, such as
    ' a constraint violation.
Catch ex As DataServiceRequestException
    Throw New ApplicationException( _
            "An error occurred when saving changes.", ex)
```

See Also

WCF Data Services Client Library

How to: Add, Modify, and Delete Entities

How to: Attach an Existing Entity to the DataServiceContext (WCF Data Services)

5/2/2018 • 2 minutes to read • Edit Online

When an entity already exists in a data service, the WCF Data Services client library enables you to attach an object that represents the entity directly to the DataServiceContext without first executing a query. For more information, see Updating the Data Service.

The example in this topic uses the Northwind sample data service and autogenerated client data service classes. This service and the client data classes are created when you complete the WCF Data Services quickstart.

Example

The following example shows how to create an existing `Customer` object that contains changes to be saved to the data service. The object is attached to the context and the UpdateObject method is called to mark the attached object as Modified before the SaveChanges method is called.

```
// Create the DataServiceContext using the service URI.
NorthwindEntities context = new NorthwindEntities(svcUri);

// Define an existing customer to attach, including the key.
Customer customer =
    Customer.CreateCustomer("ALFKI", "Alfreds Futterkiste");

// Set current property values.
customer.Address = "Obere Str. 57";
customer.City = "Berlin";
customer.PostalCode = "12209";
customer.Country = "Germany";

// Set property values to update.
customer.ContactName = "Peter Franken";
customer.ContactTitle = "Marketing Manager";
customer.Phone = "089-0877310";
customer.Fax = "089-0877451";

try
{
    // Attach the existing customer to the context and mark it as updated.
    context.AttachTo("Customers", customer);
    context.UpdateObject(customer);

    // Send updates to the data service.
    context.SaveChanges();
}
catch (DataServiceClientException ex)
{
    throw new ApplicationException(
        "An error occurred when saving changes.", ex);
}
```

```
' Create the DataServiceContext using the service URI.
Dim context = New NorthwindEntities(svcUri)

' Define an existing customer to attach, including the key.
Dim customer As Customer = _
    customer.CreateCustomer("ALFKI", "Alfreds Futterkiste")

' Set current property values.
customer.Address = "Obere Str. 57"
customer.City = "Berlin"
customer.PostalCode = "12209"
customer.Country = "Germany"

' Set property values to update.
customer.ContactName = "Peter Franken"
customer.ContactTitle = "Marketing Manager"
customer.Phone = "089-0877310"
customer.Fax = "089-0877451"

Try
    ' Attach the existing customer to the context and mark it as updated.
    context.AttachTo("Customers", customer)
    context.UpdateObject(customer)

    ' Send updates to the data service.
    context.SaveChanges()
Catch ex As DataServiceClientException
    Throw New ApplicationException( _
            "An error occurred when saving changes.", ex)
End Try
```

See Also

WCF Data Services Client Library

Asynchronous Operations (WCF Data Services)

8/31/2018 • 2 minutes to read • Edit Online

Web applications must accommodate higher latency between client and server than applications that run inside internal networks. To optimize the performance and user experience of your application, we recommend using the asynchronous methods of the DataServiceContext and DataServiceQuery<TElement> classes when accessing WCF Data Services servers over the Web.

Although the WCF Data Services servers process HTTP requests asynchronously, some methods of the WCF Data Services client libraries are synchronous and wait until the entire request-response exchange is completed before continuing execution. The asynchronous methods of the WCF Data Services client libraries do not wait for this exchange to complete and can allow your application to maintain a responsive user interface in the meantime.

You can perform asynchronous operations by using a pair of methods on the DataServiceContext and DataServiceQuery<TElement> classes that start with *Begin* and *End* respectively. The *Begin* methods register a delegate that the service calls when the operation is complete. The *End* methods should be called in the delegate that is registered to handle the callback from the completed operations. When you call the *End* method to complete an asynchronous operation, you must do so from the same DataServiceQuery<TElement> or DataServiceContext instance that you used to begin the operation. Each *Begin* method takes a `state` parameter that can pass a state object to the callback. This state object is retrieved from the IAsyncResult that is supplied with the callback and is used to call the corresponding *End* method to complete the asynchronous operation. For example, when you supply the DataServiceQuery<TElement> instance as the `state` parameter when you call the BeginExecute method on the instance, the same DataServiceQuery<TElement> instance is returned by the IAsyncResult. This instance of DataServiceQuery<TElement> is then used to call the EndExecute method to complete the query operation. For more information, see How to: Execute Asynchronous Data Service Queries.

NOTE

Only asynchronous operations are supported by the client libraries that are provided in the .NET Framework for Silverlight. For more information, see WCF Data Services (Silverlight).

The .NET Framework client libraries support the following asynchronous operations:

OPERATION	METHODS
Executing a DataServiceQuery<TElement>.	- BeginExecute - EndExecute
Executing a query from the DataServiceContext.	- BeginExecute - EndExecute
Executing a batch query from the DataServiceContext.	- BeginExecuteBatch - EndExecuteBatch
Loading a related entity into the DataServiceContext.	- BeginLoadProperty - EndLoadProperty
Saving changes to objects in the DataServiceContext	- BeginSaveChanges - EndSaveChanges

Threading Considerations for Asynchronous Operations

In a multi-threaded application, the delegate that is registered as a callback for the asynchronous operation is not necessarily invoked on the same thread that was used to call the *Begin* method, which creates the initial request. In an application where the callback must be invoked on a specific thread, you must explicitly marshal the execution of the *End* method, which handles the response, to the desired thread. For example, in Windows Presentation Foundation (WPF)-based applications and Silverlight-based applications, the response must be marshaled back to the UI thread by using the BeginInvoke method on the Dispatcher object. For more information, see Querying the Data Service (WCF Data Services/Silverlight).

See Also

WCF Data Services Client Library

How to: Execute Asynchronous Data Service Queries (WCF Data Services)

5/2/2018 • 2 minutes to read • Edit Online

By using the WCF Data Services client library, you can asynchronously perform client-server operations, such as executing queries and saving changes. For more information, see Asynchronous Operations.

NOTE

In an application where the callback must be invoked on a specific thread, you must explicitly marshal the execution of the EndExecute method. For more information, see Asynchronous Operations.

The example in this topic uses the Northwind sample data service and autogenerated client data service classes. This service and the client data classes are created when you complete the WCF Data Services quickstart.

Example

The following example shows how to execute an asynchronous query by calling the BeginExecute method to start the query. The inline delegate calls the EndExecute method to display the query results.

```
public static void BeginExecuteCustomersQuery()
{
    // Create the DataServiceContext using the service URI.
    NorthwindEntities context = new NorthwindEntities(svcUri);

    // Define the query to execute asynchronously that returns
    // all customers with their respective orders.
    DataServiceQuery<Customer> query = (DataServiceQuery<Customer>)(from cust in
context.Customers.Expand("Orders")
                                       where cust.CustomerID == "ALFKI"
                                       select cust);

    try
    {
        // Begin query execution, supplying a method to handle the response
        // and the original query object to maintain state in the callback.
        query.BeginExecute(OnCustomersQueryComplete, query);
    }
    catch (DataServiceQueryException ex)
    {
        throw new ApplicationException(
            "An error occurred during query execution.", ex);
    }
}

// Handle the query callback.
private static void OnCustomersQueryComplete(IAsyncResult result)
{
    // Get the original query from the result.
    DataServiceQuery<Customer> query =
        result as DataServiceQuery<Customer>;

    foreach (Customer customer in query.EndExecute(result))
    {
        Console.WriteLine("Customer Name: {0}", customer.CompanyName);
        foreach (Order order in customer.Orders)
        {
            Console.WriteLine("Order #: {0} - Freight $: {1}",
                order.OrderID, order.Freight);
        }
    }
}
```

```
Public Shared Sub BeginExecuteCustomersQuery()
    ' Create the DataServiceContext using the service URI.
    Dim context = New NorthwindEntities(svcUri)

    ' Define the delegate to callback into the process
    Dim callback As AsyncCallback = AddressOf OnCustomersQueryComplete

    ' Define the query to execute asynchronously that returns
    ' all customers with their respective orders.
    Dim query As DataServiceQuery(Of Customer) = _
    context.Customers.Expand("Orders")

    Try
        ' Begin query execution, supplying a method to handle the response
        ' and the original query object to maintain state in the callback.
        query.BeginExecute(callback, query)
    Catch ex As DataServiceQueryException
        Throw New ApplicationException( _
                "An error occurred during query execution.", ex)
    End Try
End Sub
' Handle the query callback.
Private Shared Sub OnCustomersQueryComplete(ByVal result As IAsyncResult)
    ' Get the original query from the result.
    Dim query As DataServiceQuery(Of Customer) = _
        CType(result.AsyncState, DataServiceQuery(Of Customer))

    ' Complete the query execution.
    For Each customer As Customer In query.EndExecute(result)
        Console.WriteLine("Customer Name: {0}", customer.CompanyName)
        For Each order As Order In customer.Orders
            Console.WriteLine("Order #: {0} - Freight $: {1}", _
                    order.OrderID, order.Freight)
        Next
    Next
End Sub
```

See Also

WCF Data Services Client Library

How to: Create an Asynchronous Windows Presentation Framework Application (WCF Data Services)

5/2/2018 • 4 minutes to read • Edit Online

With WCF Data Services, you can bind data obtained from a data service to UI element of a Windows Presentation Framework (WPF) application. For more information, see Binding Data to Controls.You can also execute operations against the data service in an asynchronous manner, which enables the application to continue to respond while waiting for a response to a data service request. Applications for Silverlight are required to access the data service asynchronously. For more information, see Asynchronous Operations.

This topic shows how to access a data service asynchronously and bind the results to elements of a WPF application. The examples in this topic use the Northwind sample data service and autogenerated client data service classes. This service and the client data classes are created when you complete the WCF Data Services quickstart.

Example

The following XAML defines the window of the WPF application.

```
    <Window x:Class="CustomerOrdersAsync"
            xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
            xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
            Height="423" Width="679" Loaded="Window_Loaded" >
    <Grid Name="LayoutRoot">
        <StackPanel Orientation="Vertical" Height="Auto" Width="Auto">
            <Label Content="Customer ID" Margin="20,0,0,0" />
            <ComboBox Name="customerIDComboBox" DisplayMemberPath="CustomerID" ItemsSource="{Binding}"
                  IsSynchronizedWithCurrentItem="True" SelectedIndex="0" Height="23" Width="120"
                  HorizontalAlignment="Left" Margin="20,0,0,0" VerticalAlignment="Center" />
            <ListView ItemsSource="{Binding Path=Orders}" Name="ordersDataGrid" Margin="34,46,34,50">
                <ListView.View>
                    <GridView AllowsColumnReorder="False" ColumnHeaderToolTip="Line Items">
                        <GridViewColumn DisplayMemberBinding="{Binding Path=OrderID, Mode=OneWay}"
                        Header="Order ID" Width="50"/>
                        <GridViewColumn DisplayMemberBinding="{Binding Path=OrderDate, Mode=TwoWay}"
                        Header="Order Date" Width="50"/>
                        <GridViewColumn DisplayMemberBinding="{Binding Path=Freight, Mode=TwoWay}"
                        Header="Freight Cost" Width="50"/>
                    </GridView>
                </ListView.View>
            </ListView>
            <Button Name="saveChangesButton" Content="Save Changes" Click="saveChangesButton_Click"
                Width="80" Height="30" Margin="450,0,0,0"/>
        </StackPanel>
    </Grid>
</Window>
```

Example

The following code-behind page for the XAML file executes an asynchronous query by using the data service and binds the results to elements in the WPF window.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
using NorthwindClient.Northwind;
using System.Data.Services.Client;
using System.Windows.Threading;

namespace NorthwindClient
{
    /// <summary>
    /// Interaction logic for OrderItems.xaml
    /// </summary>
    public partial class CustomerOrdersAsync : Window
    {
        private NorthwindEntities context;
        private DataServiceCollection<Customer> customerBinding;
        private const string customerCountry = "Germany";

        // Change this URI to the service URI for your implementation.
        private const string svcUri = "http://localhost:12345/Northwind.svc/";

        // Delegate that returns void for the query result callback.
        private delegate void OperationResultCallback();

        public CustomerOrdersAsync()
        {
            InitializeComponent();
        }

        private void Window_Loaded(object sender, RoutedEventArgs e)
        {
            // Initialize the context.
            context = new NorthwindEntities(new Uri(svcUri));

            // Define a query that returns customers and orders for a specific country.
            DataServiceQuery<Customer> query = context.Customers.Expand("Orders")
                .AddQueryOption("filter", "Country eq '" + customerCountry + "'");

            try
            {
                // Begin asynchronously saving changes using the
                // specified handler and query object state.
                query.BeginExecute(OnQueryCompleted, query);
            }
            catch (DataServiceClientException ex)
            {
                MessageBox.Show(ex.ToString());
            }
        }

        private void OnQueryCompleted(IAsyncResult result)
        {
            // Get the original query object from the state cache.
            DataServiceQuery<Customer> query =
                   (DataServiceQuery<Customer>)result.AsyncState;


            // Use the Dispatcher to ensure that the query returns in the UI thread.
            this.Dispatcher.BeginInvoke(new OperationResultCallback(delegate
            {
                try
                {
                    // Instantiate the binding collection using the
                    // results of the query execution.
                    customerBinding = new DataServiceCollection<Customer>(
                        query.EndExecute(result));

                    // Bind the collection to the root element of the UI.
                    this.LayoutRoot.DataContext = customerBinding;
                }
                catch (DataServiceRequestException ex)
                {
                    MessageBox.Show(ex.ToString());
                }
            }),null);
        }
        private void saveChangesButton_Click(object sender, RoutedEventArgs e)
        {
            try
            {
                // Start the asynchronous call to save changes.
                context.BeginSaveChanges(OnSaveChangesCompleted, null);
            }
            catch (DataServiceClientException ex)
            {
                MessageBox.Show(ex.ToString());
            }
        }
        private void OnSaveChangesCompleted(IAsyncResult result)
        {
            // Use the Dispatcher to ensure that the operation returns in the UI thread.
            this.Dispatcher.BeginInvoke(new OperationResultCallback(delegate
            {
                try
                {
                    // Complete the save changes operation.
                    context.EndSaveChanges(result);
                }
                catch (DataServiceRequestException ex)
                {
                    MessageBox.Show(ex.ToString());
                }
            }), null);
        }
    }
}
```

```
Imports System
Imports System.Collections.Generic
Imports System.Linq
Imports System.Text
Imports System.Windows
Imports System.Windows.Controls
Imports System.Windows.Data
Imports System.Windows.Documents
Imports System.Windows.Input
Imports System.Windows.Media
Imports System.Windows.Media.Imaging
Imports System.Windows.Navigation
Imports System.Windows.Shapes
Imports NorthwindClient.Northwind
Imports System.Data.Services.Client
Imports System.Windows.Threading

'/ <summary>
'/ Interaction logic for OrderItems.xaml
'/ </summary>
Partial Public Class CustomerOrdersAsync
    Inherits Window

    Dim context As NorthwindEntities
    Private Shared customerBinding As DataServiceCollection(Of Customer)
    Private Const customerCountry As String = "Germany"

    ' Change this URI to the service URI for your implementation.
    Private Const svcUri As String = "http://localhost:12345/Northwind.svc/"

    ' Define a persisted result.
    Private Shared currentResult As IAsyncResult

    Delegate Sub DispatcherDelegate()

    Private Sub Window_Loaded(ByVal sender As Object, ByVal e As RoutedEventArgs)
        ' Initialize the context.
        context = New NorthwindEntities(New Uri(svcUri))

        ' Define the delegate to callback into the process
        Dim callback As AsyncCallback = AddressOf OnQueryCompleted

        ' Define a query that returns customers and orders for a specific country.
        Dim query As DataServiceQuery(Of Customer) = _
        context.Customers.Expand("Orders") _
        .AddQueryOption("filter", "Country eq '" + customerCountry + "'")

        Try
            ' Begin asynchronously saving changes Imports the
            ' specified handler and query object state.
            query.BeginExecute(callback, query)

        Catch ex As DataServiceClientException
            MessageBox.Show(ex.ToString())
        End Try
    End Sub
    Private Sub OnQueryCompleted(ByVal result As IAsyncResult)
        ' Persist the query result for the delegate.
        currentResult = result

        ' Use the Dispatcher to ensure that the
        ' asynchronous call returns in the correct thread.
        Dispatcher.BeginInvoke(New DispatcherDelegate(AddressOf QueryCompletedByDispatcher))
    End Sub
    ' Handle the query callback.
    Private Sub QueryCompletedByDispatcher()
        Try
            ' Get the original query back from the result.
            Dim query = CType(currentResult.AsyncState, DataServiceQuery(Of Customer))

            ' Instantiate the binding collection imports the
            ' results of the query execution.
            customerBinding = New DataServiceCollection(Of Customer)( _
                        query.EndExecute(currentResult))

            ' Bind the collection to the root element of the UI.
            Me.LayoutRoot.DataContext = customerBinding
        Catch ex As DataServiceRequestException
            MessageBox.Show(ex.ToString())
        End Try
    End Sub
    Private Sub saveChangesButton_Click(ByVal sender As Object, ByVal e As RoutedEventArgs)
        ' Define the delegate to callback into the process
        Dim callback As AsyncCallback = AddressOf OnSaveChangesCompleted

        Try
            ' Start the asynchronous call to save changes.
            context.BeginSaveChanges(callback, Nothing)
        Catch ex As DataServiceClientException
            MessageBox.Show(ex.ToString())
        End Try
    End Sub
    Private Sub OnSaveChangesCompleted(ByVal result As IAsyncResult)
        ' Persist the query result for the delegate.
        currentResult = result

        ' Use the Dispatcher to ensure that the operation returns in the UI thread.
        Me.Dispatcher.BeginInvoke(New DispatcherDelegate(AddressOf SaveChangesCompletedByDispatcher))
        ' Handle the query callback.
    End Sub
    Private Sub SaveChangesCompletedByDispatcher()
        Try
            ' Complete the save changes operation.
            context.EndSaveChanges(currentResult)
        Catch ex As DataServiceRequestException
            MessageBox.Show(ex.ToString())
        End Try
    End Sub
End Class
```

See Also

WCF Data Services Client Library

Batching Operations (WCF Data Services)

8/31/2018 • 2 minutes to read • Edit Online

The Open Data Protocol (OData) supports batch processing of requests to an OData-based service. For more information, see OData: Batch Processing. In WCF Data Services, each operation that uses the DataServiceContext, such as executing a query or saving changes, results in a separate request being sent to the data service. In order to maintain a logical scope for sets of operations, you can explicitly define operational batches. This ensures that all operations in the batch are sent to the data service in a single HTTP request, enables the server to process the operations atomically, and reduces the number of round trips to the data service.

Batching Query Operations

To execute multiple queries in a single batch, you must create each query in the batch as a separate instance of the DataServiceRequest<TElement> class. When you create a query request in this manner, the query itself is defined as a URI, and it follows the rules for addressing resources. For more information, see Accessing Data Service Resources. The batched query requests are sent to the data service when the ExecuteBatch method is called that contains the query request objects. This method returns a DataServiceResponse object, which is a collection of QueryOperationResponse<T> objects that represent responses to individual queries in the batch, each of which contains either a collection of objects returned by the query or error information. When any single query operation in the batch fails, error information is returned in the QueryOperationResponse<T> object for the operation that failed and the remaining operations are still executed. For more information, see How to: Execute Queries in a Batch.

Batched queries can also be executed asynchronously. For more information, see Asynchronous Operations.

Batching the SaveChanges Operation

When you call the SaveChanges method, all changes that the context tracks are translated into REST-based operations that are sent as requests to the OData service. By default, these changes are not sent in a single request message. To require that all changes be sent in a single request, you must call the SaveChanges(SaveChangesOptions) method and include a value of Batch in the SaveChangesOptions enumeration that you supply to the method.

You can also asynchronously save batched changes. For more information, see Asynchronous Operations.

See Also

WCF Data Services Client Library

How to: Execute Queries in a Batch (WCF Data Services)

5/2/2018 • 3 minutes to read • Edit Online

By using the WCF Data Services client library, you can execute more than one query against the data service in a single batch. For more information, see Batching Operations.

The example in this topic uses the Northwind sample data service and autogenerated client data service classes. This service and the client data classes are created when you complete the WCF Data Services quickstart.

Example

The following example shows how to call the ExecuteBatch method to execute an array of DataServiceRequest<TElement> objects that contains queries that return both `Customers` and `Products` objects from the Northwind data service. The collection of QueryOperationResponse<T> objects in the returned DataServiceResponse is enumerated, and the collection of objects contained in each QueryOperationResponse<T> is also enumerated.

```
string customerId = "ALFKI";

// Create the DataServiceContext using the service URI.
NorthwindEntities context = new NorthwindEntities(svcUri);

// Create the separate query URI's, one that returns
// a single customer and another that returns all Products.
Uri customerUri = new Uri(svcUri.AbsoluteUri +
    "/Customers('" + customerId + "')/?$expand=Orders");
Uri productsUri = new Uri(svcUri.AbsoluteUri +
   "/Products");

// Create the query requests.
DataServiceRequest<Customer> customerQuery =
    new DataServiceRequest<Customer>(customerUri);
DataServiceRequest<Product> productsQuery =
                new DataServiceRequest<Product>(productsUri);

// Add the query requests to a batch request array.
DataServiceRequest[] batchRequests =
    new DataServiceRequest[]{customerQuery, productsQuery};

DataServiceResponse batchResponse;

try
{
    // Execute the query batch and get the response.
    batchResponse = context.ExecuteBatch(batchRequests);

    if (batchResponse.IsBatchResponse)
    {
        // Parse the batchResponse.BatchHeaders.
    }
    // Enumerate over the results of the query.
    foreach (QueryOperationResponse response in batchResponse)
    {
        // Handle an error response.
        if (response.StatusCode > 299 || response.StatusCode < 200)
        {
            Console.WriteLine("An error occurred.");
            Console.WriteLine(response.Error.Message);
        }
        else
        {
            // Find the response for the Customers query.
            if (response.Query.ElementType == typeof(Customer))
            {
                foreach (Customer customer in response)
                {
                    Console.WriteLine("Customer: {0}", customer.CompanyName);
                    foreach (Order order in customer.Orders)
                    {
                        Console.WriteLine("Order ID: {0} - Freight: {1}",
                            order.OrderID, order.Freight);
                    }
                }
            }
            // Find the response for the Products query.
            else if (response.Query.ElementType == typeof(Product))
            {
                foreach (Product product in response)
                {
                    Console.WriteLine("Product: {0}", product.ProductName);
                }
            }
        }
    }
}

// This type of error is raised when the data service returns with
// a response code < 200 or >299 in the top level element.
catch (DataServiceRequestException e)
{
    // Get the response from the exception.
    batchResponse = e.Response;

    if (batchResponse.IsBatchResponse)
    {
        // Parse the batchResponse.BatchHeaders.
    }

    foreach (QueryOperationResponse response in batchResponse)
    {
        if (response.Error != null)
        {
            Console.WriteLine("An error occurred.");
            Console.WriteLine(response.Error.Message);
        }
    }
}
```

```
Dim customerId = "ALFKI"

' Create the DataServiceContext using the service URI.
Dim context = New NorthwindEntities(svcUri)

' Create the separate query URI's, one that returns
' a single customer and another that returns all Products.
Dim customerUri = New Uri(svcUri.AbsoluteUri & _
    "/Customers('" & customerId & "')/?$expand=Orders")
Dim productsUri = New Uri(svcUri.AbsoluteUri & _
       "/Products")

' Create the query requests.
Dim customerQuery = New DataServiceRequest(Of Customer)(customerUri)
Dim productsQuery = New DataServiceRequest(Of Product)(productsUri)
```

```
' Add the query requests to a batch request array.
Dim batchRequests = _
    New DataServiceRequest() {customerQuery, productsQuery}

Dim batchResponse As DataServiceResponse

Try
    ' Execute the query batch and get the response.
    batchResponse = context.ExecuteBatch(batchRequests)

    If batchResponse.IsBatchResponse Then
        ' Parse the batchResponse.BatchHeaders.
    End If

    ' Enumerate over the results of the query.
    For Each response As QueryOperationResponse In batchResponse
        ' Handle an error response.
        If response.StatusCode > 299 OrElse response.StatusCode < 200 Then
            Console.WriteLine("An error occurred.")
            Console.WriteLine(response.Error.Message)
        Else
            ' Find the response for the Customers query.
            If response.Query.ElementType Is GetType(Customer) Then
                For Each customer As Customer In response
                    Console.WriteLine("Customer: {0}", customer.CompanyName)
                    For Each order As Order In customer.Orders
                        Console.WriteLine("Order ID: {0} - Freight: {1}", _
                                order.OrderID, order.Freight)
                    Next
                Next
                ' Find the response for the Products query.
            ElseIf response.Query.ElementType Is GetType(Product) Then
                For Each product As Product In response
                    Console.WriteLine("Product: {0}", product.ProductName)
                Next
            End If
        End If
    Next
    ' This type of error is raised when the data service returns with
    ' a response code < 200 or >299 in the top level element.
Catch ex As DataServiceRequestException
    ' Get the response from the exception.
    batchResponse = ex.Response

    If (batchResponse.IsBatchResponse) Then
        ' Parse the batchResponse.BatchHeaders.
    End If
    For Each response As QueryOperationResponse In batchResponse
        If response.Error IsNot Nothing Then
            Console.WriteLine("An error occurred.")
            Console.WriteLine(response.Error.Message)
        End If
    Next
End Try
```

See Also

WCF Data Services Client Library

Binding Data to Controls (WCF Data Services)

6/19/2018 • 11 minutes to read • Edit Online

With WCF Data Services, you can bind controls such as the `ComboBox` and `ListView` controls to an instance of the DataServiceCollection<T> class. This collection, which inherits from the ObservableCollection<T> class, contains the data from an Open Data Protocol (OData) feed. This class represents a dynamic data collection that provides notifications when items get added or removed. When you use an instance of DataServiceCollection<T> for data binding, the WCF Data Services client libraries handle these events to ensure that objects tracked by the DataServiceContext remain synchronized with the data in the bound UI element.

The DataServiceCollection<T> class (indirectly) implements the INotifyCollectionChanged interface to alert the context when objects are added to or removed from the collection. Data service type objects used with a DataServiceCollection<T> must also implement the INotifyPropertyChanged interface to alert the DataServiceCollection<T> when properties of objects in the binding collection have changed.

NOTE

When you use the **Add Service Reference** dialog or the DataSvcUtil.exe tool with the `/dataservicecollection` option to generate the client data service classes, the generated data classes implement the INotifyPropertyChanged interface. For more information, see How to: Manually Generate Client Data Service Classes.

Creating the Binding Collection

Create a new instance of the DataServiceCollection<T> class by calling one of the class constructor methods with a supplied DataServiceContext instance and optionally a DataServiceQuery<TElement> or LINQ query that, when it is executed, returns an IEnumerable<T> instance. This IEnumerable<T> provides the source of objects for the binding collection, which are materialized from an OData feed. For more information, see Object Materialization. By default, changes made to bound objects and items inserted into the collection are automatically tracked by the DataServiceContext. If you need to manually track these changes, call one of the constructor methods that takes a `trackingMode` parameter and specify a value of None.

The following example shows how to create an instance of DataServiceCollection<T> based on a supplied DataServiceContext and a DataServiceQuery<TElement> that returns all customers with related orders:

```
// Create a new collection that contains all customers and related orders.
DataServiceCollection<Customer> trackedCustomers =
    new DataServiceCollection<Customer>(context.Customers.Expand("Orders"));
```

```
' Create a new collection that contains all customers and related orders.
Dim trackedCustomers As DataServiceCollection(Of Customer) = _
        New DataServiceCollection(Of Customer)(context.Customers.Expand("Orders"))
```

Binding Data to Windows Presentation Foundation Elements

Because the DataServiceCollection<T> class inherits from the ObservableCollection<T> class, you can bind objects to an element, or control, in a Windows Presentation Foundation (WPF) application as you would when using the ObservableCollection<T> class for binding. For more information, see Data Binding (Windows Presentation Foundation). One way to bind data service data to WPF controls is to set the `DataContext` property of the element to the instance of the DataServiceCollection<T> class that contains the query result. In this case, you use the ItemsSource property to set the object source for the control. Use the DisplayMemberPath property to specify which property of the bound object to display. If you are binding an element to a related object that is returned by a navigation property, include the path in the binding defined for the ItemsSource property. This path is relative to the root object set by the DataContext property of the parent control. The following example sets the DataContext property of a StackPanel element to bind the parent control to an DataServiceCollection<T> of customer objects:

```
// Create a LINQ query that returns customers with related orders.
var customerQuery = from cust in context.Customers.Expand("Orders")
                    where cust.Country == customerCountry
                    select cust;

// Create a new collection for binding based on the LINQ query.
trackedCustomers = new DataServiceCollection<Customer>(customerQuery,
    TrackingMode.AutoChangeTracking,"Customers",
    OnPropertyChanged, OnCollectionChanged);

// Bind the root StackPanel element to the collection;
// related object binding paths are defined in the XAML.
this.LayoutRoot.DataContext = trackedCustomers;
```

```
// Create a LINQ query that returns customers with related orders.
var customerQuery = from cust in context.Customers.Expand("Orders")
                    where cust.Country == customerCountry
                    select cust;

// Create a new collection for binding based on the LINQ query.
trackedCustomers = new DataServiceCollection<Customer>(customerQuery);

// Bind the root StackPanel element to the collection;
// related object binding paths are defined in the XAML.
LayoutRoot.DataContext = trackedCustomers;
```

```
' Create a LINQ query that returns customers with related orders.
Dim customerQuery = From cust In context.Customers.Expand("Orders") _
                        Where cust.Country = customerCountry _
                        Select cust

' Create a new collection for binding based on the LINQ query.
trackedCustomers = New DataServiceCollection(Of Customer)(customerQuery, _
        TrackingMode.AutoChangeTracking, "Customers", _
        AddressOf OnMyPropertyChanged, AddressOf OnMyCollectionChanged)

' Bind the root StackPanel element to the collection
' related object binding paths are defined in the XAML.
Me.LayoutRoot.DataContext = trackedCustomers
```

```
' Create a LINQ query that returns customers with related orders.
Dim customerQuery = From cust In context.Customers.Expand("Orders") _
                        Where cust.Country = customerCountry _
                        Select cust

    ' Create a new collection for binding based on the LINQ query.
trackedCustomers = New DataServiceCollection(Of Customer)(customerQuery)

    ' Bind the root StackPanel element to the collection
    ' related object binding paths are defined in the XAML.
Me.LayoutRoot.DataContext = trackedCustomers
```

```
' Create a LINQ query that returns customers with related orders.
Dim customerQuery = From cust In context.Customers.Expand("Orders") _
                        Where cust.Country = customerCountry _
                        Select cust

' Create a new collection for binding based on the LINQ query.
trackedCustomers = New DataServiceCollection(Of Customers)(customerQuery, _
        TrackingMode.AutoChangeTracking, "Customers", _
        AddressOf OnMyPropertyChanged, AddressOf OnMyCollectionChanged)

    ' Bind the root StackPanel element to the collection
    ' related object binding paths are defined in the XAML.
    Me.LayoutRoot.DataContext = trackedCustomers
    Me.LayoutRoot.UpdateLayout()
```

The following example shows the XAML binding definition of the child DataGrid and ComboBox controls:

```
<StackPanel Orientation="Vertical" Height="Auto" Name="LayoutRoot" Width="Auto">
    <Label Content="Customer ID" Margin="20,0,0,0" />
    <ComboBox Name="customerIDComboBox" DisplayMemberPath="CustomerID" ItemsSource="{Binding}"
              IsSynchronizedWithCurrentItem="True" SelectedIndex="0" Height="23" Width="120"
              HorizontalAlignment="Left" Margin="20,0,0,0" VerticalAlignment="Center" />
    <ListView ItemsSource="{Binding Path=Orders}" Name="ordersDataGrid" Margin="34,46,34,50">
        <ListView.View>
            <GridView AllowsColumnReorder="False" ColumnHeaderToolTip="Line Items">
                <GridViewColumn DisplayMemberBinding="{Binding Path=OrderID, Mode=OneWay}"
                    Header="Order ID" Width="50"/>
                <GridViewColumn DisplayMemberBinding="{Binding Path=OrderDate, Mode=TwoWay}"
                    Header="Order Date" Width="50"/>
                <GridViewColumn DisplayMemberBinding="{Binding Path=Freight, Mode=TwoWay}"
                    Header="Freight Cost" Width="50"/>
            </GridView>
        </ListView.View>
    </ListView>
    <Button Name="saveChangesButton" Content="Save Changes" Click="saveChangesButton_Click"
            Width="80" Height="30" Margin="450,0,0,0"/>
</StackPanel>
```

For more information, see How to: Bind Data to Windows Presentation Foundation Elements.

When an entity participates in a one-to-many or many-to-many relationship, the navigation property for the relationship returns a collection of related objects. When you use the **Add Service Reference** dialog box or the DataSvcUtil.exe tool to generate the client data service classes, the navigation property returns an instance of DataServiceCollection<T>. This enables you to bind related objects to a control, and support common WPF binding scenarios, such as the master-detail binding pattern for related entities. In the previous XAML example, the XAML code binds the master DataServiceCollection<T> to the root data element. The order DataGrid is then bound to the Orders DataServiceCollection<T> returned from the selected Customers object, which is in turn bound to the root data element of the Window.

Binding Data to Windows Forms Controls

To bind objects to a Windows Form control, set the `DataSource` property of the control to the instance of the DataServiceCollection<T> class that contains the query result.

NOTE

Data binding is only supported for controls that listen for change events by implementing the INotifyCollectionChanged and INotifyPropertyChanged interfaces. When a control does not support this kind of change notification, changes that are made to the underlying DataServiceCollection<T> are not reflected in the bound control.

The following example binds an DataServiceCollection<T> to a ComboBox control:

```
// Create a new collection for binding based on the LINQ query.
trackedCustomers = new DataServiceCollection<Customer>(customerQuery);

//Bind the Customers combobox to the collection.
customersComboBox.DisplayMember = "CustomerID";
customersComboBox.DataSource = trackedCustomers;
```

```
' Create a new collection for binding based on the LINQ query.
trackedCustomers = New DataServiceCollection(Of Customer)(customerQuery)

'Bind the Customers combobox to the collection.
customersComboBox.DisplayMember = "CustomerID"
customersComboBox.DataSource = trackedCustomers
```

When you use the **Add Service Reference** dialog to generate the client data service classes, a project data source is also created that is based on the generated DataServiceContext. With this data source, you can create UI elements or controls that display data from the data service simply by dragging items from the **Data Sources** window onto the designer. These items become elements in the application UI that are bound to the data source. For more information, see How to: Bind Data Using a Project Data Source.

Binding Paged Data

A data service can be configured to limit the amount of queried data that is returned in a single response message. For more information, see Configuring the Data Service. When the data service is paging response data, each response contains a link that is used to return the next page of results. For more information, see Loading Deferred Content. In this case, you must explicitly load pages by calling the Load method on the DataServiceCollection<T> by passing the URI obtained from the NextLinkUri property, as in the following example:

```
// Create a new collection for binding based on the LINQ query.
trackedCustomers = new DataServiceCollection<Customer>(customerQuery);

// Load all pages of the response at once.
while (trackedCustomers.Continuation != null)
{
    trackedCustomers.Load(
        context.Execute<Customer>(trackedCustomers.Continuation.NextLinkUri));
}
```

```
    ' Create a new collection for binding based on the LINQ query.
trackedCustomers = New DataServiceCollection(Of Customer)(customerQuery)

    ' Load all pages of the response at once.
While trackedCustomers.Continuation IsNot Nothing
    trackedCustomers.Load( _
            context.Execute(Of Customer)(trackedCustomers.Continuation.NextLinkUri))
End While
```

Related objects are loaded in a similar manner. For more information, see How to: Bind Data to Windows Presentation Foundation Elements.

Customizing Data Binding Behaviors

The DataServiceCollection<T> class enables you to intercept the events raised when changes are made to the collection, such as an object being added or removed, and when changes are made to the properties of object in a collection. You can modify the data binding events to override the default behavior, which includes the following constraints:

- No validation is performed within the delegates.
- Adding an entity automatically adds related entities.
- Deleting an entity does not delete the related entities.

When you create a new instance of DataServiceCollection<T>, you have the option to specify the following parameters that define delegates to methods that handle the events raised when bound objects are changed:

- `entityChanged` - a method that is called when the property of a bound object is changed. This Func<T,TResult> delegate accepts an EntityChangedParams object and returns a Boolean value that indicates whether the default behavior, to call UpdateObject on the DataServiceContext, should still occur.
- `entityCollectionChanged` - a method that is called when an object is added or removed from the binding collection. This Func<T,TResult> delegate accepts an EntityCollectionChangedParams object and returns a Boolean value that indicates whether the default behavior, to call AddObject for an Add action or DeleteObject for a Remove action on the DataServiceContext, should still occur.

NOTE

WCF Data Services performs no validation of the custom behaviors that you implement in these delegates.

In the following example, the Remove action is customized to call the DeleteLink and DeleteObject method to remove `Orders_Details` entities that belong to a deleted `Orders` entity. This custom action is performed because dependent entities are not automatically deleted when the parent entity is deleted.

```
// Method that is called when the CollectionChanged event is handled.
private bool OnCollectionChanged(
    EntityCollectionChangedParams entityCollectionChangedinfo)
{
    if (entityCollectionChangedinfo.Action ==
        NotifyCollectionChangedAction.Remove)
    {
        // Delete the related items when an order is deleted.
        if (entityCollectionChangedinfo.TargetEntity.GetType() == typeof(Order))
        {
            // Get the context and object from the supplied parameter.
            DataServiceContext context = entityCollectionChangedinfo.Context;
            Order deletedOrder = entityCollectionChangedinfo.TargetEntity as Order;

            if (deletedOrder.Order_Details.Count == 0)
            {
                // Load the related OrderDetails.
                context.LoadProperty(deletedOrder, "Order_Details");
            }

            // Delete the order and its related items;
            foreach (Order_Detail item in deletedOrder.Order_Details)
            {
                context.DeleteObject(item);
            }

            // Delete the order and then return true since the object is already deleted.
            context.DeleteObject(deletedOrder);

            return true;
        }
        else
        {
            return false;
        }
    }
    else
    {
        // Use the default behavior.
        return false;
    }
}
```

```vb
' Method that is called when the CollectionChanged event is handled.
Private Function OnMyCollectionChanged( _
    ByVal entityCollectionChangedinfo As EntityCollectionChangedParams) As Boolean
    If entityCollectionChangedinfo.Action = _
        NotifyCollectionChangedAction.Remove Then

        ' Delete the related items when an order is deleted.
        If entityCollectionChangedinfo.TargetEntity.GetType() Is GetType(Order) Then

            ' Get the context and object from the supplied parameter.
            Dim context = entityCollectionChangedinfo.Context
            Dim deletedOrder As Order = _
            CType(entityCollectionChangedinfo.TargetEntity, Order)

            If deletedOrder.Order_Details.Count = 0 Then
                ' Load the related OrderDetails.
                context.LoadProperty(deletedOrder, "Order_Details")
            End If

            ' Delete the order and its related items
            For Each item As Order_Detail In deletedOrder.Order_Details
                context.DeleteObject(item)
            Next

            ' Delete the order and then return false since the object is already deleted.
            context.DeleteObject(deletedOrder)

            Return True
        Else
            Return False
        End If
    Else
        ' Use the default behavior.
        Return False
    End If
End Function
```

```
' Method that is called when the CollectionChanged event is handled.
Private Function OnMyCollectionChanged( _
    ByVal entityCollectionChangedinfo As EntityCollectionChangedParams) As Boolean
    If entityCollectionChangedinfo.Action = _
        NotifyCollectionChangedAction.Remove Then

        ' Delete the related items when an order is deleted.
        If entityCollectionChangedinfo.TargetEntity.GetType() Is GetType(Orders) Then

            ' Get the context and object from the supplied parameter.
            Dim context = entityCollectionChangedinfo.Context
            Dim deletedOrder As Orders = _
            CType(entityCollectionChangedinfo.TargetEntity, Orders)

            ' Load the related OrderDetails.
            context.LoadProperty(deletedOrder, "Order_Details")

            ' Delete the order and its related items
            For Each item As Order_Details In deletedOrder.Order_Details
                'context.DeleteLink(deletedOrder, "Order_Details", item)
                context.DeleteObject(item)
            Next

            ' Delete the order and then return false since the object is already deleted.
            context.DeleteObject(deletedOrder)

            Return False
        Else
            Return True
        End If
    Else
        ' Use the default behavior.
        Return True
    End If
End Function
```

For more information, see How to: Customize Data Binding Behaviors.

The default behavior when an object is removed from a DataServiceCollection<T> by using the Remove method is that the object is also marked as deleted in the DataServiceContext. To change this behavior, you can specify a delegate to a method in the `entityCollectionChanged` parameter that is called when the CollectionChanged event occurs.

Data Binding with Custom Client Data Classes

To be able to load objects into a DataServiceCollection<T>, the objects themselves must implement the INotifyPropertyChanged interface. Data service client classes that are generated when you use the **Add Service Reference** dialog box or the DataSvcUtil.exe tool implement this interface. If you provide your own client data classes, you must use another type of collection for data binding. When objects change, you must handle events in the data bound controls to call the following methods of the DataServiceContext class:

- AddObject - when a new object is added to the collection.
- DeleteObject - when an object is removed from the collection.
- UpdateObject - when a property is changed on an object in the collection.
- AddLink - when an object is added to a collection of related object.
- SetLink - when an object is added to a collection of related objects.

For more information, see Updating the Data Service.

See Also

How to: Manually Generate Client Data Service Classes

How to: Add a Data Service Reference

How to: Bind Data to Windows Presentation Foundation Elements (WCF Data Services)

5/2/2018 • 6 minutes to read • Edit Online

With WCF Data Services, you can bind Windows Presentation Foundation (WPF) elements such as a ListBox``or ComboBox to an instance of DataServiceCollection<T>, which handles the events raised by the controls to keep the DataServiceContext synchronized with changes made to data in the controls. For more information, see Binding Data to Controls.

The example in this topic uses the Northwind sample data service and autogenerated client data service classes. This service and the client data classes are created when you complete the WCF Data Services quickstart.

Example

The following example is from the code-behind page for an Extensible Application Markup Language (XAML) page that defines the `SalesOrders` window in WPF. When the window is loaded, a DataServiceCollection<T> is created based on the result of a query that returns customers, filtered by country. All of the pages of this paged result are loaded, along with the related orders, and are bound to the DataContext property of the StackPanel that is the root layout control for the WPF window. For more information, see Loading Deferred Content.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
using System.Data.Services.Client;
using NorthwindClient.Northwind;

namespace NorthwindClient
{
    public partial class CustomerOrdersWpf3 : Window
    {
        private NorthwindEntities context;
        private DataServiceCollection<Customer> trackedCustomers;
        private const string customerCountry = "Germany";
        private const string svcUri = "http://localhost:12345/Northwind.svc/";

        public CustomerOrdersWpf3()
        {
            InitializeComponent();
        }

        private void Window_Loaded(object sender, RoutedEventArgs e)
        {
            try
            {
                // Initialize the context for the data service.
                context = new NorthwindEntities(new Uri(svcUri));
                // Create a LINQ query that returns customers with related orders.
                var customerQuery = from cust in context.Customers
                                    where cust.Country == customerCountry
                                    select cust;

                // Create a new collection for binding based on the LINQ query.
                trackedCustomers = new DataServiceCollection<Customer>(customerQuery);

                // Load all pages of the response at once.
                while (trackedCustomers.Continuation != null)
                {
                    trackedCustomers.Load(
                        context.Execute<Customer>(trackedCustomers.Continuation.NextLinkUri));
                }

                // Bind the root StackPanel element to the collection;
                // related object binding paths are defined in the XAML.
                LayoutRoot.DataContext = trackedCustomers;
            }
            catch (DataServiceQueryException ex)
            {
                MessageBox.Show("The query could not be completed:\n" + ex.ToString());
            }
            catch (InvalidOperationException ex)
            {
                MessageBox.Show("The following error occurred:\n" + ex.ToString());
            }
        }

        private void customerIDComboBox_SelectionChanged(object sender, SelectionChangedEventArgs e)
        {
            Customer selectedCustomer =
                this.customerIDComboBox.SelectedItem as Customer;

            try
            {
                // Load the first page of related orders for the selected customer.
                context.LoadProperty(selectedCustomer, "Orders");

                // Load all remaining pages.
                while (selectedCustomer.Orders.Continuation != null)
                {
                    selectedCustomer.Orders.Load(
                        context.Execute<Order>(selectedCustomer.Orders.Continuation.NextLinkUri));
                }
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.ToString());
            }
        }

        private void saveChangesButton_Click(object sender, RoutedEventArgs e)
        {
            try
            {
                // Save changes to the data service.
                context.SaveChanges();
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.ToString());
            }
        }
    }
}
```

```
Imports System
Imports System.Collections.Generic
Imports System.Linq
Imports System.Text
Imports System.Windows
Imports System.Windows.Controls
Imports System.Windows.Data
Imports System.Windows.Documents
Imports System.Windows.Input
Imports System.Windows.Media
Imports System.Windows.Media.Imaging
Imports System.Windows.Navigation
Imports System.Windows.Shapes
Imports System.Data.Services.Client
Imports NorthwindClient.Northwind

Partial Public Class CustomerOrdersWpf3
    Inherits Window

    Private context As NorthwindEntities
    Private trackedCustomers As DataServiceCollection(Of Customer)
    Private Const customerCountry As String = "Germany"
    Private Const svcUri As String = "http://localhost:12345/Northwind.svc/"

    Private Sub Window_Loaded(ByVal sender As Object, ByVal e As RoutedEventArgs)
        Try
            ' Initialize the context for the data service.
            context = New NorthwindEntities(New Uri(svcUri))

            ' Create a LINQ query that returns customers with related orders.
            Dim customerQuery = From cust In context.Customers _
                                Where cust.Country = customerCountry _
                                Select cust

                ' Create a new collection for binding based on the LINQ query.
            trackedCustomers = New DataServiceCollection(Of Customer)(customerQuery)

                ' Load all pages of the response at once.
            While trackedCustomers.Continuation IsNot Nothing
                trackedCustomers.Load( _
                        context.Execute(Of Customer)(trackedCustomers.Continuation.NextLinkUri))
            End While

            ' Bind the root StackPanel element to the collection
            ' related object binding paths are defined in the XAML.
            Me.LayoutRoot.DataContext = trackedCustomers
        Catch ex As DataServiceQueryException
            MessageBox.Show("The query could not be completed:\n" + ex.ToString())
        Catch ex As InvalidOperationException
            MessageBox.Show("The following error occurred:\n" + ex.ToString())
        End Try
    End Sub
    Private Sub customerIDComboBox_SelectionChanged(ByVal sender As Object, ByVal e As
SelectionChangedEventArgs)
        Dim selectedCustomer As Customer = _
            CType(Me.customerIDComboBox.SelectedItem, Customer)
        Try
            If selectedCustomer.Orders.Count = 0 Then
                ' Load the first page of related orders for the selected customer.
                context.LoadProperty(selectedCustomer, "Orders")
            End If

            ' Load all remaining pages.
            While selectedCustomer.Orders.Continuation IsNot Nothing
                selectedCustomer.Orders.Load( _
                        context.Execute(Of Order)(selectedCustomer.Orders.Continuation.NextLinkUri))
            End While
        Catch ex As Exception
            MessageBox.Show(ex.ToString())
        End Try
    End Sub
    Private Sub saveChangesButton_Click(ByVal sender As Object, ByVal e As RoutedEventArgs)
        Try
            ' Save changes to the data service.
            context.SaveChanges()
        Catch ex As Exception
            MessageBox.Show(ex.ToString())
        End Try
    End Sub
End Class
```

Example

The following XAML defines the `SalesOrders` window in WPF for the previous example.

```
    <Window x:Class="CustomerOrdersWpf3"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
             Height="423" Width="679" Loaded="Window_Loaded">
    <StackPanel Orientation="Vertical" Height="Auto" Name="LayoutRoot" Width="Auto">
        <Label Content="Customer ID" Margin="20,0,0,0" />
        <ComboBox Name="customerIDComboBox" DisplayMemberPath="CustomerID" ItemsSource="{Binding}"
                  SelectionChanged="customerIDComboBox_SelectionChanged"
                  IsSynchronizedWithCurrentItem="True" SelectedIndex="0" Height="23" Width="120"
                  HorizontalAlignment="Left" Margin="20,0,0,0" VerticalAlignment="Center" />
        <ListView ItemsSource="{Binding Path=Orders}" Name="ordersDataGrid" Margin="34,46,34,50">
            <ListView.View>
                <GridView AllowsColumnReorder="False" ColumnHeaderToolTip="Line Items">
                    <GridViewColumn DisplayMemberBinding="{Binding Path=OrderID, Mode=OneWay}"
                        Header="Order ID" Width="50"/>
                    <GridViewColumn DisplayMemberBinding="{Binding Path=OrderDate, Mode=TwoWay}"
                        Header="Order Date" Width="50"/>
                    <GridViewColumn DisplayMemberBinding="{Binding Path=Freight, Mode=TwoWay}"
                        Header="Freight Cost" Width="50"/>
                </GridView>
            </ListView.View>
        </ListView>
        <Button Name="saveChangesButton" Content="Save Changes" Click="saveChangesButton_Click"
                Width="80" Height="30" Margin="450,0,0,0"/>
    </StackPanel>
</Window>
```

Example

The following example is from the code-behind page for an Extensible Application Markup Language (XAML) page that defines the `SalesOrders` window in WPF. When the window is loaded, a DataServiceCollection<T> is created based on the result of a query that returns customers with related objects, filtered by country. This result is bound to the DataContext property of the StackPanel that is the root layout control for the WPF window.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
using System.Data.Services.Client;
using NorthwindClient.Northwind;

namespace NorthwindClient
{
    public partial class CustomerOrdersWpf : Window
    {
        private NorthwindEntities context;
        private DataServiceCollection<Customer> trackedCustomers;
        private const string customerCountry = "Germany";
        private const string svcUri = "http://localhost:12345/Northwind.svc/";

        public CustomerOrdersWpf()
        {
            InitializeComponent();
        }

        private void Window_Loaded(object sender, RoutedEventArgs e)
        {
            try
            {
                // Initialize the context for the data service.
                context = new NorthwindEntities(new Uri(svcUri));

                // Create a LINQ query that returns customers with related orders.
                var customerQuery = from cust in context.Customers.Expand("Orders")
                                    where cust.Country == customerCountry
                                    select cust;

                // Create a new collection for binding based on the LINQ query.
                trackedCustomers = new DataServiceCollection<Customer>(customerQuery);

                // Bind the root StackPanel element to the collection;
                // related object binding paths are defined in the XAML.
                LayoutRoot.DataContext = trackedCustomers;
            }
            catch (DataServiceQueryException ex)
            {
                MessageBox.Show("The query could not be completed:\n" + ex.ToString());
            }
            catch (InvalidOperationException ex)
            {
                MessageBox.Show("The following error occurred:\n" + ex.ToString());
            }
        }
        private void saveChangesButton_Click(object sender, RoutedEventArgs e)
        {
            // Save changes to the data service.
            context.SaveChanges();
        }
    }
}
```

```
Imports System
Imports System.Collections.Generic
Imports System.Linq
Imports System.Text
Imports System.Windows
Imports System.Windows.Controls
Imports System.Windows.Data
Imports System.Windows.Documents
Imports System.Windows.Input
Imports System.Windows.Media
Imports System.Windows.Media.Imaging
Imports System.Windows.Navigation
Imports System.Windows.Shapes
Imports System.Data.Services.Client
Imports NorthwindClient.Northwind

Partial Public Class CustomerOrdersWpf
    Inherits Window

    Private context As NorthwindEntities
    Private trackedCustomers As DataServiceCollection(Of Customer)
    Private Const customerCountry As String = "Germany"
    Private Const svcUri As String = "http://localhost:12345/Northwind.svc/"
    Private Sub Window_Loaded(ByVal sender As Object, ByVal e As RoutedEventArgs)
        Try
            ' Initialize the context for the data service.
            context = New NorthwindEntities(New Uri(svcUri))

            ' Create a LINQ query that returns customers with related orders.
            Dim customerQuery = From cust In context.Customers.Expand("Orders") _
                                Where cust.Country = customerCountry _
                                Select cust

                ' Create a new collection for binding based on the LINQ query.
            trackedCustomers = New DataServiceCollection(Of Customer)(customerQuery)

                ' Bind the root StackPanel element to the collection
                ' related object binding paths are defined in the XAML.
            Me.LayoutRoot.DataContext = trackedCustomers
        Catch ex As DataServiceQueryException
            MessageBox.Show("The query could not be completed:\n" + ex.ToString())
        Catch ex As InvalidOperationException
            MessageBox.Show("The following error occurred:\n" + ex.ToString())
        End Try
    End Sub
    Private Sub saveChangesButton_Click(ByVal sender As Object, ByVal e As RoutedEventArgs)
        ' Save changes to the data service.
        context.SaveChanges()
    End Sub
End Class
```

Example

The following XAML defines the `SalesOrders` window in WPF for the previous example.

```
    <Window x:Class="CustomerOrdersWpf"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
             Height="423" Width="679" Loaded="Window_Loaded">
    <StackPanel Orientation="Vertical" Height="Auto" Name="LayoutRoot" Width="Auto">
        <Label Content="Customer ID" Margin="20,0,0,0" />
        <ComboBox Name="customerIDComboBox" DisplayMemberPath="CustomerID" ItemsSource="{Binding}"
                  IsSynchronizedWithCurrentItem="True" SelectedIndex="0" Height="23" Width="120"
                  HorizontalAlignment="Left" Margin="20,0,0,0" VerticalAlignment="Center" />
        <ListView ItemsSource="{Binding Path=Orders}" Name="ordersDataGrid" Margin="34,46,34,50">
            <ListView.View>
                <GridView AllowsColumnReorder="False" ColumnHeaderToolTip="Line Items">
                    <GridViewColumn DisplayMemberBinding="{Binding Path=OrderID, Mode=OneWay}"
                        Header="Order ID" Width="50"/>
                    <GridViewColumn DisplayMemberBinding="{Binding Path=OrderDate, Mode=TwoWay}"
                        Header="Order Date" Width="50"/>
                    <GridViewColumn DisplayMemberBinding="{Binding Path=Freight, Mode=TwoWay}"
                        Header="Freight Cost" Width="50"/>
                </GridView>
            </ListView.View>
        </ListView>
        <Button Name="saveChangesButton" Content="Save Changes" Click="saveChangesButton_Click"
                Width="80" Height="30" Margin="450,0,0,0"/>
    </StackPanel>
</Window>
```

How to: Bind Data Using a Project Data Source (WCF Data Services)

8/29/2018 • 6 minutes to read • Edit Online

You can create data sources that are based on the generated data objects in an WCF Data Services client application. When you add a reference to a data service by using the **Add Service Reference** dialog, a project data source is created along with the generated client data classes. One data source is created for each entity set that the data service exposes. You can create forms that display data from the service by dragging these data source items from the **Data Sources** window onto the designer. These items become controls that are bound to the data source. During execution, this data source is bound to an instance of the DataServiceCollection<T> class, which is filled with objects that are returned by a query to the data service. For more information, see Binding Data to Controls.

The examples in this topic use the Northwind sample data service and autogenerated client data service classes. This service and the client data classes are created when you complete the WCF Data Services quickstart.

Use a project data source in a WPF window

1. In Visual Studio, in a WPF project, add a reference to the Northwind data service. For more information, see How to: Add a Data Service Reference.

2. In the **Data Sources** window, expand the `Customers` node in the **NorthwindEntities** project data source.

3. Click the **CustomerID** item, select **ComboBox** from the list, and drag the **CustomerID** item from the **Customers** node to the designer.

   This creates the following object elements in the XAML file for the window:

   - A CollectionViewSource element named `customersViewSource`. The DataContext property of the top-level Grid object element is set to this new CollectionViewSource.

   - A data-bound ComboBox named `CustomerID`.

   - A Label.

4. Drag the **Orders** navigation property to the designer.

   This creates the following additional object elements in the XAML file for the window:

   - A second CollectionViewSource element named `customersOrdersViewSource`, the source of which is the `customerViewSource`.

   - A data-bound DataGrid control named `ordersDataGrid`.

5. (Optional) Drag additional items from the **Customers** node to the designer.

6. Open the code page for the form and add the following `using` statements (`Imports` in Visual Basic):

```
using System.Data.Services.Client;
using NorthwindClient.Northwind;
```

7. In the partial class that defines the form, add the following code that creates an ObjectContext instance and defines the `customerID` constant.

```
private NorthwindEntities context;
private CollectionViewSource customersViewSource;
private DataServiceCollection<Customer> trackedCustomers;
private const string customerCountry = "Germany";
private const string svcUri = "http://localhost:12345/Northwind.svc/";
```

```
Private context As NorthwindEntities
Private customersViewSource As CollectionViewSource
Private trackedCustomers As DataServiceCollection(Of Customer)
Private Const customerCountry As String = "Germany"
Private Const svcUri As String = "http://localhost:12345/Northwind.svc/"
```

8. In the designer, select the window.

> **NOTE**
>
> Make sure that you select the window itself, rather than selecting content that is within the window. If the window is selected, the **Name** text box near the top of the **Properties** window should contain the name of the window.

9. In the **Properties** window, select the **Events** button.

10. Find the **Loaded** event, and then double-click the drop-down list next to this event.

    Visual Studio opens the code-behind file for the window and generates a Loaded event handler.

11. In the newly created Loaded event handler, copy and paste the following code.

```
// Initialize the context for the data service.
context = new NorthwindEntities(new Uri(svcUri));

// Create a LINQ query that returns customers with related orders.
var  customerQuery = from cust in context.Customers.Expand("Orders")
                     where cust.Country == customerCountry
                     select cust;

// Create a new collection for binding based on the LINQ query.
trackedCustomers = new DataServiceCollection<Customer>(customerQuery);

try
{
    // Get the customersViewSource resource and set the binding to the collection.
    customersViewSource =
        ((CollectionViewSource)(this.FindResource("customersViewSource")));
    customersViewSource.Source = trackedCustomers;
    customersViewSource.View.MoveCurrentToFirst();
}
catch (DataServiceQueryException ex)
{
    MessageBox.Show("The query could not be completed:\n" + ex.ToString());
}
catch (InvalidOperationException ex)
{
    MessageBox.Show("The following error occurred:\n" + ex.ToString());
}
```

```
' Initialize the context for the data service.
context = New NorthwindEntities(New Uri(svcUri))

' Create a LINQ query that returns customers with related orders.
Dim customerQuery = From cust In context.Customers.Expand("Orders") _
                         Where cust.Country = customerCountry _
                         Select cust

' Create a new collection for binding based on the LINQ query.
trackedCustomers = New DataServiceCollection(Of Customer)(customerQuery)

    try
        ' Get the customersViewSource resource and set the binding to the collection.
    customersViewSource = _
        CType(Me.FindResource("customersViewSource"), CollectionViewSource)
    customersViewSource.Source = trackedCustomers
    customersViewSource.View.MoveCurrentToFirst()
Catch ex As DataServiceQueryException
    MessageBox.Show("The query could not be completed:\n" + ex.ToString())
Catch ex As InvalidOperationException
    MessageBox.Show("The following error occurred:\n" + ex.ToString())
End Try
```

12. This code creates an instance of DataServiceCollection<T> for the `Customers` type based on the execution of a LINQ query that returns an IEnumerable<T> of `Customers` along with related `Orders` objects from the Northwind data service and binds it to the `customersViewSource`.

Use a project data source in a Windows form

1. In the **Data Sources** window, expand the **Customers** node in the **NorthwindEntities** project data source.

2. Click the **CustomerID** item, select **ComboBox** from the list, and drag the **CustomerID** item from the **Customers** node to the designer.

   This creates the following controls on the form:

   - An instance of BindingSource named `customersBindingSource`.
   - An instance of BindingNavigator named `customersBindingNavigator`. You can delete this control as it will not be needed.
   - A data-bound ComboBox named `CustomerID`.
   - A Label.

3. Drag the **Orders** navigation property to the form.

4. This creates the `ordersBindingSource` control with the DataSource property of the control set to the `customersBindingSource` and the DataMember property set to `Customers`. It also creates the `ordersDataGridView` data-bound control on the form, accompanied by an appropriately titled label control.

5. (Optional) Drag additional items from the **Customers** node to the designer.

6. Open the code page for the form and add the following `using` statements (`Imports` in Visual Basic):

```
using System.Data.Services.Client;
using NorthwindClient.Northwind;
```

```
Imports System.Data.Services.Client
Imports NorthwindClient.Northwind
```

7. In the partial class that defines the form, add the following code that creates an ObjectContext instance and defines the `customerID` constant.

```
private NorthwindEntities context;
private DataServiceCollection<Customer> trackedCustomers;
private const string customerCountry = "Germany";
private const string svcUri = "http://localhost:12345/Northwind.svc/";
```

```
Private context As NorthwindEntities
Private trackedCustomers As DataServiceCollection(Of Customer)
Private Const customerCountry As String = "Germany"
Private Const svcUri As String = "http:'localhost:12345/Northwind.svc/"
```

8. In the form designer, double-click the form.

   This opens the code page for the form and creates the method that handles the `Load` event for the form.

9. In the `Load` event handler, copy and paste the following code.

```
// Initialize the context for the data service.
context = new NorthwindEntities(new Uri(svcUri));
try
{
    // Create a LINQ query that returns customers with related orders.
    var customerQuery = from cust in context.Customers.Expand("Orders")
                        where cust.Country == customerCountry
                        select cust;

    // Create a new collection for binding based on the LINQ query.
    trackedCustomers = new DataServiceCollection<Customer>(customerQuery);

    //Bind the Customers combobox to the collection.
    customersComboBox.DisplayMember = "CustomerID";
    customersComboBox.DataSource = trackedCustomers;
}
catch (DataServiceQueryException ex)
{
    MessageBox.Show("The query could not be completed:\n" + ex.ToString());
}
catch (InvalidOperationException ex)
{
    MessageBox.Show("The following error occurred:\n" + ex.ToString());
}
```

```
' Initialize the context for the data service.
context = New NorthwindEntities(New Uri(svcUri))
Try
    ' Create a LINQ query that returns customers with related orders.
    Dim customerQuery = From cust In context.Customers.Expand("Orders") _
                        Where cust.Country = customerCountry _
                        Select cust

    ' Create a new collection for binding based on the LINQ query.
    trackedCustomers = New DataServiceCollection(Of Customer)(customerQuery)

    'Bind the Customers combobox to the collection.
    customersComboBox.DisplayMember = "CustomerID"
    customersComboBox.DataSource = trackedCustomers
Catch ex As DataServiceQueryException
    MessageBox.Show("The query could not be completed:\n" + ex.ToString())
Catch ex As InvalidOperationException
    MessageBox.Show("The following error occurred:\n" + ex.ToString())
```

10. This code creates an instance of DataServiceCollection<T> for the `Customers` type based on the execution of a DataServiceQuery<TElement> that returns an IEnumerable<T> of `Customers` from the Northwind data service and binds it to the `customersBindingSource`.

See Also

- WCF Data Services Client Library
- How to: Bind Data to Windows Presentation Foundation Elements

How to: Customize Data Binding Behaviors (WCF Data Services)

5/2/2018 • 8 minutes to read • Edit Online

With WCF Data Services, you can supply custom logic that is called by the DataServiceCollection<T> when an object is added or removed from the binding collection or when a property change is detected. This custom logic is provided as methods, referenced as Func<T,TResult> delegates, that return a value of `false` when the default behavior should still be performed when the custom method completes and `true` when subsequent processing of the event should be stopped.

The examples in this topic supply custom methods for both the `entityChanged` and `entityCollectionChanged` parameters of DataServiceCollection<T>. The examples in this topic use the Northwind sample data service and autogenerated client data service classes. This service and the client data classes are created when you complete the WCF Data Services quickstart.

Example

The following code-behind page for the XAML file creates a DataServiceCollection<T> with custom methods that are called when changes occur to data that is bound to the binding collection. When the CollectionChanged event occurs, the supplied method prevents an item that has been removed from the binding collection from being deleted from the data service. When the PropertyChanged event occurs, the `ShipDate` value is validated to ensure that changes are not made to orders that have already shipped.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
using System.Data.Services.Client;
using NorthwindClient.Northwind;
using System.Collections.Specialized;

namespace NorthwindClient
{
    public partial class CustomerOrdersCustom : Window
    {
        private NorthwindEntities context;
        private DataServiceCollection<Customer> trackedCustomers;
        private const string customerCountry = "Germany";
        private const string svcUri = "http://localhost:12345/Northwind.svc/";

        public CustomerOrdersCustom()
        {
            InitializeComponent();
        }

        private void Window_Loaded(object sender, RoutedEventArgs e)
        {
            try
            {
                // Initialize the context for the data service.
                context = new NorthwindEntities(new Uri(svcUri));

                // Create a LINQ query that returns customers with related orders.
                var customerQuery = from cust in context.Customers.Expand("Orders")
                                    where cust.Country == customerCountry
                                    select cust;

                // Create a new collection for binding based on the LINQ query.
                trackedCustomers = new DataServiceCollection<Customer>(customerQuery,
                    TrackingMode.AutoChangeTracking,"Customers",
                    OnPropertyChanged, OnCollectionChanged);

                // Bind the root StackPanel element to the collection;
                // related object binding paths are defined in the XAML.
                this.LayoutRoot.DataContext = trackedCustomers;
            }
            catch (DataServiceQueryException ex)
            {
                MessageBox.Show("The query could not be completed:\n" + ex.ToString());
            }
            catch (InvalidOperationException ex)
            {
                MessageBox.Show("The following error occurred:\n" + ex.ToString());
            }
        }

        // Method that is called when the CollectionChanged event is handled.
        private bool OnCollectionChanged(
            EntityCollectionChangedParams entityCollectionChangedinfo)
        {
            if (entityCollectionChangedinfo.Action ==
                NotifyCollectionChangedAction.Remove)
            {
                // Delete the related items when an order is deleted.
                if (entityCollectionChangedinfo.TargetEntity.GetType() == typeof(Order))
                {
                    // Get the context and object from the supplied parameter.
                    DataServiceContext context = entityCollectionChangedinfo.Context;
                    Order deletedOrder = entityCollectionChangedinfo.TargetEntity as Order;

                    if (deletedOrder.Order_Details.Count == 0)
                    {
                        // Load the related OrderDetails.
                        context.LoadProperty(deletedOrder, "Order_Details");
                    }

                    // Delete the order and its related items;
                    foreach (Order_Detail item in deletedOrder.Order_Details)
                    {
                        context.DeleteObject(item);
                    }

                    // Delete the order and then return true since the object is already deleted.
                    context.DeleteObject(deletedOrder);

                    return true;
                }
                else
                {
                    return false;
                }
            }
            else
            {
                // Use the default behavior.
                return false;
            }
        }

        // Method that is called when the PropertyChanged event is handled.
        private bool OnPropertyChanged(EntityChangedParams entityChangedInfo)
        {
            // Validate a changed order to ensure that changes are not made
            // after the order ships.
            if ((entityChangedInfo.Entity.GetType() == typeof(Order)) &&
                ((Order)(entityChangedInfo.Entity)).ShippedDate < DateTime.Today)
            {
                throw new ApplicationException(string.Format(
                    "The order {0} cannot be changed because it shipped on {1}.",
                    ((Order)(entityChangedInfo.Entity)).OrderID,
                    ((Order)(entityChangedInfo.Entity)).ShippedDate));
            }
            return false;
        }

        private void deleteButton_Click(object sender, RoutedEventArgs e)
        {
            if (customerIDComboBox.SelectedItem != null)
            {
                // Get the Orders binding collection.
                DataServiceCollection<Order> trackedOrders =
                    ((Customer)(customerIDComboBox.SelectedItem)).Orders;

                // Remove the currently selected order.
                trackedOrders.Remove((Order)(ordersDataGrid.SelectedItem));
            }
        }

        private void saveChangesButton_Click(object sender, RoutedEventArgs e)
        {
            try
            {
                // Save changes to the data service.
                context.SaveChanges();
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.ToString());
            }
        }
    }
}
```

```
Imports System
Imports System.Collections.Generic
Imports System.Linq
Imports System.Text
Imports System.Windows
Imports System.Windows.Controls
Imports System.Windows.Data
Imports System.Windows.Documents
Imports System.Windows.Input
Imports System.Windows.Media
Imports System.Windows.Media.Imaging
Imports System.Windows.Navigation
Imports System.Windows.Shapes
Imports System.Data.Services.Client
Imports NorthwindClient.Northwind
Imports System.Collections.Specialized

Partial Public Class CustomerOrdersCustom
    Inherits Window
    Private context As NorthwindEntities
    Private trackedCustomers As DataServiceCollection(Of Customer)
    Private Const customerCountry As String = "Germany"
    Private Const svcUri As String = "http://localhost:12345/Northwind.svc/"
    Private Sub Window_Loaded(ByVal sender As Object, ByVal e As RoutedEventArgs)
        Try
            ' Initialize the context for the data service.
            context = New NorthwindEntities(New Uri(svcUri))

            ' Create a LINQ query that returns customers with related orders.
            Dim customerQuery = From cust In context.Customers.Expand("Orders") _
                                Where cust.Country = customerCountry _
                                Select cust

            ' Create a new collection for binding based on the LINQ query.
            trackedCustomers = New DataServiceCollection(Of Customer)(customerQuery, _
                    TrackingMode.AutoChangeTracking, "Customers", _
                    AddressOf OnMyPropertyChanged, AddressOf OnMyCollectionChanged)

            ' Bind the root StackPanel element to the collection
            ' related object binding paths are defined in the XAML.
            Me.LayoutRoot.DataContext = trackedCustomers
        Catch ex As DataServiceQueryException
            MessageBox.Show("The query could not be completed:\n" + ex.ToString())
        Catch ex As InvalidOperationException
            MessageBox.Show("The following error occurred:\n" + ex.ToString())
        End Try
    End Sub
    ' Method that is called when the CollectionChanged event is handled.
    Private Function OnMyCollectionChanged( _
        ByVal entityCollectionChangedinfo As EntityCollectionChangedParams) As Boolean
        If entityCollectionChangedinfo.Action = _
            NotifyCollectionChangedAction.Remove Then

            ' Delete the related items when an order is deleted.
            If entityCollectionChangedinfo.TargetEntity.GetType() Is GetType(Order) Then

                ' Get the context and object from the supplied parameter.
                Dim context = entityCollectionChangedinfo.Context
                Dim deletedOrder As Order = _
                CType(entityCollectionChangedinfo.TargetEntity, Order)

                If deletedOrder.Order_Details.Count = 0 Then
                    ' Load the related OrderDetails.
                    context.LoadProperty(deletedOrder, "Order_Details")
                End If

                ' Delete the order and its related items
                For Each item As Order_Detail In deletedOrder.Order_Details
                    context.DeleteObject(item)
                Next

                ' Delete the order and then return false since the object is already deleted.
                context.DeleteObject(deletedOrder)

                Return True
            Else
                Return False
            End If
        Else
            ' Use the default behavior.
            Return False
        End If
    End Function
    ' Method that is called when the PropertyChanged event is handled.
    Private Function OnMyPropertyChanged( _
    ByVal entityChangedInfo As EntityChangedParams) As Boolean
        ' Validate a changed order to ensure that changes are not made
        ' after the order ships.
        If entityChangedInfo.Entity.GetType() Is GetType(Order) AndAlso _
```

```
            (CType(entityChangedInfo.Entity, Order).ShippedDate < DateTime.Today) Then
            Throw New ApplicationException(String.Format( _
                "The order {0} cannot be changed because it shipped on {1}.", _
                CType(entityChangedInfo.Entity, Order).OrderID, _
                CType(entityChangedInfo.Entity, Order).ShippedDate))
            Return False
        Else
            Return True
        End If
    End Function
    Private Sub deleteButton_Click(ByVal sender As Object, ByVal e As RoutedEventArgs)
        ' Get the Orders binding collection.
        If customerIDComboBox.SelectedItem IsNot Nothing Then
            Dim trackedOrders As DataServiceCollection(Of Order) = _
                (CType(customerIDComboBox.SelectedItem, Customer)).Orders

            ' Remove the currently selected order.
            trackedOrders.Remove(CType(ordersDataGrid.SelectedItem, Order))
        End If
    End Sub
    Private Sub saveChangesButton_Click(ByVal sender As Object, ByVal e As RoutedEventArgs)
        Try
            ' Save changes to the data service.
            context.SaveChanges()
        Catch ex As Exception
            MessageBox.Show(ex.ToString())
        End Try
    End Sub
End Class
```

```
Imports System
Imports System.Collections.Generic
Imports System.Linq
Imports System.Text
Imports System.Windows
Imports System.Windows.Controls
Imports System.Windows.Data
Imports System.Windows.Documents
Imports System.Windows.Input
Imports System.Windows.Media
Imports System.Windows.Media.Imaging
Imports System.Windows.Navigation
Imports System.Windows.Shapes
Imports System.Data.Services.Client
Imports NorthwindClient.NorthwindModel
Imports System.Collections.Specialized

Partial Public Class CustomerOrdersCustom
    Inherits Window
    Private context As NorthwindEntities
    Private trackedCustomers As DataServiceCollection(Of Customers)
    Private Const customerCountry As String = "Germany"
    Private Const svcUri As String = "http://localhost:12345/Northwind.svc/"
    Private Sub Window_Loaded(ByVal sender As Object, ByVal e As RoutedEventArgs)
        Try
            ' Initialize the context for the data service.
            context = New NorthwindEntities(New Uri(svcUri))

            ' Create a LINQ query that returns customers with related orders.
            Dim customerQuery = From cust In context.Customers.Expand("Orders") _
                                Where cust.Country = customerCountry _
                                Select cust

            ' Create a new collection for binding based on the LINQ query.
            trackedCustomers = New DataServiceCollection(Of Customers)(customerQuery, _
                    TrackingMode.AutoChangeTracking, "Customers", _
                    AddressOf OnMyPropertyChanged, AddressOf OnMyCollectionChanged)
                ' Bind the root StackPanel element to the collection
                ' related object binding paths are defined in the XAML.
                Me.LayoutRoot.DataContext = trackedCustomers
                Me.LayoutRoot.UpdateLayout()
        Catch ex As DataServiceQueryException
            MessageBox.Show("The query could not be completed:\n" + ex.ToString())
        Catch ex As InvalidOperationException
            MessageBox.Show("The following error occured:\n" + ex.ToString())
        End Try
    End Sub
    ' Method that is called when the CollectionChanged event is handled.
    Private Function OnMyCollectionChanged( _
        ByVal entityCollectionChangedinfo As EntityCollectionChangedParams) As Boolean
        If entityCollectionChangedinfo.Action = _
            NotifyCollectionChangedAction.Remove Then

            ' Delete the related items when an order is deleted.
            If entityCollectionChangedinfo.TargetEntity.GetType() Is GetType(Orders) Then

                ' Get the context and object from the supplied parameter.
                Dim context = entityCollectionChangedinfo.Context
                Dim deletedOrder As Orders = _
                CType(entityCollectionChangedinfo.TargetEntity, Orders)

                ' Load the related OrderDetails.
                context.LoadProperty(deletedOrder, "Order_Details")

                ' Delete the order and its related items
                For Each item As Order_Details In deletedOrder.Order_Details
                    'context.DeleteLink(deletedOrder, "Order_Details", item)
                    context.DeleteObject(item)
                Next

                ' Delete the order and then return false since the object is already deleted.
                context.DeleteObject(deletedOrder)

                Return False
            Else
                Return True
            End If
        Else
            ' Use the default behavior.
            Return True
        End If
    End Function
    ' Method that is called when the PropertyChanged event is handled.
    Private Function OnMyPropertyChanged( _
    ByVal entityChangedInfo As EntityChangedParams) As Boolean
        ' Validate a changed order to ensure that changes are not made
        ' after the order ships.
        If entityChangedInfo.Entity.GetType() Is GetType(Orders) AndAlso _
            (CType(entityChangedInfo.Entity, Orders).ShippedDate < DateTime.Today) Then
            Throw New ApplicationException(String.Format( _
                "The order {0} cannot be changed because it shipped on {1}.", _
                CType(entityChangedInfo.Entity, Orders).OrderID, _
                CType(entityChangedInfo.Entity, Orders).ShippedDate))
            Return True
        End If
    End Function
    Private Sub deleteButton_Click(ByVal sender As Object, ByVal e As RoutedEventArgs)
        ' Get the Orders binding collection.
        Dim trackedOrders As DataServiceCollection(Of Orders) = _
            (CType(customerIDComboBox.SelectedItem, Customers)).Orders

        ' Remove the currently selected order.
        trackedOrders.Remove(CType(ordersDataGrid.SelectedItem, Orders))
    End Sub
    Private Sub saveChangesButton_Click(ByVal sender As Object, ByVal e As RoutedEventArgs)
```

```
        Try
            ' Save changes to the data service.
            context.SaveChanges()
        Catch ex As Exception
            MessageBox.Show(ex.ToString())
        End Try
    End Sub
End Class
```

Example

The following XAML defines the window for the previous example.

```
<Window x:Class="CustomerOrdersCustom"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
             Height="423" Width="679" Loaded="Window_Loaded" >
    <StackPanel Orientation="Vertical" Height="Auto" Name="LayoutRoot" Width="Auto">
        <Label Content="Customer ID" Margin="20,0,0,0" />
        <ComboBox Name="customerIDComboBox" DisplayMemberPath="CustomerID" ItemsSource="{Binding}"
                  IsSynchronizedWithCurrentItem="True" SelectedIndex="0" Height="23" Width="120"
                  HorizontalAlignment="Left" Margin="20,0,0,0" VerticalAlignment="Center" />
        <ListView ItemsSource="{Binding Path=Orders}" Name="ordersDataGrid" Margin="34,46,34,50">
            <ListView.View>
                <GridView AllowsColumnReorder="False" ColumnHeaderToolTip="Line Items">
                    <GridViewColumn DisplayMemberBinding="{Binding Path=OrderID, Mode=OneWay}"
                        Header="Order ID" Width="50"/>
                    <GridViewColumn DisplayMemberBinding="{Binding Path=OrderDate, Mode=TwoWay}"
                        Header="Order Date" Width="50"/>
                    <GridViewColumn DisplayMemberBinding="{Binding Path=Freight, Mode=TwoWay}"
                        Header="Freight Cost" Width="50"/>
                </GridView>
            </ListView.View>
        </ListView>
        <StackPanel Orientation="Horizontal">
            <Button Name="deleteButton" Content="Delete Order" Click="deleteButton_Click"
                Width="80" Height="30" Margin="450,0,10,0"/>
            <Button Name="saveChangesButton" Content="Save Changes" Click="saveChangesButton_Click"
                Width="80" Height="30" Margin="10,0,0,0"/>
        </StackPanel>
    </StackPanel>
</Window>
```

See Also

WCF Data Services Client Library

Calling Service Operations (WCF Data Services)

8/31/2018 • 12 minutes to read • Edit Online

The Open Data Protocol (OData) defines service operations for a data service. WCF Data Services enables you to define such operations as methods on the data service. Like other data service resources, these service operations are addressed by using URIs. A service operation can return collections of entity types, single entity type instances, and primitive types, such as integer and string. A service operation can also return `null` (`Nothing` in Visual Basic). The WCF Data Services client library can be used to access service operations that support HTTP GET requests. These kinds of service operations are defined as methods that have the WebGetAttribute applied. For more information, see Service Operations.

Service operations are exposed in the metadata returned by a data service that implements the OData. In the metadata, service operations are represented as `FunctionImport` elements. When generating the strongly-typed DataServiceContext, the Add Service Reference and DataSvcUtil.exe tools ignore this element. Because of this, you will not find a method on the context that can be used to call a service operation directly. However, you can still use the WCF Data Services client to call service operations in one of these two ways:

- By calling the Execute method on the DataServiceContext, supplying the URI of the service operation, along with any parameters. This method is used to call any GET service operation.
- By using the CreateQuery method on the DataServiceContext to create a DataServiceQuery<TElement> object. When calling CreateQuery, the name of the service operation is supplied to the `entitySetName` parameter. This method returns a DataServiceQuery<TElement> object that calls the service operation when enumerated or when the Execute method is called. This method is used to call GET service operations that return a collection. A single parameter can be supplied by using the AddQueryOption method. The DataServiceQuery<TElement> object returned by this method can be further composed against like any query object. For more information, see Querying the Data Service.

Considerations for Calling Service Operations

The following considerations apply when using the WCF Data Services client to call service operations.

- When accessing the data service asynchronously, you must use the equivalent asynchronous BeginExecute/EndExecute methods on DataServiceContext or the BeginExecute/EndExecute methods on DataServiceQuery<TElement>.
- The WCF Data Services client library cannot materialize the results from a service operation that returns a collection of primitive types.
- The WCF Data Services client library does not support calling POST service operations. Service operations that are called by an HTTP POST are defined by using the WebInvokeAttribute with the `Method="POST"` parameter. To call a service operation by using an HTTP POST request, you must instead use an HttpWebRequest.
- You cannot use CreateQuery to call a GET service operation that returns a single result, of either entity or primitive type, or that requires more than one input parameter. You must instead call the Execute method.
- Consider creating an extension method on the strongly-typed DataServiceContext partial class, which is generated by the tools, that uses either the CreateQuery or the Execute method to call a service operation. This enables you to call service operations directly from the context. For more information, see the blog post Service Operations and the WCF Data Services Client.

- When you use CreateQuery to call a service operation, the client library automatically escapes characters supplied to the AddQueryOption by performing percent-encoding of reserved characters, such as ampersand (&), and escaping of single-quotes in strings. However, when you call one of the *Execute* methods to call a service operation, you must remember to perform this escaping of any user-supplied string values. Single-quotes in URIs are escaped as pairs of single-quotes.

Examples of Calling Service Operations

This section contains the following examples of how to call service operations by using the WCF Data Services client library:

- Calling Execute<T> to Return a Collection of Entities
- Using CreateQuery<T> to Return a Collection of Entities
- Calling Execute<T> to Return a Single Entity
- Calling Execute<T> to Return a Collection of Primitive Values
- Calling Execute<T> to Return a Single Primitive Value
- Calling a Service Operation that Returns No Data
- Calling a Service Operation Asynchronously

Calling Execute<T> to Return a Collection of Entities

The following example calls a service operation named GetOrdersByCity, which takes a string parameter of `city` and returns an IQueryable<T>:

```
// Define the service operation query parameter.
string city = "London";

// Define the query URI to access the service operation with specific
// query options relative to the service URI.
string queryString = string.Format("GetOrdersByCity?city='{0}'", city)
    + "&$orderby=ShippedDate desc"
    + "&$expand=Order_Details";

// Create the DataServiceContext using the service URI.
NorthwindEntities context = new NorthwindEntities(svcUri2);

try
{
    // Execute the service operation that returns all orders for the specified city.
    var results = context.Execute<Order>(new Uri(queryString, UriKind.Relative));

    // Write out order information.
    foreach (Order o in results)
    {
        Console.WriteLine(string.Format("Order ID: {0}", o.OrderID));

        foreach (Order_Detail item in o.Order_Details)
        {
            Console.WriteLine(String.Format("\tItem: {0}, quantity: {1}",
                item.ProductID, item.Quantity));
        }
    }
}
catch (DataServiceQueryException ex)
{
    QueryOperationResponse response = ex.Response;

    Console.WriteLine(response.Error.Message);
}
```

```
' Define the service operation query parameter.
Dim city As String = "London"

' Define the query URI to access the service operation with specific
' query options relative to the service URI.
Dim queryString As String = String.Format("GetOrdersByCity?city='{0}'", city) _
                            & "&$orderby=ShippedDate desc" _
                            & "&$expand=Order_Details"

' Create the DataServiceContext using the service URI.
Dim context As NorthwindEntities = New NorthwindEntities(svcUri2)

Try

    ' Execute the service operation that returns all orders for the specified city.
    Dim results = context.Execute(Of Order)(New Uri(queryString, UriKind.Relative))

    ' Write out order information.
    For Each o As Order In results
        Console.WriteLine(String.Format("Order ID: {0}", o.OrderID))
        For Each item As Order_Detail In o.Order_Details
            Console.WriteLine(String.Format(vbTab & "Item: {0}, quantity: {1}", _
                item.ProductID, item.Quantity))
        Next
    Next
Catch ex As DataServiceQueryException
    Dim response As QueryOperationResponse = ex.Response

    Console.WriteLine(response.Error.Message)
End Try
```

In this example, the service operation returns a collection of `Order` objects with related `Order_Detail` objects.

Using CreateQuery<T> to Return a Collection of Entities

The following example uses the CreateQuery to return a DataServiceQuery<TElement> that is used to call the same GetOrdersByCity service operation:

```
// Define the service operation query parameter.
string city = "London";

// Create the DataServiceContext using the service URI.
NorthwindEntities context = new NorthwindEntities(svcUri2);

// Use the CreateQuery method to create a query that accessess
// the service operation passing a single parameter.
var query = context.CreateQuery<Order>("GetOrdersByCity")
    .AddQueryOption("city", string.Format("'{0}'", city))
    .Expand("Order_Details");

try
{
    // The query is executed during enumeration.
    foreach (Order o in query)
    {
        Console.WriteLine(string.Format("Order ID: {0}", o.OrderID));

        foreach (Order_Detail item in o.Order_Details)
        {
            Console.WriteLine(String.Format("\tItem: {0}, quantity: {1}",
                item.ProductID, item.Quantity));
        }
    }
}
catch (DataServiceQueryException ex)
{
    QueryOperationResponse response = ex.Response;

    Console.WriteLine(response.Error.Message);
}
```

```
' Define the service operation query parameter.
Dim city As String = "London"

' Create the DataServiceContext using the service URI.
Dim context As NorthwindEntities = New NorthwindEntities(svcUri2)

' Use the CreateQuery method to create a query that accessess
' the service operation passing a single parameter.
Dim query = context.CreateQuery(Of Order)("GetOrdersByCity") _
        .AddQueryOption("city", String.Format("'{0}'", city)).Expand("Order_Details")

Try
    ' The query is executed during enumeration.
    For Each o As Order In query
        Console.WriteLine(String.Format("Order ID: {0}", o.OrderID))
        For Each item As Order_Detail In o.Order_Details
            Console.WriteLine(String.Format(vbTab & "Item: {0}, quantity: {1}", _
                item.ProductID, item.Quantity))
        Next
    Next
Catch ex As DataServiceQueryException
    Dim response As QueryOperationResponse = ex.Response
    Console.WriteLine(response.Error.Message)
End Try
```

In this example, the AddQueryOption method is used to add the parameter to the query, and the Expand method is used to include related Order_Details objects in the results.

Calling Execute<T> to Return a Single Entity

The following example calls a service operation named GetNewestOrder that returns only a single Order entity:

```
// Define the query URI to access the service operation,
// relative to the service URI.
string queryString = "GetNewestOrder";

// Create the DataServiceContext using the service URI.
NorthwindEntities context = new NorthwindEntities(svcUri2);

try
{
    // Execute a service operation that returns only the newest single order.
    Order order
        = (context.Execute<Order>(new Uri(queryString, UriKind.Relative)))
        .FirstOrDefault();

    // Write out order information.
    Console.WriteLine(string.Format("Order ID: {0}", order.OrderID));
    Console.WriteLine(string.Format("Order date: {0}", order.OrderDate));
}
catch (DataServiceQueryException ex)
{
    QueryOperationResponse response = ex.Response;

    Console.WriteLine(response.Error.Message);
}
```

```
' Define the query URI to access the service operation,
' relative to the service URI.
Dim queryString As String = "GetNewestOrder"

' Create the DataServiceContext using the service URI.
Dim context As NorthwindEntities = New NorthwindEntities(svcUri2)

Try
    ' Execute a service operation that returns only the newest single order.
    Dim o As Order = _
        context.Execute(Of Order)( _
            New Uri(queryString, UriKind.Relative)).FirstOrDefault()

    ' Write out order information.
    Console.WriteLine(String.Format("Order ID: {0}", o.OrderID))
    Console.WriteLine(String.Format("Order date: {0}", o.OrderDate))
Catch ex As DataServiceQueryException
    Dim response As QueryOperationResponse = ex.Response

    Console.WriteLine(response.Error.Message)
End Try
```

In this example, the FirstOrDefault method is used to request only a single Order entity on execution.

Calling Execute<T> to Return a Collection of Primitive Values

The following example calls a service operation that returns a collection of string values:

```
// Define the query URI to access the service operation,
// relative to the service URI.
string queryString = "GetCustomerNames";

// Create the DataServiceContext using the service URI.
NorthwindEntities context = new NorthwindEntities(svcUri2);

try
{
    // Execute a service operation that returns only the newest single order.
    IEnumerable<string> customerNames
        = context.Execute<string>(new Uri(queryString, UriKind.Relative));

    foreach (string name in customerNames)
    {
        // Write out order information.
        Console.WriteLine(name);
    }
}
catch (DataServiceQueryException ex)
{
    QueryOperationResponse response = ex.Response;

    Console.WriteLine(response.Error.Message);
}
```

Calling Execute<T> to Return a Single Primitive Value

The following example calls a service operation that returns a single string value:

```
// Define the query URI to access the service operation,
// relative to the service URI.
string queryString = "CountOpenOrders";

// Create the DataServiceContext using the service URI.
NorthwindEntities context = new NorthwindEntities(svcUri2);

try
{
    // Execute a service operation that returns the integer
    // count of open orders.
    int numOrders
        = (context.Execute<int>(new Uri(queryString, UriKind.Relative)))
        .FirstOrDefault();

    // Write out the number of open orders.
    Console.WriteLine(string.Format("Open orders as of {0}: {1}",
        DateTime.Today.Date, numOrders));
}
catch (DataServiceQueryException ex)
{
    QueryOperationResponse response = ex.Response;

    Console.WriteLine(response.Error.Message);
}
```

```
' Define the query URI to access the service operation,
' relative to the service URI.
Dim queryString As String = "CountOpenOrders"

' Create the DataServiceContext using the service URI.
Dim context As NorthwindEntities = New NorthwindEntities(svcUri2)

Try
    ' Execute a service operation that returns the integer
    ' count of open orders.
    Dim numOrders As Integer = context.Execute(Of Integer)( _
        New Uri(queryString, UriKind.Relative)).FirstOrDefault()

    ' Write out the number of open orders.
    Console.WriteLine(String.Format("Open orders as of {0}: {1}",
        DateTime.Today.Date, numOrders))
Catch ex As DataServiceQueryException
    Dim response As QueryOperationResponse = ex.Response

    Console.WriteLine(response.Error.Message)
End Try
```

Again in this example, the FirstOrDefault method is used to request only a single integer value on execution.

Calling a Service Operation that Returns No Data

The following example calls a service operation that returns no data:

```
// Define the query URI to access the service operation,
// relative to the service URI.
string queryString = "ReturnsNoData";

// Create the DataServiceContext using the service URI.
NorthwindEntities context = new NorthwindEntities(svcUri2);

try
{
    // Execute a service operation that returns void.
    context.Execute<string>(new Uri(queryString, UriKind.Relative));
}
catch (DataServiceQueryException ex)
{
    QueryOperationResponse response = ex.Response;

    Console.WriteLine(response.Error.Message);
}
```

```
' Define the query URI to access the service operation,
' relative to the service URI.
Dim queryString As String = "ReturnsNoData"

' Create the DataServiceContext using the service URI.
Dim context As NorthwindEntities = New NorthwindEntities(svcUri2)

Try
    ' Execute a service operation that returns void.
    context.Execute(Of String)( _
        New Uri(queryString, UriKind.Relative))
Catch ex As DataServiceQueryException
    Dim response As QueryOperationResponse = ex.Response

    Console.WriteLine(response.Error.Message)
End Try
```

Because not data is returned, the value of the execution is not assigned. The only indication that the request has succeeded is that no DataServiceQueryException is raised.

Calling a Service Operation Asynchronously

The following example calls a service operation asynchronously by calling BeginExecute and EndExecute:

```
// Define the service operation query parameter.
string city = "London";

// Define the query URI to access the service operation with specific
// query options relative to the service URI.
string queryString = string.Format("GetOrdersByCity?city='{0}'", city)
    + "&$orderby=ShippedDate desc"
    + "&$expand=Order_Details";

// Create the DataServiceContext using the service URI.
NorthwindEntities context = new NorthwindEntities(svcUri2);

// Execute the service operation that returns
// all orders for the specified city.
var results = context.BeginExecute<Order>(
    new Uri(queryString, UriKind.Relative),
    OnAsyncExecutionComplete, context);
```

```
' Define the service operation query parameter.
Dim city As String = "London"

' Define the query URI to access the service operation with specific
' query options relative to the service URI.
Dim queryString As String = String.Format("GetOrdersByCity?city='{0}'", city) _
        & "&$orderby=ShippedDate desc" _
        & "&$expand=Order_Details"

' Create the DataServiceContext using the service URI.
Dim context As NorthwindEntities = New NorthwindEntities(svcUri2)

' Define the delegate to callback into the process
Dim callback As AsyncCallback = AddressOf OnAsyncExecutionComplete

' Execute the service operation that returns
' all orders for the specified city.
Dim results = context.BeginExecute(Of Order)( _
    New Uri(queryString, UriKind.Relative), _
    callback, context)
```

```
private static void OnAsyncExecutionComplete(IAsyncResult result)
{
    // Get the context back from the stored state.
    var context = result.AsyncState as NorthwindEntities;

    try
    {
        // Complete the exection and write out the results.
        foreach (Order o in context.EndExecute<Order>(result))
        {
            Console.WriteLine(string.Format("Order ID: {0}", o.OrderID));

            foreach (Order_Detail item in o.Order_Details)
            {
                Console.WriteLine(String.Format("\tItem: {0}, quantity: {1}",
                    item.ProductID, item.Quantity));
            }
        }
    }
    catch (DataServiceQueryException ex)
    {
        QueryOperationResponse response = ex.Response;

        Console.WriteLine(response.Error.Message);
    }
}
```

```
Private Shared Sub OnAsyncExecutionComplete(ByVal result As IAsyncResult)

    ' Get the context back from the stored state.
    Dim context = TryCast(result.AsyncState, NorthwindEntities)

    Try
        ' Complete the exection and write out the results.
        For Each o As Order In context.EndExecute(Of Order)(result)
            Console.WriteLine(String.Format("Order ID: {0}", o.OrderID))

            For Each item As Order_Detail In o.Order_Details
                Console.WriteLine(String.Format(vbTab & "Item: {0}, quantity: {1}", _
                    item.ProductID, item.Quantity))
            Next
        Next
    Catch ex As DataServiceQueryException
        Dim response As QueryOperationResponse = ex.Response
        Console.WriteLine(response.Error.Message)
    End Try
End Sub
```

Because no data is returned, the value returned by the execution is not assigned. The only indication that the request has succeeded is that no DataServiceQueryException is raised.

The following example calls the same service operation asynchronously by using CreateQuery:

```
// Define the service operation query parameter.
string city = "London";

// Create the DataServiceContext using the service URI.
NorthwindEntities context = new NorthwindEntities(svcUri2);

// Use the CreateQuery method to create a query that accessess
// the service operation passing a single parameter.
var query = context.CreateQuery<Order>("GetOrdersByCity")
    .AddQueryOption("city", string.Format("'{0}'", city))
    .Expand("Order_Details");

// Execute the service operation that returns
// all orders for the specified city.
var results =
    query.BeginExecute(OnAsyncQueryExecutionComplete, query);
```

```
' Define the service operation query parameter.
Dim city As String = "London"

' Create the DataServiceContext using the service URI.
Dim context As NorthwindEntities = New NorthwindEntities(svcUri2)

' Use the CreateQuery method to create a query that accessess
' the service operation passing a single parameter.
Dim query = context.CreateQuery(Of Order)("GetOrdersByCity") _
            .AddQueryOption("city", String.Format("'{0}'", city)) _
            .Expand("Order_Details")

' Define the delegate to callback into the process
Dim callback As AsyncCallback = AddressOf OnAsyncQueryExecutionComplete

' Execute the service operation that returns
' all orders for the specified city.
Dim results = _
    query.BeginExecute(callback, query)
```

```
private static void OnAsyncQueryExecutionComplete(IAsyncResult result)
{
    // Get the query back from the stored state.
    var query = result.AsyncState as DataServiceQuery<Order>;

    try
    {
        // Complete the exection and write out the results.
        foreach (Order o in query.EndExecute(result))
        {
            Console.WriteLine(string.Format("Order ID: {0}", o.OrderID));

            foreach (Order_Detail item in o.Order_Details)
            {
                Console.WriteLine(String.Format("\tItem: {0}, quantity: {1}",
                    item.ProductID, item.Quantity));
            }
        }
    }
    catch (DataServiceQueryException ex)
    {
        QueryOperationResponse response = ex.Response;

        Console.WriteLine(response.Error.Message);
    }
}
```

```
Private Shared Sub OnAsyncQueryExecutionComplete(ByVal result As IAsyncResult)
    ' Get the query back from the stored state.
    Dim query = TryCast(result.AsyncState, DataServiceQuery(Of Order))

    Try
        ' Complete the exection and write out the results.
        For Each o As Order In query.EndExecute(result)

            Console.WriteLine(String.Format("Order ID: {0}", o.OrderID))

            For Each item As Order_Detail In o.Order_Details
                Console.WriteLine(String.Format(vbTab & "Item: {0}, quantity: {1}", _
                    item.ProductID, item.Quantity))
            Next
        Next
    Catch ex As DataServiceQueryException
        Dim response As QueryOperationResponse = ex.Response

        Console.WriteLine(response.Error.Message)
    End Try
End Sub
```

See Also

WCF Data Services Client Library

Managing the Data Service Context (WCF Data Services)

5/2/2018 • 5 minutes to read • Edit Online

The DataServiceContext class encapsulates operations that are supported against a specified data service. Although OData services are stateless, the context is not. Therefore, you can use the DataServiceContext class to maintain state on the client between interactions with the data service in order to support features such as change management. This class also manages identities and tracks changes.

Merge Options and Identity Resolution

When a DataServiceQuery<TElement> is executed, the entities in the response feed are materialized into objects. For more information, see Object Materialization. The way in which entries in a response message are materialized into objects is performed based on identity resolution and depends on the merge option under which the query was executed. When multiple queries or load requests are executed in the scope of a single DataServiceContext, the WCF Data Services client only tracks a single instance of an object that has a specific key value. This key, which is used to perform identity resolution, uniquely identifies an entity.

By default, the client only materializes an entry in the response feed into an object for entities that are not already being tracked by the DataServiceContext. This means that changes to objects already in the cache are not overwritten. This behavior is controlled by specifying a MergeOption value for queries and load operations. This option is specified by setting the MergeOption property on the DataServiceContext. The default merge option value is AppendOnly. This only materializes objects for entities that are not already being tracked, which means that existing objects are not overwritten. Another way to prevent changes to objects on the client from being overwritten by updates from the data service is to specify PreserveChanges. When you specify OverwriteChanges, values of objects on the client are replaced by the latest values from the entries in the response feed, even if changes have already been made to these objects. When a NoTracking merge option is used, the DataServiceContext cannot send changes made on client objects to the data service. With this option, changes are always overwritten with values from the data service.

Managing Concurrency

OData supports optimistic concurrency that enables the data service to detect update conflicts. The data service provider can be configured in such a way that the data service checks for changes to entities by using a concurrency token. This token includes one or more properties of an entity type that are validated by the data service to determine whether a resource has changed. Concurrency tokens, which are included in the eTag header of requests to and responses from the data service, are managed for you by the WCF Data Services client. For more information, see Updating the Data Service.

The DataServiceContext tracks changes made to objects that have been reported manually by using AddObject, UpdateObject, and DeleteObject, or by a DataServiceCollection<T>. When the SaveChanges method is called, the client sends changes back to the data service. SaveChanges can fail when data changes in the client conflict with changes in the data service. When this occurs, you must query for the entity resource again to receive the update data. To overwrite changes in the data service, execute the query using the PreserveChanges merge option. When you call SaveChanges again, the changes preserved on the client are persisted to the data service, as long as other changes have not already been made to the resource in the data service.

Saving Changes

Changes are tracked in the DataServiceContext instance but not sent to the server immediately. After you are finished with the required changes for a specified activity, call SaveChanges to submit all the changes to the data service. A DataServiceResponse object is returned after the SaveChanges operation is complete. The DataServiceResponse object includes a sequence of OperationResponse objects that, in turn, contain a sequence of EntityDescriptor or LinkDescriptor instances that represent the changes persisted or attempted. When an entity is created or modified in the data service, the EntityDescriptor includes a reference to the updated entity, including any server-generated property values, such as the generated `ProductID` value in the previous example. The client library automatically updates the .NET Framework object to have these new values.

For successful insert and update operations, the state property of the EntityDescriptor or LinkDescriptor object associated with the operation is set to Unchanged and the new values are merged by using OverwriteChanges. When an insert, update, or delete operation fails in the data service, the entity state remains the same as it was before SaveChanges was called, and the Error property of the OperationResponse is set to an DataServiceRequestException that contains information about the error. For more information, see Updating the Data Service.

Setting the HTTP Method for Updates

By default, the .NET Framework client library sends updates to existing entities as MERGE requests. A MERGE request updates selected properties of the entity; however the client always includes all properties in the MERGE request, even properties that have not changed. The OData protocol also supports sending PUT requests to update entities. In a PUT request, an existing entity is essentially replaced with a new instance of the entity with property values from the client. To use PUT requests, set the ReplaceOnUpdate flag on the SaveChangesOptions enumeration when calling SaveChanges.

> **NOTE**
>
> A PUT request will behave differently than a MERGE request when the client does not know about all properties of the entity. This might occur when projecting an entity type into a new type on the client. It might also occur when new properties have been added to the entity in the service data model and the IgnoreMissingProperties property on the DataServiceContext is set to `true` to ignore such client mapping errors. In these cases, a PUT request will reset any properties that are unknown to the client to their default values.

POST Tunneling

By default, the client library sends create, read, update, and delete requests to an OData service by using the corresponding HTTP methods of POST, GET, PUT/MERGE/PATCH, and DELETE. This upholds the basic principles of Representational State Transfer (REST). However, not every Web server implementation supports the full set of HTTP methods. In some cases, the supported methods might be restricted to just GET and POST. This can happen when an intermediary, like a firewall, blocks requests with certain methods. Because the GET and POST methods are most often supported, OData prescribes a way to execute any unsupported HTTP methods by using a POST request. Known as *method tunneling* or *POST tunneling*, this enables a client to send a POST request with the actual method specified in the custom `X-HTTP-Method` header. To enable POST tunneling for requests, set the UsePostTunneling property on the DataServiceContext instance to `true`.

See Also

WCF Data Services Client Library
Updating the Data Service
Asynchronous Operations
Batching Operations

Working with Binary Data (WCF Data Services)

8/31/2018 • 5 minutes to read • Edit Online

The WCF Data Services client library enables you to retrieve and update binary data from an Open Data Protocol (OData) feed in one of the following ways:

- As a primitive type property of an entity. This is the recommended method for working with small binary data objects that can be easily loaded into memory. In this case, the binary property is an entity property exposed by the data model, and the data service serializes the binary data as base-64 binary encoded XML in the response message.
- As a separate binary resource stream. This is the recommended method for accessing and changing binary large object (BLOB) data that may represent a photo, video, or any other type of binary encoded data.

WCF Data Services implements the streaming of binary data by using HTTP as defined in the OData. In this mechanism, binary data is treated as a media resource that is separate from but related to an entity, which is called a media link entry. For more information, see Streaming Provider.

TIP

For a step-by-step example of how to create a Windows Presentation Foundation (WPF) client application that downloads binary image files from an OData service that stores photos, see the post Data Services Streaming Provider Series-Part 2: Accessing a Media Resource Stream from the Client. To download the sample code for the stream photo data service featured in the blog post, see the Streaming Photo Data Service Sample in MSDN Code Gallery.

Entity Metadata

An entity that has a related media resource stream is indicated in the data service metadata by the `HasStream` attribute applied to an entity type that is the media link entry. In the following example, the `PhotoInfo` entity is a media link entry that has a related media resource, indicated by the `HasStream` attribute.

```
<EntityType xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata"
            Name="PhotoInfo" m:HasStream="true">
  <Key>
    <PropertyRef Name="PhotoId" />
  </Key>
  <Property Name="PhotoId" Type="Int32" Nullable="false"
            annotation:StoreGeneratedPattern="Identity" />
  <Property Name="FileName" Type="String" Nullable="false" />
  <Property Name="FileSize" Type="Int32" Nullable="true" />
  <Property Name="DateTaken" Type="DateTime" Nullable="true" />
  <Property Name="TakenBy" Type="String" Nullable="true" />
  <Property Name="DateAdded" Type="DateTime" Nullable="false" />
  <Property Name="Exposure" Type="PhotoData.Exposure" Nullable="false" />
  <Property Name="Dimensions" Type="PhotoData.Dimensions" Nullable="false" />
  <Property Name="DateModified" Type="DateTime" Nullable="false" />
  <Property Name="Comments" Type="String" MaxLength="Max"
            FixedLength="false" Unicode="true" />
  <Property Name="ContentType" Type="String" MaxLength="50" FixedLength="false" Unicode="true" />
</EntityType>
```

The remaining examples in this topic show how to access and change the media resource stream. For a complete example of how to consume a media resource stream in a .NET Framework client application by using the WCF

Data Services client library, see the post Accessing a Media Resource Stream from the Client.

Accessing the Binary Resource Stream

The WCF Data Services client library provides methods for accessing binary resource streams from an OData-based data service. When downloading a media resource, you can either use the URI of the media resource or you can get a binary stream that contains the media resource data itself. You can also upload media resource data as a binary stream.

TIP

For a step-by-step example of how to create a Windows Presentation Foundation (WPF) client application that downloads binary image files from an OData service that stores photos, see the post Data Services Streaming Provider Series-Part 2: Accessing a Media Resource Stream from the Client. To download the sample code for the stream photo data service featured in the blog post, see the Streaming Photo Data Service Sample in MSDN Code Gallery.

Getting the URI of the Binary Stream

When retrieving certain types of media resources, such as images and other media files, it is often easier to use the URI of the media resource in your application than handling the binary data stream itself. To get the URI of a resource stream associated with a give media link entry, you must call the GetReadStreamUri method on the DataServiceContext instance that is tracking the entity. The following example shows how to call the GetReadStreamUri method to get the URI of a media resource stream that is used to create a new image on the client:

```
// Use the ReadStreamUri of the Media Resource for selected PhotoInfo object
// as the URI source of a new bitmap image.
photoImage.Source = new BitmapImage(context.GetReadStreamUri(currentPhoto));
```

```
' Use the ReadStreamUri of the Media Resource for selected PhotoInfo object
' as the URI source of a new bitmap image.
photoImage.Source = New BitmapImage(context.GetReadStreamUri(currentPhoto))
```

Downloading the Binary Resource Stream

When retrieving a binary resource stream, you must call the GetReadStream method on the DataServiceContext instance that is tracking the media link entry. This method sends a request to the data service that returns a DataServiceStreamResponse object, which has a reference to the stream that contains the resource. Use this method when your application requires the binary resource as a Stream. The following example shows how to call the GetReadStream method to retrieve a stream that is used to create a new image on the client:

```
// Get the read stream for the Media Resource of the currently selected
// entity (Media Link Entry).
using (DataServiceStreamResponse response =
    context.GetReadStream(currentEmployee, "image/bmp"))
{
    // Use the returned binary stream to create a bitmap image that is
    // the source of the image control.
    employeeImage.Source = CreateBitmapFromStream(response.Stream);
}
```

```
' Get the read stream for the Media Resource of the currently selected
' entity (Media Link Entry).
Using response As DataServiceStreamResponse = _
        context.GetReadStream(currentEmployee, "image/bmp")

    ' Use the returned binary stream to create a bitmap image that is
    ' the source of the image control.
    employeeImage.Source = CreateBitmapFromStream(response.Stream)
End Using
```

NOTE

The Content-Length header in the response message that contains the binary steam is not set by the data service. This value may not reflect the actual length of the binary data stream.

Uploading a Media Resource as a Stream

To insert or update a media resource, call the SetSaveStream method on the DataServiceContext instance that is tracking the entity. This method sends a request to the data service that contains the media resource read from the supplied stream. The following example shows how to call the SetSaveStream method to send an image to the data service:

```
// Set the file stream as the source of binary stream
// to send to the data service. The Slug header is the file name and
// the content type is determined from the file extension.
// A value of 'true' means that the stream is closed by the client when
// the upload is complete.
context.SetSaveStream(photoEntity, imageStream, true,
    photoEntity.ContentType, photoEntity.FileName);
```

```
' Set the file stream as the source of binary stream
' to send to the data service. The Slug header is the file name and
' the content type is determined from the file extension.
' A value of 'true' means that the stream is closed by the client when
' the upload is complete.
context.SetSaveStream(photoEntity, imageStream, True, _
    photoEntity.ContentType, photoEntity.FileName)
```

In this example, the SetSaveStream method is called by supplying a value of `true` for the `closeStream` parameter. This guarantees that the DataServiceContext closes the stream after the binary data is uploaded to the data service.

NOTE

When you call SetSaveStream, the stream is not sent to the data service until SaveChanges is called.

See Also

WCF Data Services Client Library
Binding Data to Controls