



CODE  
PROJECT®  
For those who code

[articles](#)[Q&A](#)[forums](#)[lounge](#)[Follow](#)

# Create a WCF DataService in Visual Studio 2017



NobsterTheLobster, 16 May 2018



4.87 (7 votes)

Rate: ★★★★★

Publish a WCF DataService with entity model, updatable views, filtering and authentication



[Download WcfDataServiceItemTemplate.zip - 3.5 KB](#)



[Download AuthenticationModule.zip - 5.3 KB](#)

- [Introduction](#)
- [Background](#)
- [Creating the Service](#)
- [Making views updatable](#)
- [Basic Authentication](#)
- [Azure Active Directory OAuth 2.0 Authentication](#)
- [AD Roles](#)
- [AD Groups](#)
- [Resolving AD Group Object IDs](#)
- [Stored procedure entity with multiple result sets](#)
- [Filtering Data](#)
- [Alternative Hosting](#)

## Introduction

There are plenty of articles available on creating web services, however it was my experience that some of the information was slightly out of date (using older tools/frameworks) and it was also my experience that I had to piece information from many sources together to get around bugs or implement all features. This article is an attempt to make a definitive guide for making a WCF Data Service in WCF using entity framework 6 and WCF Data Services 5.6. In my next article, I will show how to consume data from the service in a universal Windows platform app.

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.

## Background

Some great articles on odata and WCF are:

- <http://www.codeproject.com/Articles/393623/OData-Services>
- <http://www.codeproject.com/Articles/572417/AplusBeginner-splusTutorialplusforplusCreatingpl>

Some principles on OData:

- <http://www.odata.org/>

Some good information on authentication:

- [https://msdn.microsoft.com/en-us/library/dd728284\(v=vs.110\).aspx](https://msdn.microsoft.com/en-us/library/dd728284(v=vs.110).aspx)
- <https://blogs.msdn.microsoft.com/odatateam/2010/07/21/odata-and-authentication-part-6-custom-basic-authentication/>
- <https://azure.microsoft.com/en-us/resources/samples/active-directory-dotnet-webapi-manual-jwt-validation/>

Information on filtering data (query interception):

- [https://msdn.microsoft.com/en-us/library/dd744837\(v=vs.110\).aspx](https://msdn.microsoft.com/en-us/library/dd744837(v=vs.110).aspx)

Some workarounds for errors:

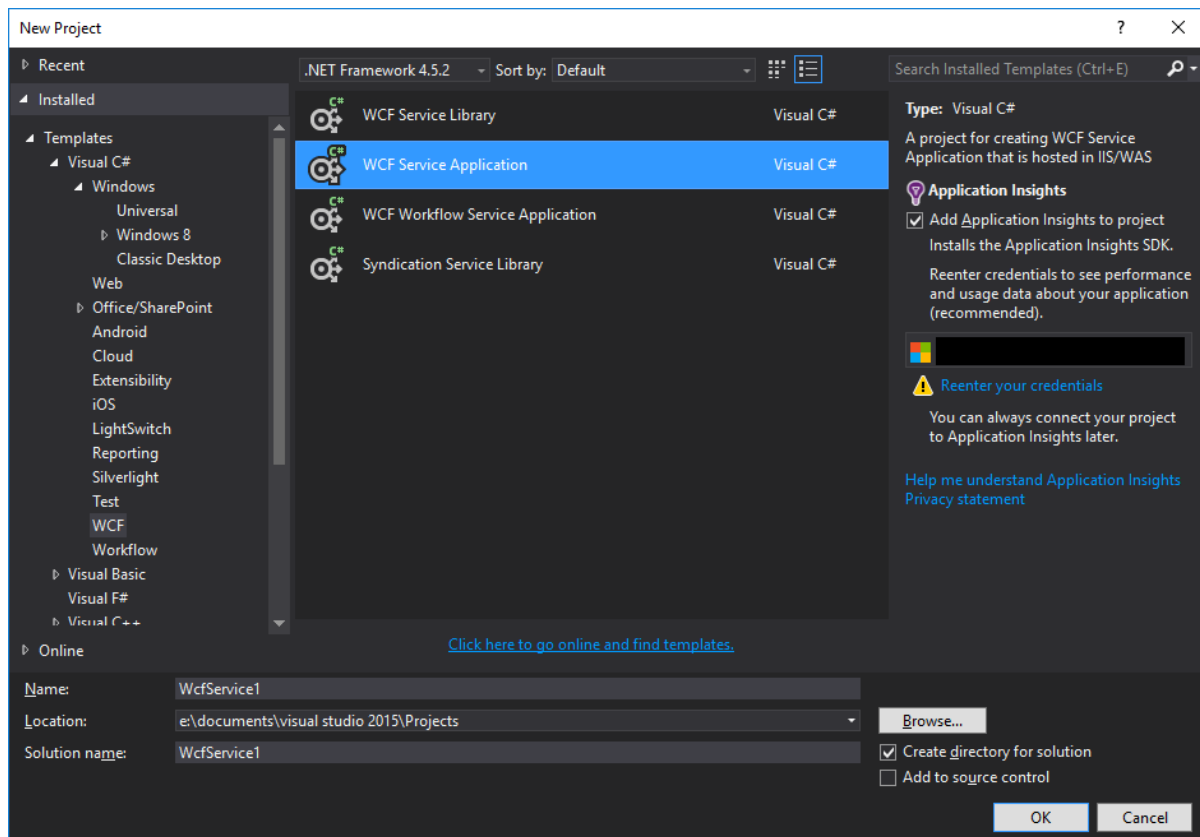
- <https://social.msdn.microsoft.com/Forums/en-US/1b023c1b-7cd3-4e4b-9a93-328c3194e907/vs2013-ef6-entityframeworkdataservice-could-not-be-found?forum=adodotnetdataservices>

Information on stored procedure entities with multiple result sets:

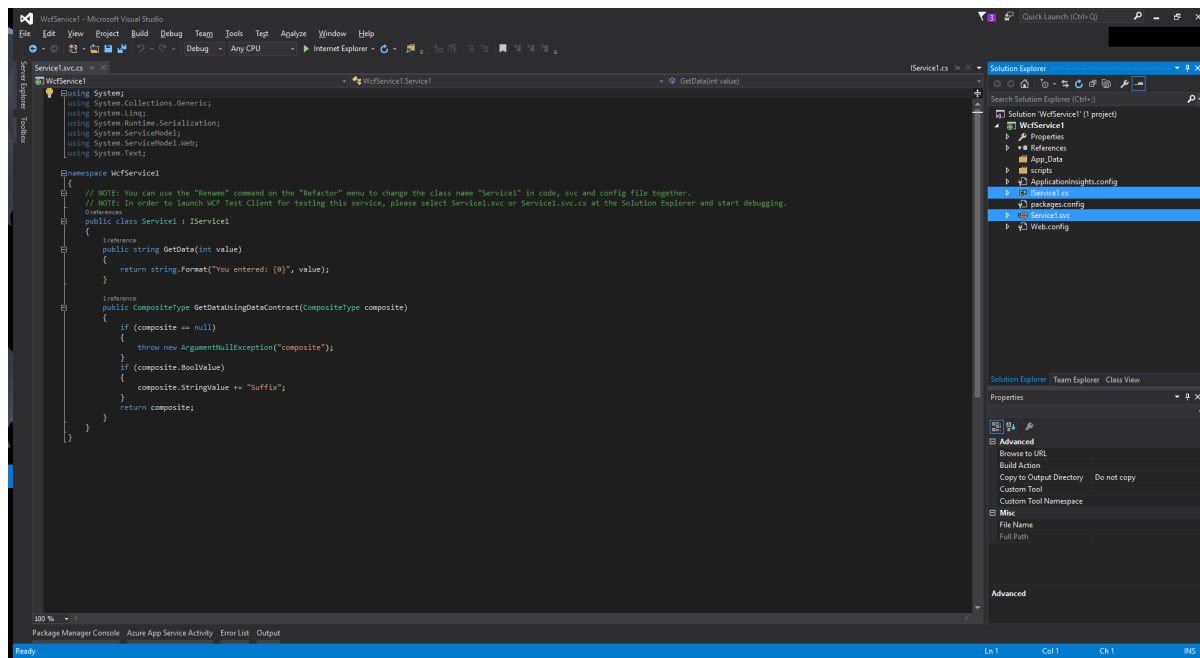
- <https://msdn.microsoft.com/en-us/data/jj691402.aspx>

## Creating the Service

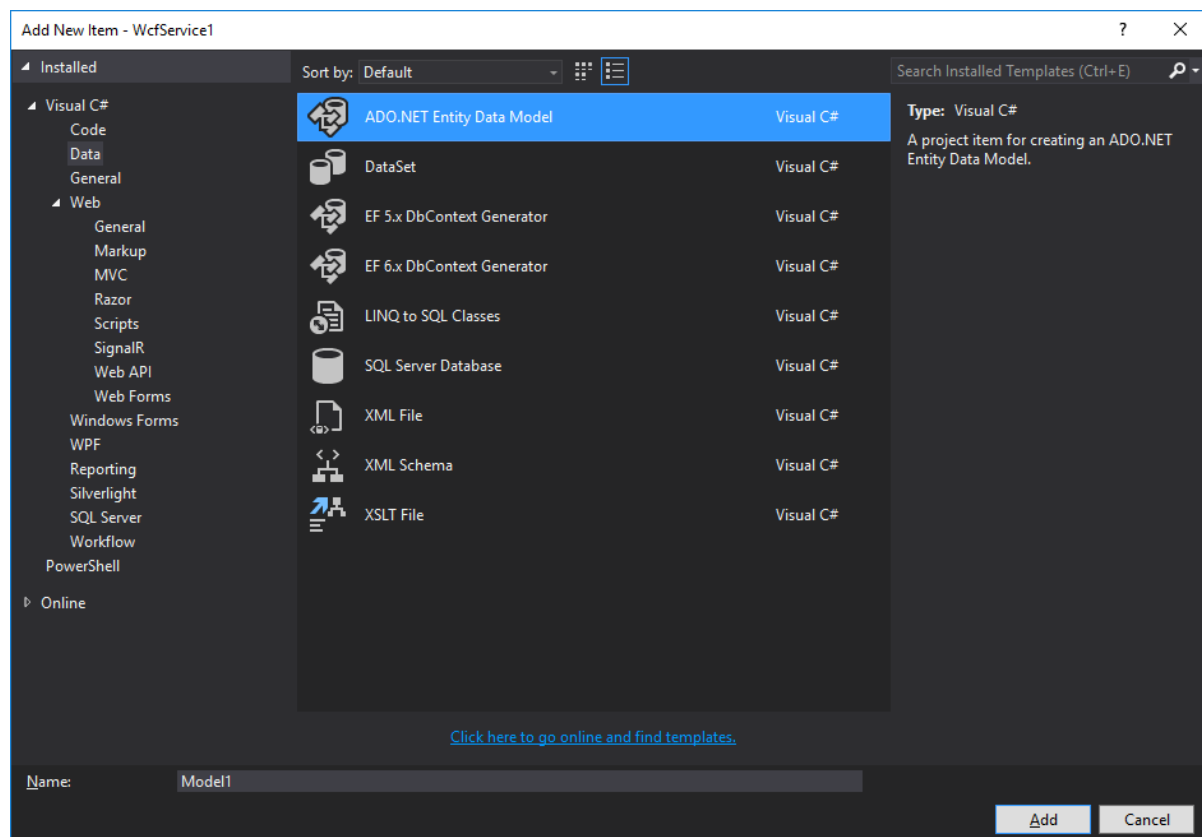
1. Go to **File -> New Project**.
2. In the list of Installed Templates, select the **Visual C# | WCF** tree node and then select the **WCF Service Application**.



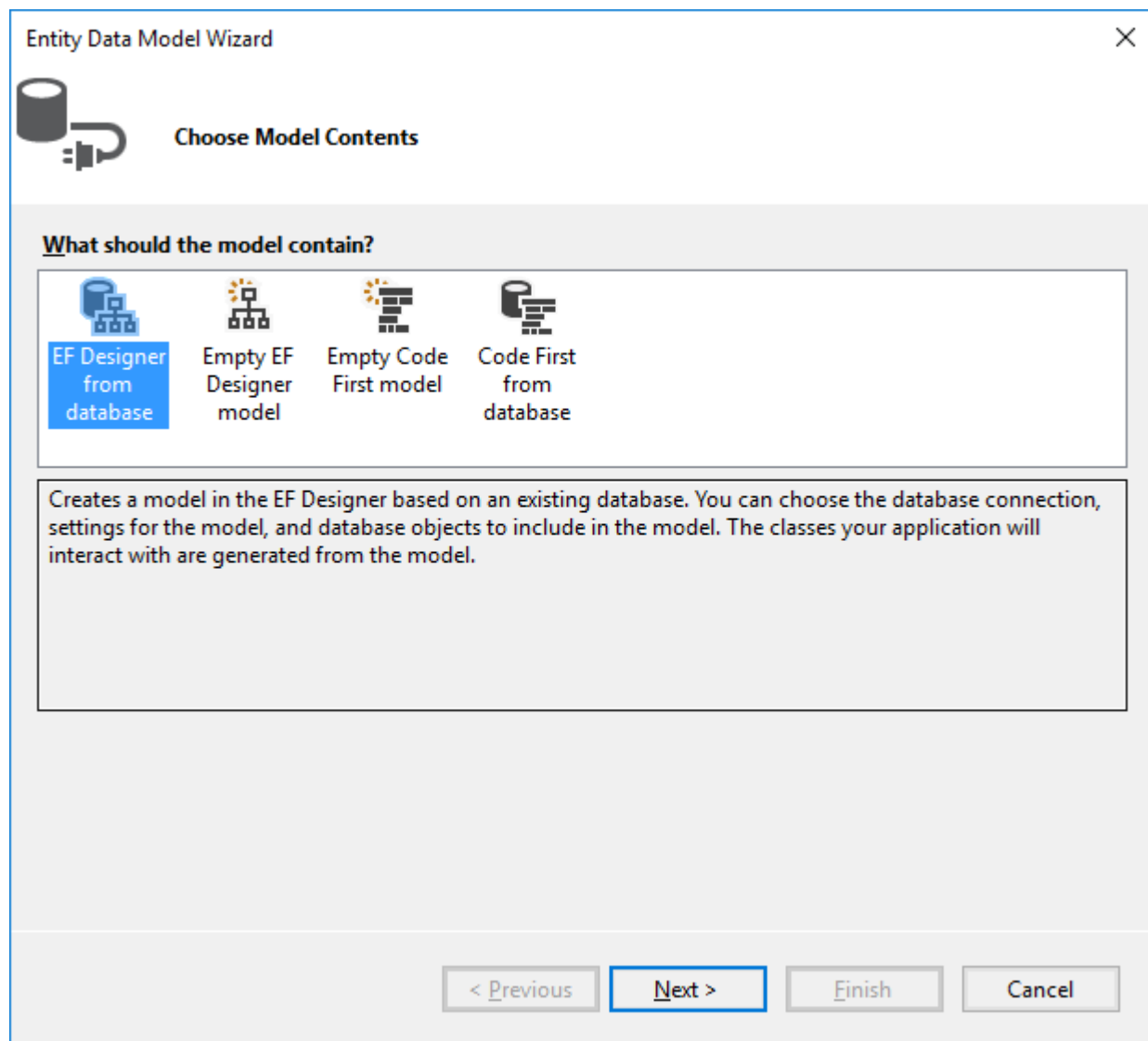
3. Delete *IService1.cs* and *Service1.svc* from the resulting project.



4. Add a new item to the project. In the list of Installed Templates, select the **Visual C# | Data** tree node and then select the **ADO.NET Entity Data Model**.



5. For the purposes of this article, I am selecting to build my entity module from an existing database.



6. Define your connection. Typically a cloud server.

Connection Properties

Enter information to connect to the selected data source or click "Change" to choose a different data source and/or provider.

Data source:  
Microsoft SQL Server (SqlClient) Change...

Server name:  
[Redacted] Refresh

Log on to the server

☐ Use Windows Authentication

☒ Use SQL Server Authentication

User name: [Redacted]

Password: [Redacted]

☐ Save my password

Connect to a database

☒ Select or enter a database name:  
[Redacted]

☐ Attach a database file:  
[Redacted] Browse...

Logical name:  
[Redacted]


Advanced...

Test Connection OK Cancel

7. Save your connection to the *web.config* file.

Entity Data Model Wizard

×

 Choose Your Data Connection

**Which data connection should your application use to connect to the database?**

New Connection...

This connection string appears to contain sensitive data (for example, a password) that is required to connect to the database. Storing sensitive data in the connection string can be a security risk. Do you want to include this sensitive data in the connection string?

☐ No, exclude sensitive data from the connection string. I will set it in my application code.

☒ Yes, include the sensitive data in the connection string.

Connection string:

^

v

☒ Save connection settings in Web.Config as:

< Previous


Next >

Finish

Cancel

8. Specify entity version 6.0.

Entity Data Model Wizard ✕

 Choose Your Version

**Which version of Entity Framework do you want to use?**

☒ Entity Framework 6.x

☐ Entity Framework 5.0

**i** It is also possible to install and use other versions of Entity Framework.  
[Learn more about this](#)

< Previous **Next >** Finish Cancel

9. Select the tables/view/procs to include in your entity model.



Entity Data Model Wizard

Choose Your Database Objects and Settings

**Which database objects do you want to include in your model?**

- ☒ Tables
  - > ☐ dbo
- ☒ Views
  - ☐ dbo
    - ☐ Behaviours
    - ☐ Billpay
    - ☐ CigVending
    - ☐ DiscountReasons
    - ☐ DriveOffs
    - ☐ Hardware
    - ☐ Limits
    - ☐ Loyalty

☒ Pluralize or singularize generated object names

☒ Include foreign key columns in the model

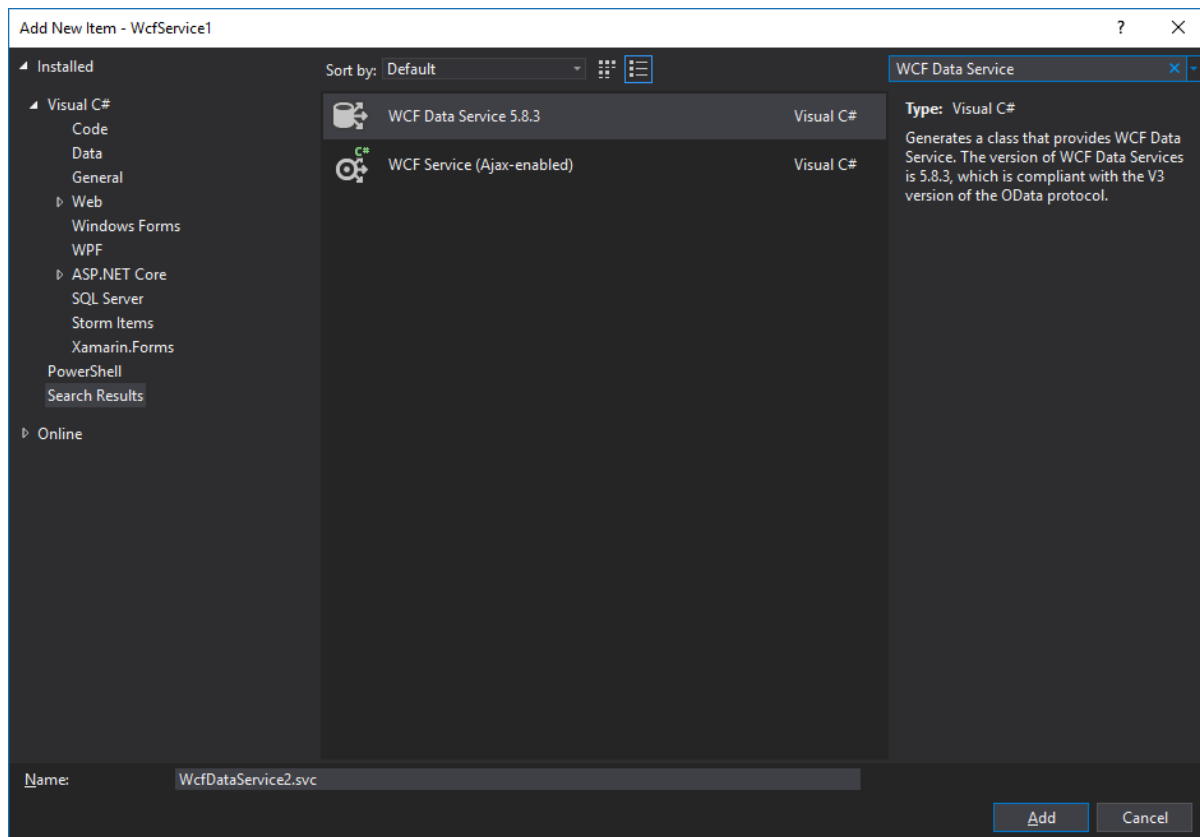
☐ Import selected stored procedures and functions into the entity model

**Model Namespace:**

FutaTillHOModel

< Previous   Next >   **Finish**   Cancel

10. Unzip the attached data service template **WcfDataServiceItemTemplate.zip** to your Visual Studio C# Item-Templates folder which is typically "{Documents}\Visual Studio 2017\Templates\ItemTemplates\Visual C#" (where {Documents} is your user profile documents folder.) Note the unzipped hierarchy should be a single level with the template files with in e.g. ...\\ItemTemplates\\Visual C#\\**WcfDataServiceItemTemplate**\\WebDataService.vstemplate. Now in visual studio, right click your project and choose to add a new item. Search the installed templates for WCF Data Service and add the new WCF Data Service 5.8.3 item to your project. If the template does not appear then restart visual studio.



11. Edit the `WcfDataService1.svc` file Change the `<TODOReplaceWithYourEntitySetName>` to be the name of your entity model in my case `futaTillH0Entities`

Optionally:

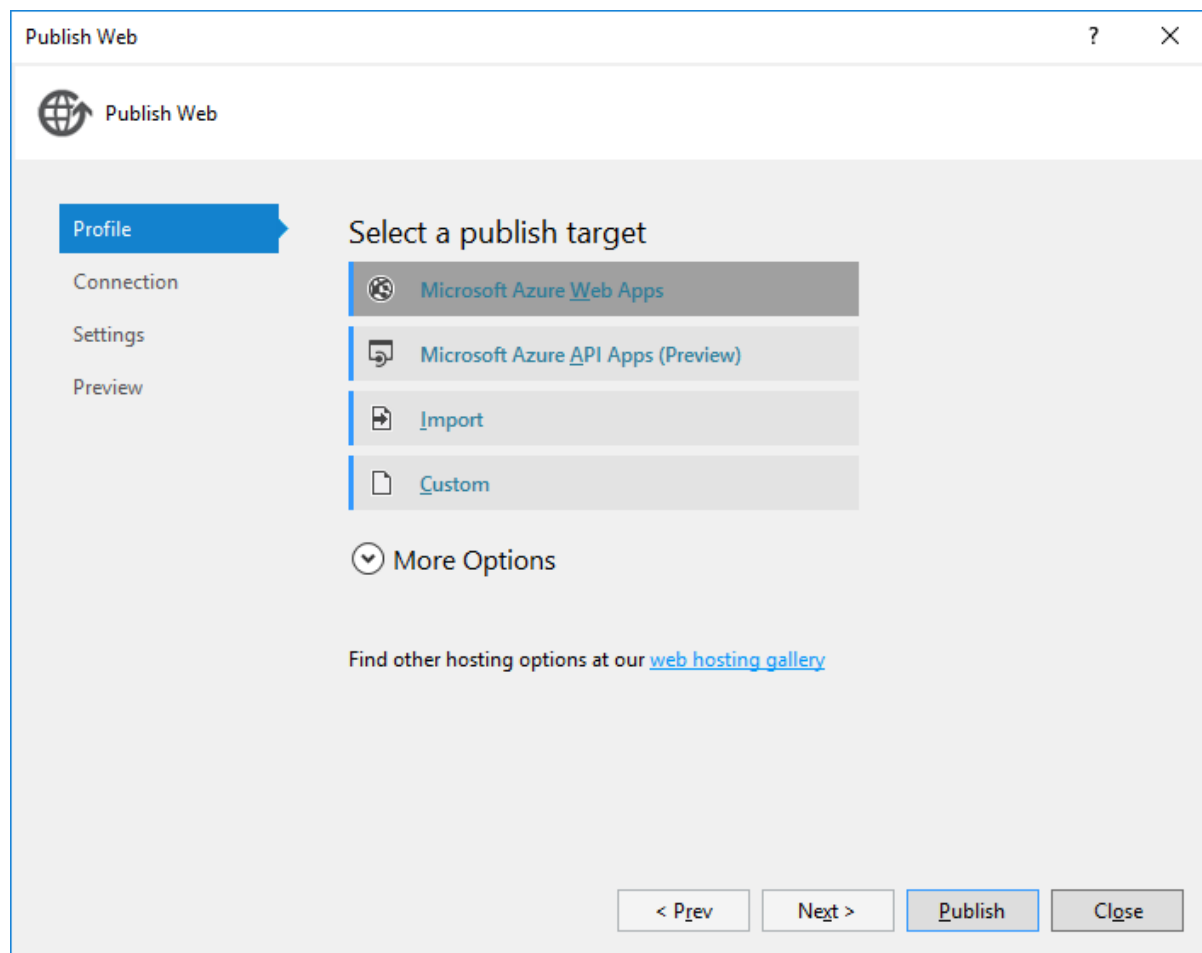
- Set the access rules for your entity names. Use the star symbol to mean all entities.
- Set the `UseVerboseErrors` property in order to see proper feedback of errors.

[Hide](#) [Copy Code](#)

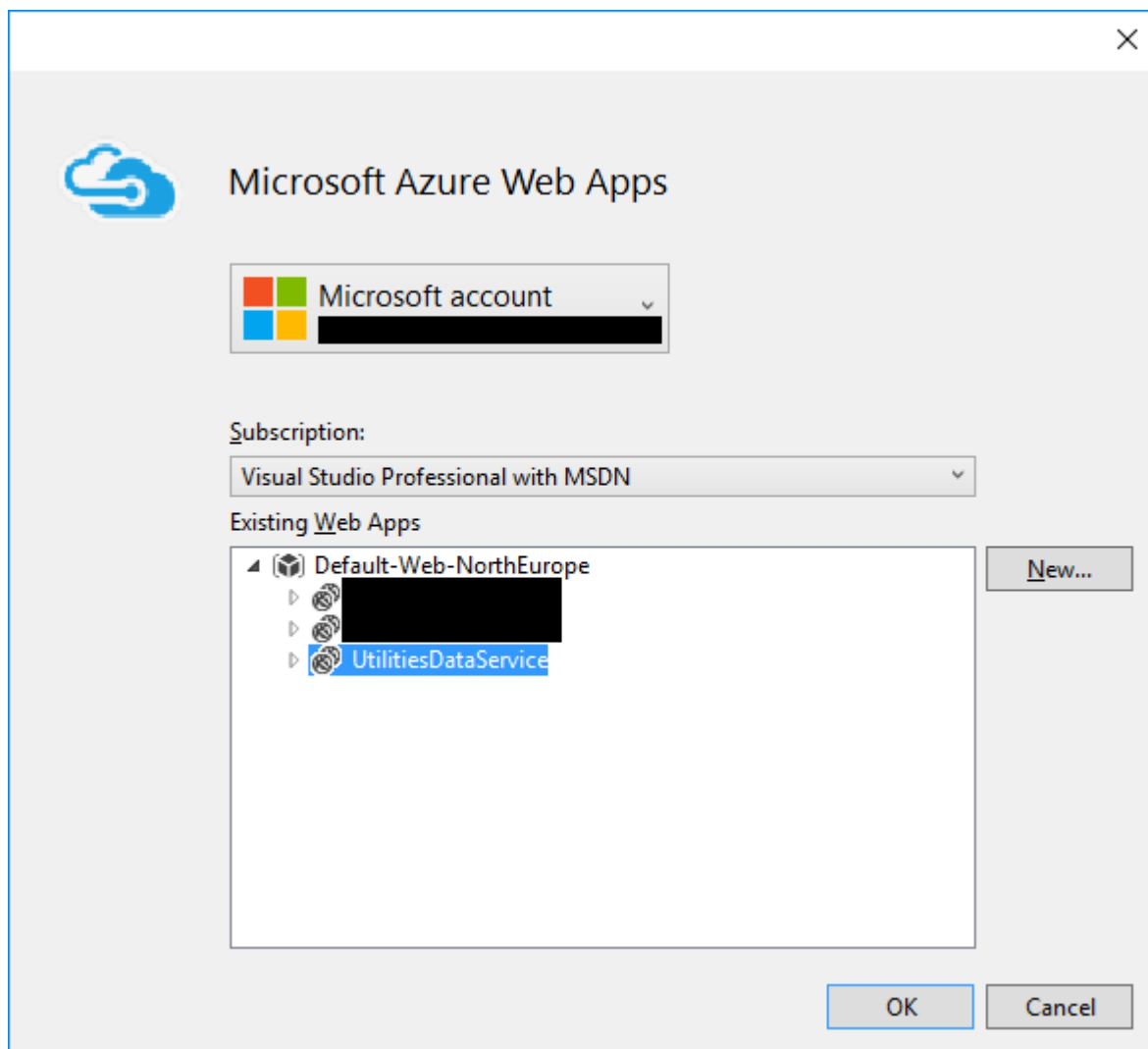
```
using System.Data.Services.Providers;

namespace WcfService1
{
    public class WcfDataService1 : EntityFrameworkDataService<FutaTillH0Entities>
    {
        // This method is called only once to initialize service-wide policies.
        public static void InitializeService(DataServiceConfiguration config)
        {
            // TODO: set rules to indicate which entity sets and service operations are visible,
            // updatable, etc.
            // Examples:
            config.SetEntitySetAccessRule("*", EntitySetRights.AllRead);
            // config.SetServiceOperationAccessRule("MyServiceOperation",
            // ServiceOperationRights.All);
            config.DataServiceBehavior.MaxProtocolVersion = DataServiceProtocolVersion.V3;
            config.UseVerboseErrors = true;
        }
    }
}
```

12. Create a new web app either from Visual Studio server explorer or your Azure portal. Then right click your project and click publish. In the publish screen, select Azure Web Apps.



13. Select your web app that you want to publish too. In my case, the name is **UtilitiesDataService**.



14. Click Next if settings are correct.

Publish Web

Profile

Connection

Settings

Preview

**UtilitiesDataService \***

Publish method: Web Deploy

Server: [Redacted]

Site name: UtilitiesDataService

User name: \$UtilitiesDataService

Password: [Redacted]

☒ Save password

Destination URL: [Redacted]

Validate Connection

< Prev Next > Publish Close

15. Click Next if settings are correct.

Publish Web

Profile

Connection

**Settings**

Preview

**UtilitiesDataService \***

Configuration: Release

File Publish Options

Databases

FutaTillHOEntities

Use this connection string at runtime (update destination web.config)

Update database [Configure database updates](#)

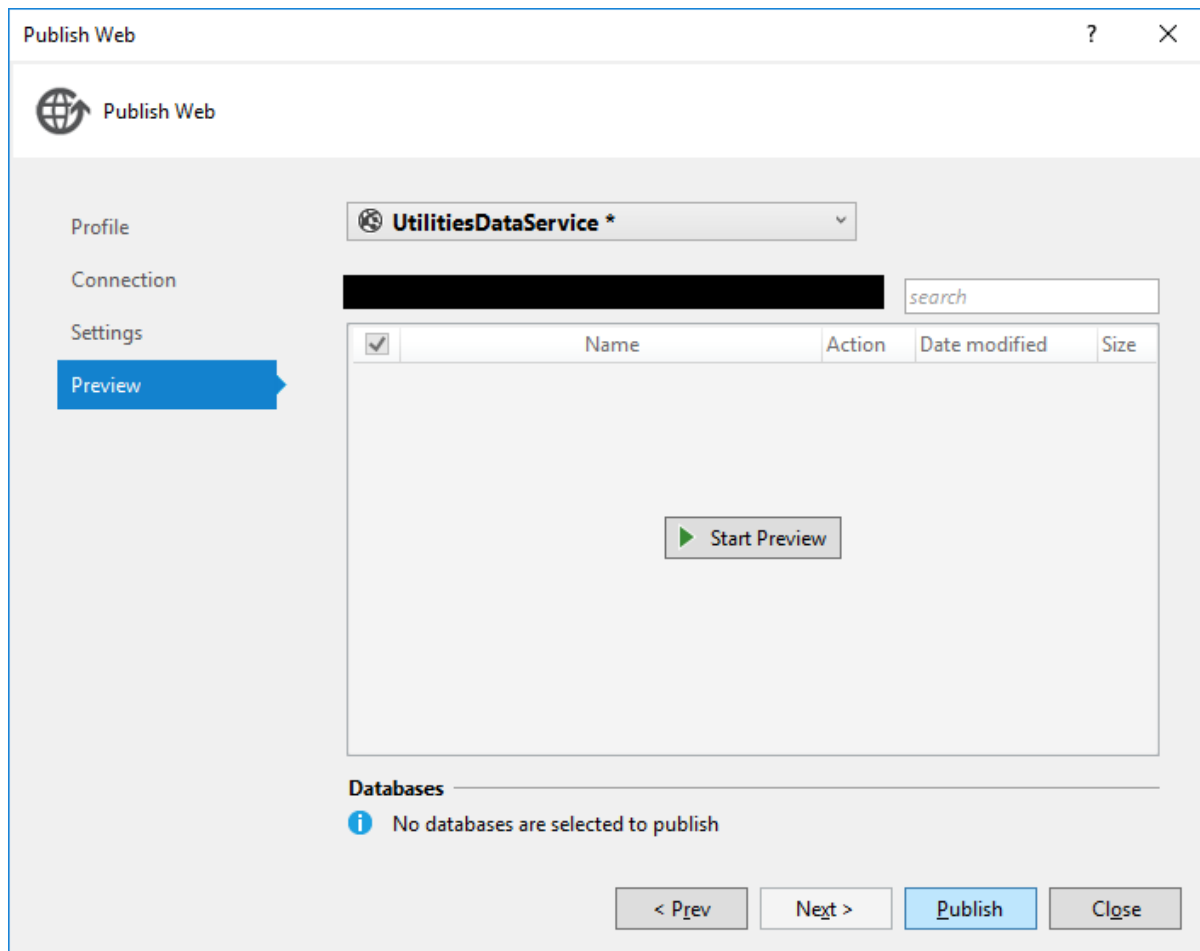
< Prev

Next >

Publish

Close

16. Click Publish.



17. You should now be able to test your web-service using the URL of your web-app (visible in your azure portal) + the service name + the entity name

Example:

- <http://mydataservicename.azurewebsites.net/WcfDataService1.svc/EntityName>

To retrieve a specific record by including the primary key in the URL:

- [http://mydataservicename.azurewebsites.net/WcfDataService1.svc/EntityName\(RecordID\)](http://mydataservicename.azurewebsites.net/WcfDataService1.svc/EntityName(RecordID))

If the key is composite, then use the following notation:

- [http://mydataservicename.azurewebsites.net/WcfDataService1.svc/EntityName\(PrimaryKey1=2,PrimaryKey2='ABC'\)](http://mydataservicename.azurewebsites.net/WcfDataService1.svc/EntityName(PrimaryKey1=2,PrimaryKey2='ABC'))

By default, the information will be serialized as an atom feed but you can include the format specifier to get data in json:

- [http://mydataservicename.azurewebsites.net/WcfDataService1.svc/EntityName?\\$format=json](http://mydataservicename.azurewebsites.net/WcfDataService1.svc/EntityName?$format=json)

You can also include functions such as top/skip/expand and combine them using & character:

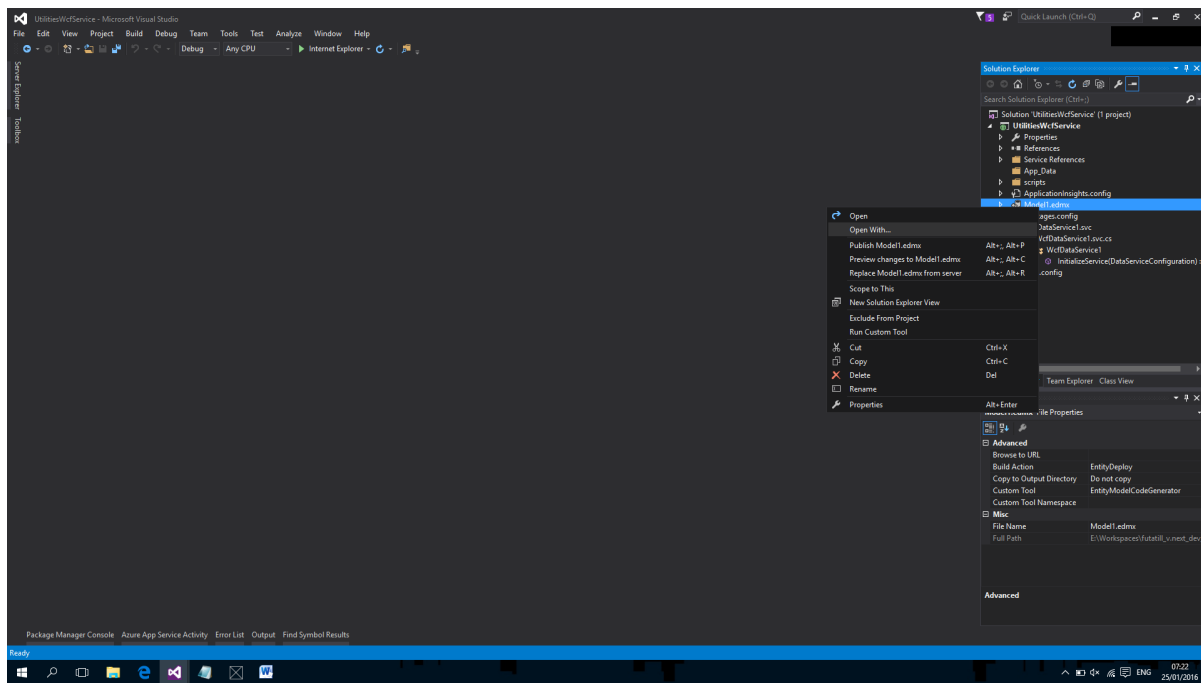
- [http://mydataservicename.azurewebsites.net/WcfDataService1.svc/EntityName?\\$format=json&\\$top=20](http://mydataservicename.azurewebsites.net/WcfDataService1.svc/EntityName?$format=json&$top=20)

Look at <http://www.odata.org/> for more examples.

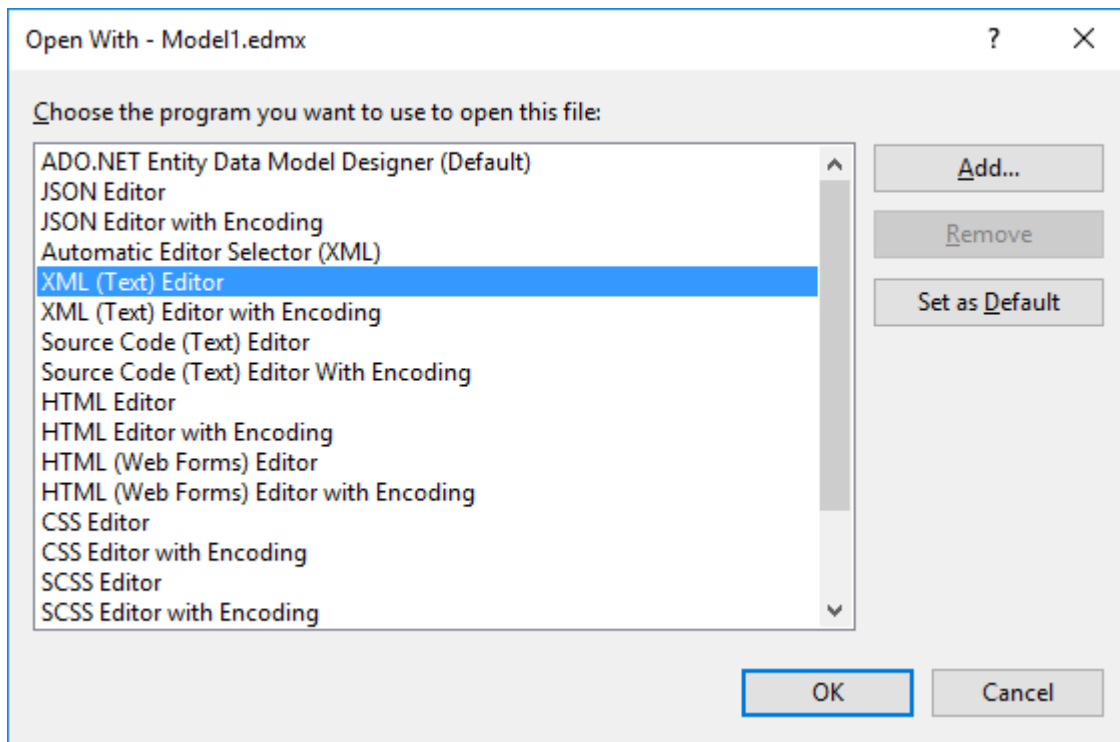
## Making Views Updatable

By default, the **entityset** is setup in such a way to prevent updates on views. If your view is updatable, then perform the following steps:

1. Right click the **model** element and select open with.

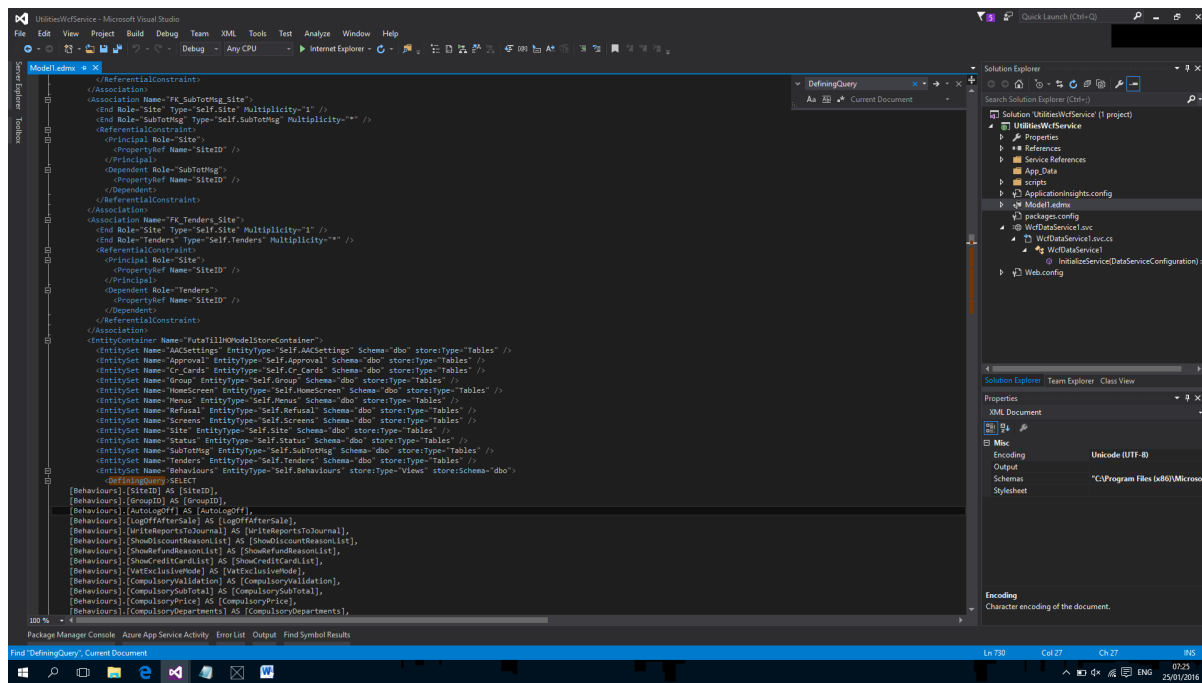


2. Select the XML editor and click OK.

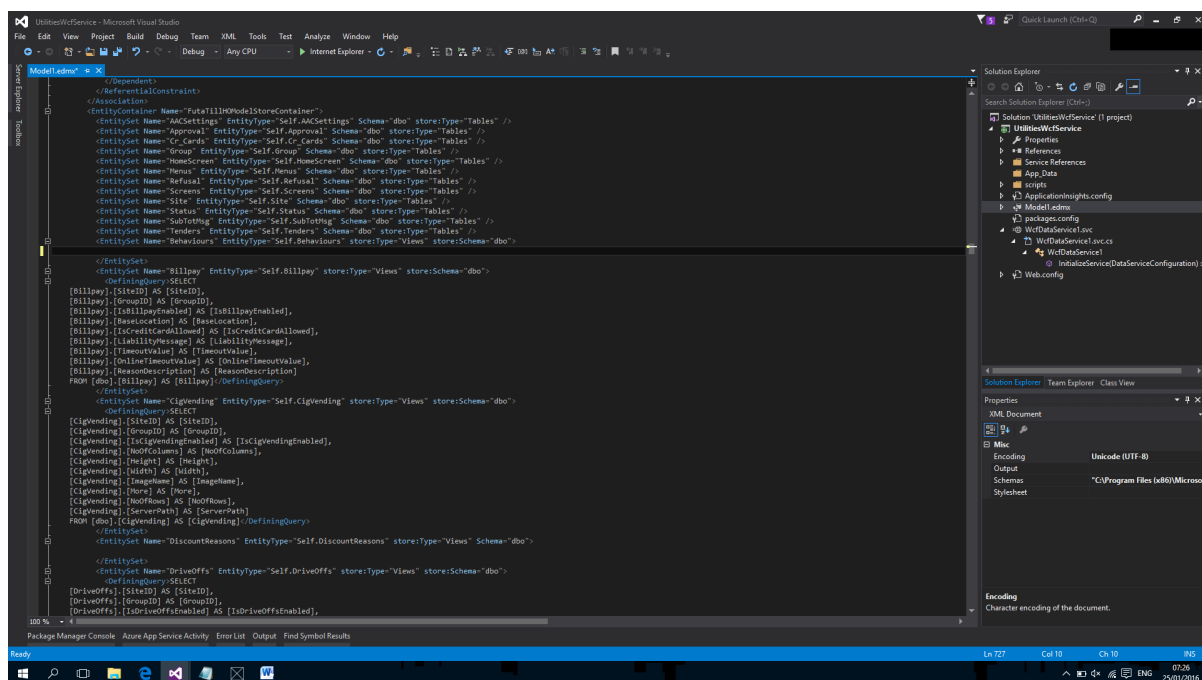


3. Do a find for the text **<DefiningQuery>**. You will notice that this element and inner query is present for your views, but not for your tables.





- Remove the **DefiningQuery** element. In this screen, I've shown the element removed from the **Behaviours** view, however you will need to remove it from all of the views.



- Strangely, you also need to change the text **store:Schema="dbo"** to be just **Schema="dbo"**. Basically, if you look at how tables are defined, you can see the difference.

## [Table Definition]

```
<EntitySet Name="Tenders" EntityType="Self.Tenders"
Schema="dbo" store:Type="Tables" />
```

## [Original View Definition]

Hide Copy Code

Hide Copy Code

```
<EntitySet Name="Behaviours" EntityType="Self.Behaviours"
store:Type="Views" store:Schema="dbo"/>
```

### [Corrected View Definition]

Hide Copy Code

```
<EntitySet Name="Behaviours"
EntityType="Self.Behaviours" store:Type="Views" Schema="dbo"/>
```

6. After editing and saving the XML, it is sometimes necessary to go back into the entity model designer and click the save button in the tool bar.

## Basic Authentication

In order to setup basic authentication, you need two additional classes in your service. I have attached the entire source for these classes to the article.

The **BasicAuthenticationModule** class sets up the event handling for an authentication request and forwards to the **BasicAuthenticationProvider.Authenticate** method.

Hide Copy Code

```
public class BasicAuthenticationModule : IHttpModule
{
    public void Init(HttpApplication context)
    {
        //Attach handling for authentication requests.
        context.AuthenticateRequest
            += new EventHandler(context_AuthenticateRequest);
    }
    void context_AuthenticateRequest(object sender, EventArgs e)
    {
        //Unbox the application.
        HttpApplication application = (HttpApplication)sender;

        //Send to provider for authentication.
        if (!BasicAuthenticationProvider.Authenticate(application.Context))
        {
            application.Context.Response.Status = "401 Unauthorized";
            application.Context.Response.StatusCode = 401;
            application.Context.Response.AddHeader("WWW-Authenticate", "Basic");
            application.CompleteRequest();
        }
    }
    public void Dispose() { }
}
```

Here is the **BasicAuthenticationProvider** class. Please note that Basic Authentication in itself is not secure since the username and password are sent unencrypted. Therefore, basic authentication should only be allowed in an SSL environment. The code that enforces this condition has been commented out, in the code sample below, to allow testing.

Hide Shrink ▲ Copy Code

```
public class BasicAuthenticationProvider
{
    /// <summary>
    /// Authenticate and the set the current http context user.
    /// </summary>
    /// <param name="context"></param>
    /// <returns></returns>
    public static bool Authenticate(HttpContext context)
    {

```

```

//This needs to be uncommented for live site.
//This will reject the login when not using SSL.
//if (!HttpContext.Current.Request.IsSecureConnection)
//    return false;
//I only want to execute code for authorization requests.
if (!HttpContext.Current.Request.Headers.AllKeys.Contains("Authorization"))
    return false;

string authHeader = HttpContext.Current.Request.Headers["Authorization"];

IPrincipal principal;
if (TryGetPrincipal(authHeader, out principal))
{
    HttpContext.Current.User = principal;
    return true;
}
return false;
}

/// <summary>
/// 
/// </summary>
/// <param name="authHeader"></param>
/// <param name="principal"></param>
/// <returns></returns>
private static bool TryGetPrincipal(string authHeader, out IPrincipal principal)
{
    var creds = ParseAuthHeader(authHeader);
    if (creds != null && TryGetPrincipal(creds, out principal))
        return true;

    principal = null;
    return false;
}

```

In basic authentication, the user name and password are stored in the header. This method extracts the username and password credentials into an array.

Hide Shrink ▲ Copy Code

```

/// <summary>
/// In basic authentication the user name and password are in the auth header in base64 encoding.
/// </summary>
/// <param name="authHeader"></param>
/// <returns>The array of credentials i.e. the username and password.</returns>
private static string[] ParseAuthHeader(string authHeader)
{
    // Check this is a Basic Auth header
    if (
        authHeader == null ||
        authHeader.Length == 0 ||
        !authHeader.StartsWith("Basic")
    ) return null;

    // Pull out the Credentials with are separated by ':' and Base64 encoded
    string base64Credentials = authHeader.Substring(6);
    string[] credentials = Encoding.ASCII.GetString(
        Convert.FromBase64String(base64Credentials)
    ).Split(new char[] { ':' });

    if (credentials.Length != 2 ||
        string.IsNullOrEmpty(credentials[0]) ||
        string.IsNullOrEmpty(credentials[1])
    ) return null;

    // Okay this is the credentials
    return credentials;
}

```

It is in this overload of **TryGetPrincipal** that you must hook up your username/password data store. In the code sample below, I've used another entity model with a single stored procedure called **aspnet\_GetUserCredentials** to return multiple result sets containing the login information for the supplied username. The first result set returns the username and password details, the second contains the roles for the user and the third contains the role permissions. The supplied password (from the authentication header) is hashed and compared with the stored hash. If there is a match, then the principle object is initialized to a new container for the user credentials, roles and associated permissions.

Hide Shrink ▲ Copy Code

```
/// <summary>
///
/// </summary>
/// <param name="creds"></param>
/// <param name="principal"></param>
/// <returns></returns>
private static bool TryGetPrincipal(string[] creds, out IPrincipal principal)
{
    bool located = false;
    principal = null;

    //The user match.
    var user = new User_SprocResult();

    //The list of roles.
    var roles = new List<Role_SprocResult>();

    //The list of permissions.
    var permissions = new List<Permission_SprocResult>();

    //Use the entity context.
    using (var dbContext = new AuthenticationEntities())
    {
        //Get first enumerate result set.
        var result = dbContext.aspnet_GetUserCredentials("Utilities", creds[0]);
        user = result.FirstOrDefault();

        //Get second result set
        var result2 = result.GetNextResult<Role_SprocResult>();
        roles.AddRange(result2);

        //Get third result set
        permissions.AddRange(result2.GetNextResult<Permission_SprocResult>());
    }

    //If there are any user matches.
    if (user != null)
    {
        //Get the hash of this users password using the salt provided.
        byte[] bytes = Encoding.Unicode.GetBytes(creds[1]);
        byte[] src = Convert.FromBase64String(user.PasswordSalt);
        byte[] dst = new byte[src.Length + bytes.Length];
        Buffer.BlockCopy(src, 0, dst, 0, src.Length);
        Buffer.BlockCopy(bytes, 0, dst, src.Length, bytes.Length);
        HashAlgorithm algorithm = HashAlgorithm.Create("SHA1");
        byte[] inArray = algorithm.ComputeHash(dst);

        //If the resulting hash is equal to the stored hash for this user.
        if (string.Compare(Convert.ToBase64String(inArray), user.Password) == 0)
        {
            //Tag as located.
            located = true;

            //Set new principal.
            principal = new CustomPrincipal(user.UserName,
                roles.Select(r=>r.RoleName).ToArray(),
                permissions.Select(r=>r.PermissionId).ToArray());
        }
    }
}
```

```
//Return result.
return located;
}
```

Here is the specialized container for the user principal.

Hide Shrink ▲ Copy Code

```
public class CustomPrincipal : IPrincipal
{
    string[] _roles;
    string[] _permissions;
    IIdentity _identity;

    public CustomPrincipal(string name, string[] roles, string[] permissions)
    {
        this._roles = roles;
        this._permissions = permissions;
        this._identity = new GenericIdentity(name);
    }

    public IIdentity Identity
    {
        get { return _identity; }
    }

    public bool IsInRole(string role)
    {
        return _roles.Contains(role);
    }

    public bool HasPermission(string permission)
    {
        return _permissions.Contains(permission);
    }
}
```

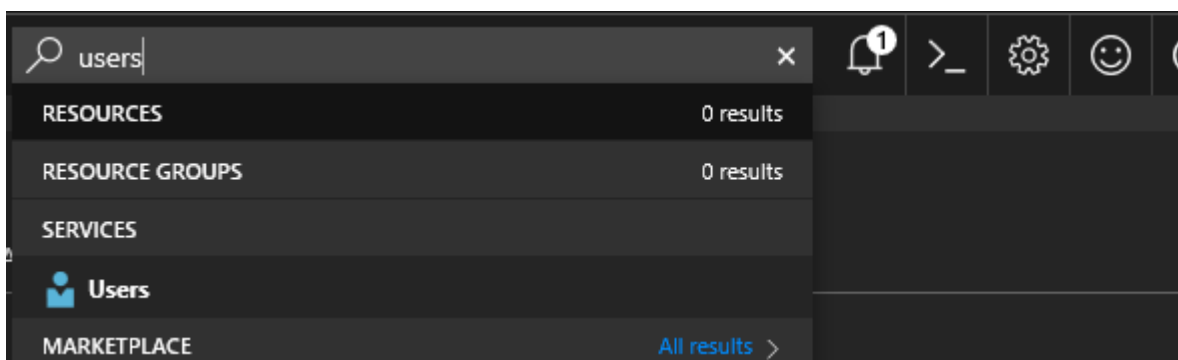
In order for your web service to actually implement the authentication module, you must add it to the modules node of your *web.config*.

Hide Copy Code

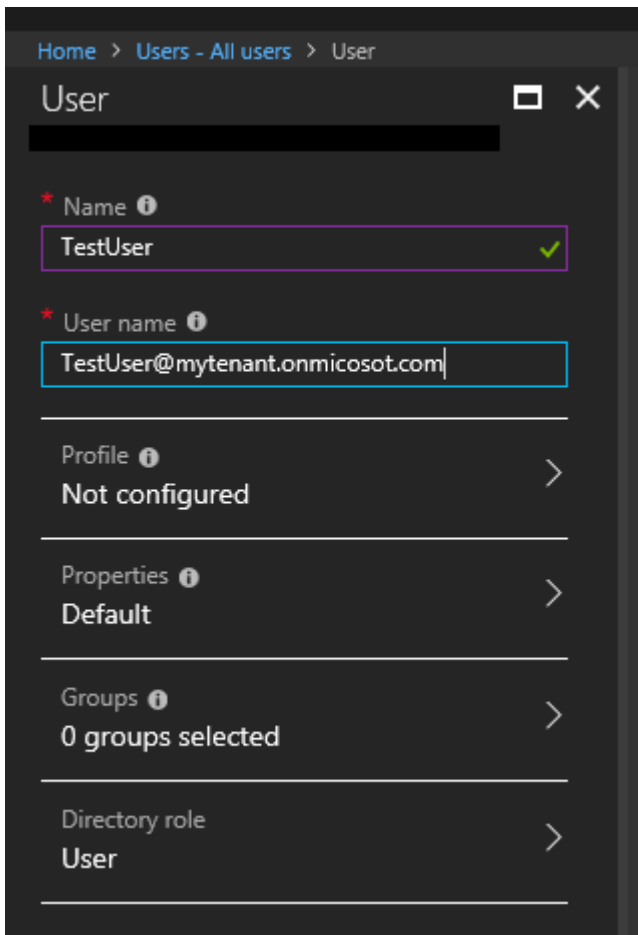
```
<modules runAllManagedModulesForAllRequests="true">
  <add name="BasicAuthentication" type="UtilitiesWcfService.BasicAuthenticationModule" />
  <remove name="ApplicationInsightsWebTracking" />
  <add name="ApplicationInsightsWebTracking"
    type="Microsoft.ApplicationInsights.Web.ApplicationInsightsHttpModule,
    Microsoft.AI.Web" precondition="managedHandler" />
</modules>
```

## Azure AD OAuth 2.0 Authentication

Firstly you will need to setup some users in your tenant if you haven't already done so. Search for the Users blade in your azure portal.

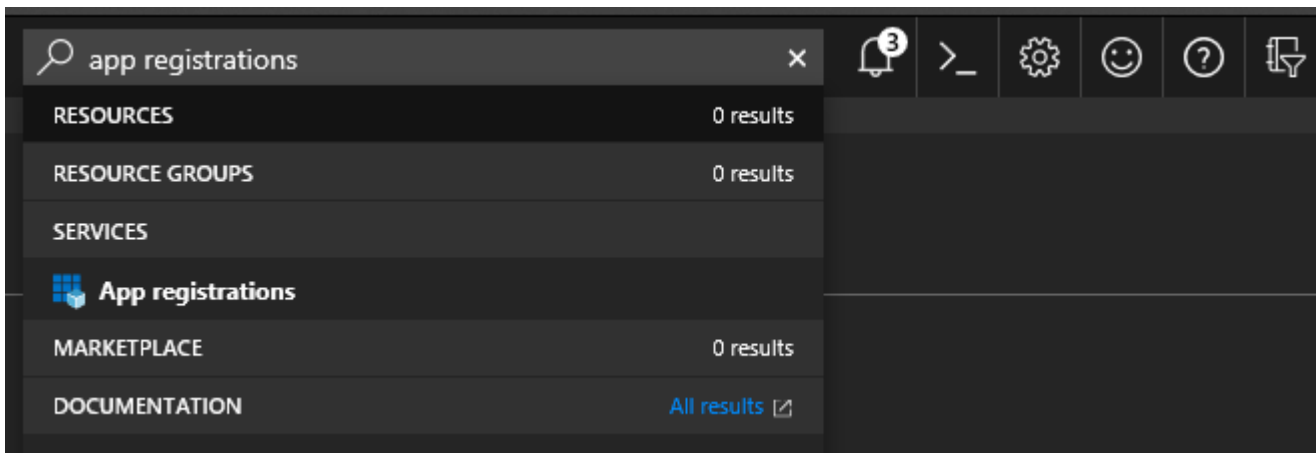


Click on the add button and add a new user



The screenshot shows the 'User' configuration page in the Azure portal. The breadcrumb navigation at the top reads 'Home > Users - All users > User'. The page title is 'User'. Below the title, there are two required fields: 'Name' and 'User name'. The 'Name' field contains 'TestUser' and has a green checkmark. The 'User name' field contains 'TestUser@mytenant.onmicosot.com'. Below these fields, there are four expandable sections: 'Profile' (Not configured), 'Properties' (Default), 'Groups' (0 groups selected), and 'Directory role' (User). Each section has a right-pointing chevron icon.

Next you need to register your data service in your tenant. In your azure portal search for App Registrations.



The screenshot shows the search results for 'app registrations' in the Azure portal. The search bar at the top contains 'app registrations'. Below the search bar, there is a table with the following rows: 'RESOURCES' (0 results), 'RESOURCE GROUPS' (0 results), 'SERVICES' (0 results), 'App registrations' (0 results), 'MARKETPLACE' (0 results), and 'DOCUMENTATION' (All results). The 'App registrations' row is highlighted. To the right of the table, there are several icons: a bell with a '3' badge, a '>\_' icon, a gear icon, a smiley face icon, a question mark icon, and a document icon.

Then click on the add button and create a new web api registration.

Create

\* Name ⓘ  
MyDataService ✓

Application type ⓘ  
Web app / API ▼

\* Sign-on URL ⓘ  
https://mydataservice.azurewebsites.net ✓

Make a note of the Sign-on URL and the generated application id GUID. You will need them in the dataservice. You will also need the Sign-on URL for any application that is consuming data from the WCF data service.

Back in the data service, install the following nuget.

Hide Copy Code

```
PM> Install-Package System.IdentityModel.Protocols.OpenIdConnect
```

Then add a new **OAuthProtectionModule** this class serves the same purpose as the **BasicAuthenticationModule** and that is to attach the AuthenticateRequest event. Inside the event handler the **OAuthAuthenticationProvider** class is called to do the authentication or return an error response if applicable.

Hide Shrink ▲ Copy Code

```
/// <summary>
/// This class is an IHttpModule is used to check the access token on every incoming request to the
site.
/// </summary>
public class OAuthProtectionModule : IHttpModule
{
    /// <summary>
    /// This method is used to do all the initialization for this class.
    /// </summary>
    /// <param name="context">The <see cref="HttpApplication"/> object which contains this module.
</param>
    public void Init(HttpApplication context)
    {
        context.AuthenticateRequest += OnAuthenticateRequest;
    }

    /// <summary>
    /// Handle the HTTP pipeline AuthenticateRequest event, after ensuring that the module has been
initialized.
    /// </summary>
    /// <param name="sender">Sender of this event.</param>
    /// <param name="args">Event arguments.</param>
    void OnAuthenticateRequest(object sender, EventArgs args)
    {
        //Unbox the application.
        HttpApplication application = (HttpApplication)sender;
```

```

        //Send to provider for authentication.
        if (!OAuthAuthenticationProvider.Authenticate(
            out int statusCode,
            out string httpStatus,
            out string wwwAuthenticateResponse))
        {
            //Set the status and status code.
            application.Context.Response.Status = httpStatus;
            application.Context.Response.StatusCode = statusCode;

            //If there is a WWW-Authenticate payload.
            if (!String.IsNullOrEmpty(wwwAuthenticateResponse))
            {
                //Add the Authenticate header.
                application.Context.Response.AddHeader("WWW-Authenticate", wwwAuthenticateResponse);
            }
            application.CompleteRequest();
        }
    }

    public void Dispose() { }
}

```

Here is the full definition of the **OAuthAuthenticationProvider** class. It shares some similarity with the **BasicAuthenticationProvider** in that it has an **Authenticate** method which passes the authorization header to the **TryGetPrinciple** method and assigns the resulting principle (if any) to the user context. Note that the bulk of this code comes from the following microsoft sample <https://azure.microsoft.com/en-us/resources/samples/active-directory-dotnet-webapi-manual-jwt-validation/>

With OAuth unlike basic authentication we are not looking for user credentials in the authorization header. The authorization header for oAuth2.0 should be the word "bearer" followed by a space followed by the base64url encoded access token. In the authenticate method I am stripping the word bearer out of the authorization header with a simply substring before passing to the TryGetPrinciple method.

Note that you need to fill in your tenant id e.g. mytenant.onmicrosoft.com, your dataservice application id GUID and your dataservice sign-on url.

Hide Shrink ▲ Copy Code

```

public static class OAuthAuthenticationProvider
{
    #region Fields

    /// <summary>
    /// The name of the azure AD tenant.
    /// </summary>
    static string tenant = "%YourTenantName%";

    /// <summary>
    /// The signin authority.
    /// </summary>
    static string authority = $"https://login.microsoftonline.com/{tenant}";

    /// <summary>
    /// OpenIdConnect configuration
    /// </summary>
    static ConfigurationManager<OpenIdConnectConfiguration> configurationManager =
        new ConfigurationManager<OpenIdConnectConfiguration>($"{{authority}}/.well-known/openid-configuration",
            new OpenIdConnectConfigurationRetriever());

    /// <summary>
    /// The application id of the web app and the app id url.
    /// </summary>
    static string[] audiences = { "%DataService Application ID%", "%DataService Sign-On URL%" };

    /// <summary>
    ///

```



```

    /// </summary>
    static string scopeClaimType = "http://schemas.microsoft.com/identity/claims/scope";

    /// <summary>
    /// The standard www-authenticate response header.
    /// </summary>
    static string wwwAuthenticate = $"Bearer realm=\"{audiences[1]}\", authorization_uri=\"{authority}\", resource_id=\"{audiences[1]}\"";

    #endregion

    /// <summary>
    /// Authenticate and the set the current http context user.
    /// </summary>
    /// <param name="context">The http context.</param>
    /// <returns>True if the user has been authenticated.</returns>
    public static bool Authenticate(out int statusCode, out string httpStatus, out string
wwwAuthenticateResponse)
    {
        //Set initial result.
        bool result = false;

        //If no authorization was provided.
        if (!HttpContext.Current.Request.Headers.AllKeys.Contains("Authorization"))
        {
            httpStatus = "401 Unauthorized";
            statusCode = (int)HttpStatusCode.Unauthorized;
            wwwAuthenticateResponse = wwwAuthenticate;
        }
        //Try to resolve the claims principle.
        else if (TryGetPrincipal(HttpContext.Current.Request.Headers["Authorization"].Substring(7),
            out IPrincipal principal,
            out statusCode,
            out httpStatus,
            out wwwAuthenticateResponse))
        {
            //Set the claims principle.
            HttpContext.Current.User = principal;
            Thread.CurrentPrincipal = principal;
            result = true;
        }

        //Return the result.
        return result;
    }

    /// <summary>
    /// This method parses the incoming token and validates it.
    /// </summary>
    /// <param name="accessToken">The incoming access token.</param>
    /// <param name="error">This out paramter is set if any error occurs.</param>
    /// <returns>True on success, False on error.</returns>
    static bool TryGetPrincipal(string accessToken, out IPrincipal principal, out int statusCode, out
string httpStatus, out string wwwAuthenticateResponse)
    {
        bool overallResult = false;
        principal = null;
        statusCode = 0;
        httpStatus = null;
        wwwAuthenticateResponse = null;

        #if DEBUG
            //Show token in execeptions.
            IdentityModelEventSource.ShowPII = true;
        #endif

        try
        {
            //Retrieve the configuration for the tennent.
            OpenIdConnectConfiguration config = GetConfigurationNonAsync();

```

```

        // validate the token
        var claimsPrincipal = new JwtSecurityTokenHandler().ValidateToken(accessToken,
            new TokenValidationParameters
            {
                ValidateAudiences = audiences,
                ValidIssuer = config.Issuer,
                IssuerSigningKeys = config.SigningKeys
            }, out SecurityToken validatedToken); ;

        // If the token is scoped, verify that required permission is set in the scope claim.
        if (claimsPrincipal.FindFirst(scopeClaimType) != null &&
            claimsPrincipal.FindFirst(scopeClaimType).Value != "user_impersonation")
        {
            statusCode = (int)HttpStatusCode.Forbidden;
            httpStatus = "403 Forbidden";
            wwwAuthenticateResponse = wwwAuthenticate;
        }
        else
        {
            //Set out parameter.
            principal = claimsPrincipal;

            //Indicate overall success.
            overallResult = true;
        }
    }
    catch (SecurityTokenException ex)
    {
        statusCode = (int)HttpStatusCode.Unauthorized;
        httpStatus = "401 Unauthorized";
        wwwAuthenticateResponse = wwwAuthenticate + $" error=\"invalid_token\",
error_description=\"{ex.Message}\"";
    }
    catch (Exception)
    {
        statusCode = (int)HttpStatusCode.InternalServerError;
        httpStatus = "500 InternalServerError";
    }

    //Return result.
    return overallResult;
}

static string BuildWWWAuthenticateResponseHeader()
{
    return $"Bearer authorization_uri=\"{authority}\", resource_id=\"{audiences[0]}";
}

/// <summary>
/// Retrieve configuration information used to validate the access token.
/// </summary>
static OpenIdConnectConfiguration GetConfigurationNonAsync()
{
    //Get the configuration.
    OpenIdConnectConfiguration config = null;
    Task.Run(async () =>
    {
        // Get open id connect configuration.
        config = await configurationManager.GetConfigurationAsync();
    }).Wait();

    //Return the configuration.
    return config;
}
}

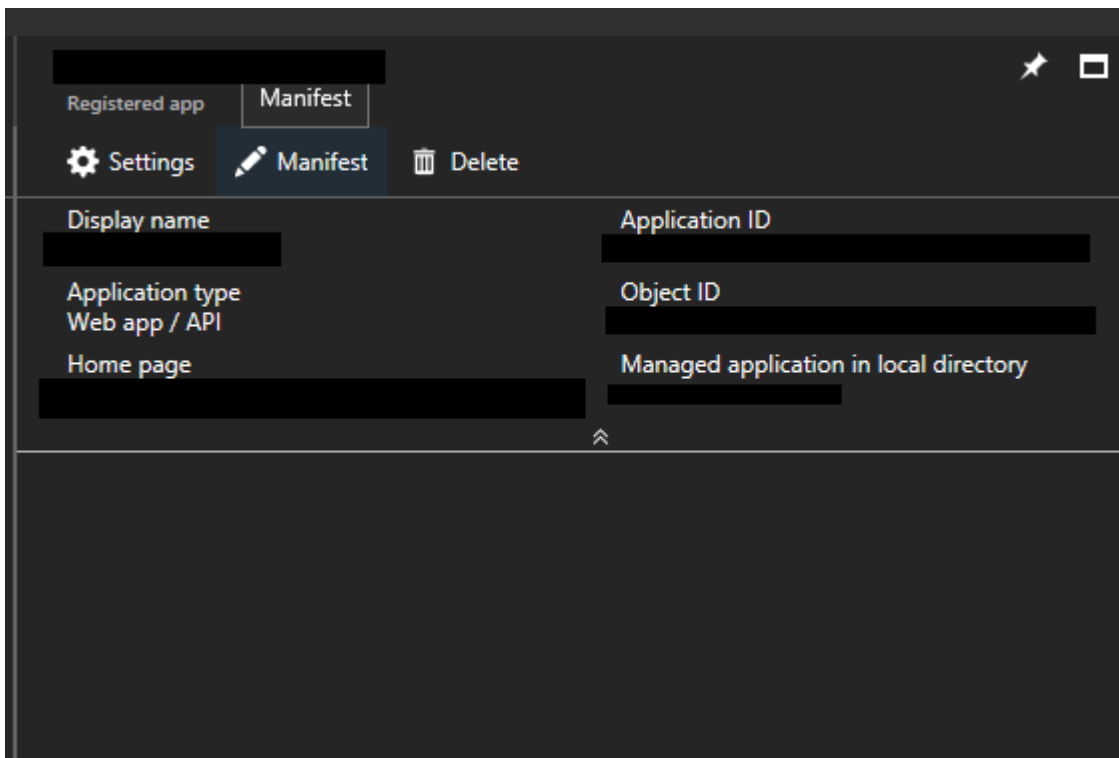
```

Finally you will want to change the web.config to redirect to the new authentication model.

```
<system.webServer>
  <modules runAllManagedModulesForAllRequests="true">
    <!--<add name="BasicAuthentication" type="UtilitiesWcfService.BasicAuthenticationModule" />-->
    <add name="OAuthProtectionModule" type="UtilitiesWcfService.OAuthProtectionModule" />
  </modules>
```

## AD Roles

To have roles appear in the access token and thus be able from the validated claims principle you need to first setup the roles themselves. In the app registrations blade in the azure portal, select your app and click on the manifest button.



Alter the roles field in the json manifest as desired. Here is an example spec for 3 roles. Note that the GUIDs can be anything; I used the following site <https://guidgenerator.com/>

```
{
  "appRoles": [
    {
      "allowedMemberTypes": [
        "User"
      ],
      "displayName": "LocalUser",
      "id": "04a25ad2-ebd2-46fc-b85a-646ef2c9c5c9",
      "isEnabled": true,
      "description": "Add or edit menus for a specific site.",
      "value": "LocalUser"
    },
    {
      "allowedMemberTypes": [
        "User"
      ]
    }
  ]
}
```

```

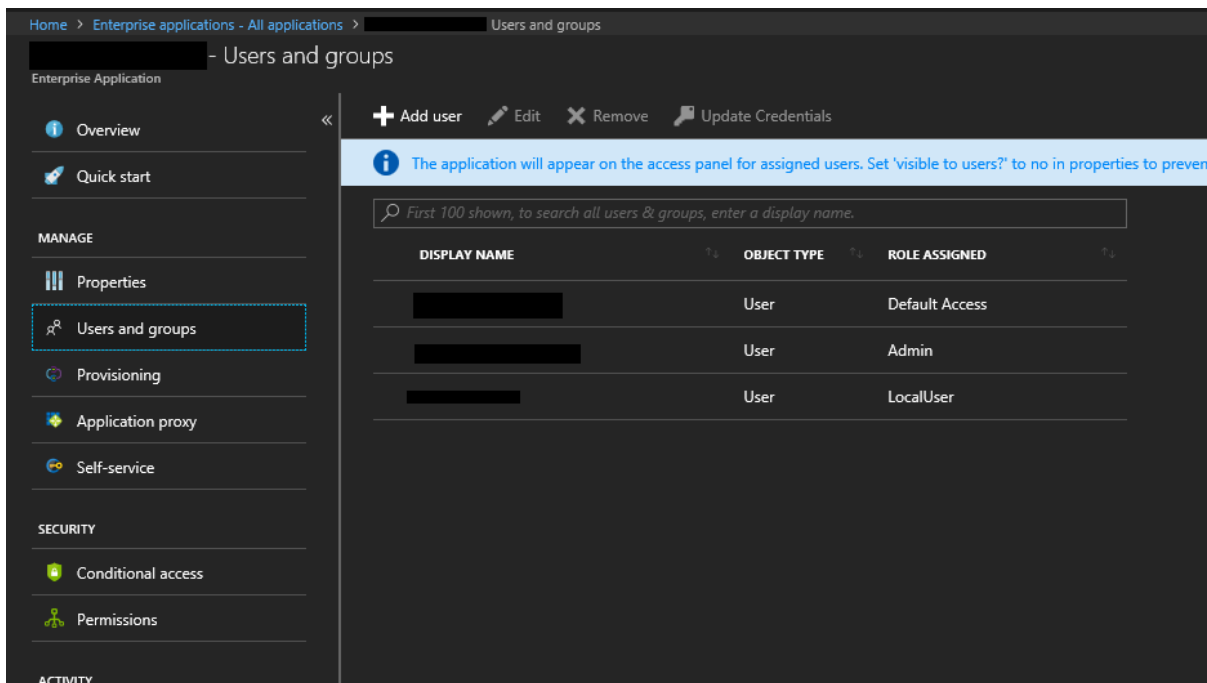
],
  "displayName": "GroupUser",
  "id": "434723c9-90ef-4320-bd68-cad24ca66c92",
  "isEnabled": true,
  "description": "Add or edit sites and site menus for a specific group",
  "value": "GroupUser"
},
{
  "allowedMemberTypes": [
    "User"
  ],
  "displayName": "Admin",
  "id": "06579139-bd57-402c-b29f-b69532de2117",
  "isEnabled": true,
  "description": "Add or edit groups, sites and site menus.",
  "value": "Admin"
}
],

```

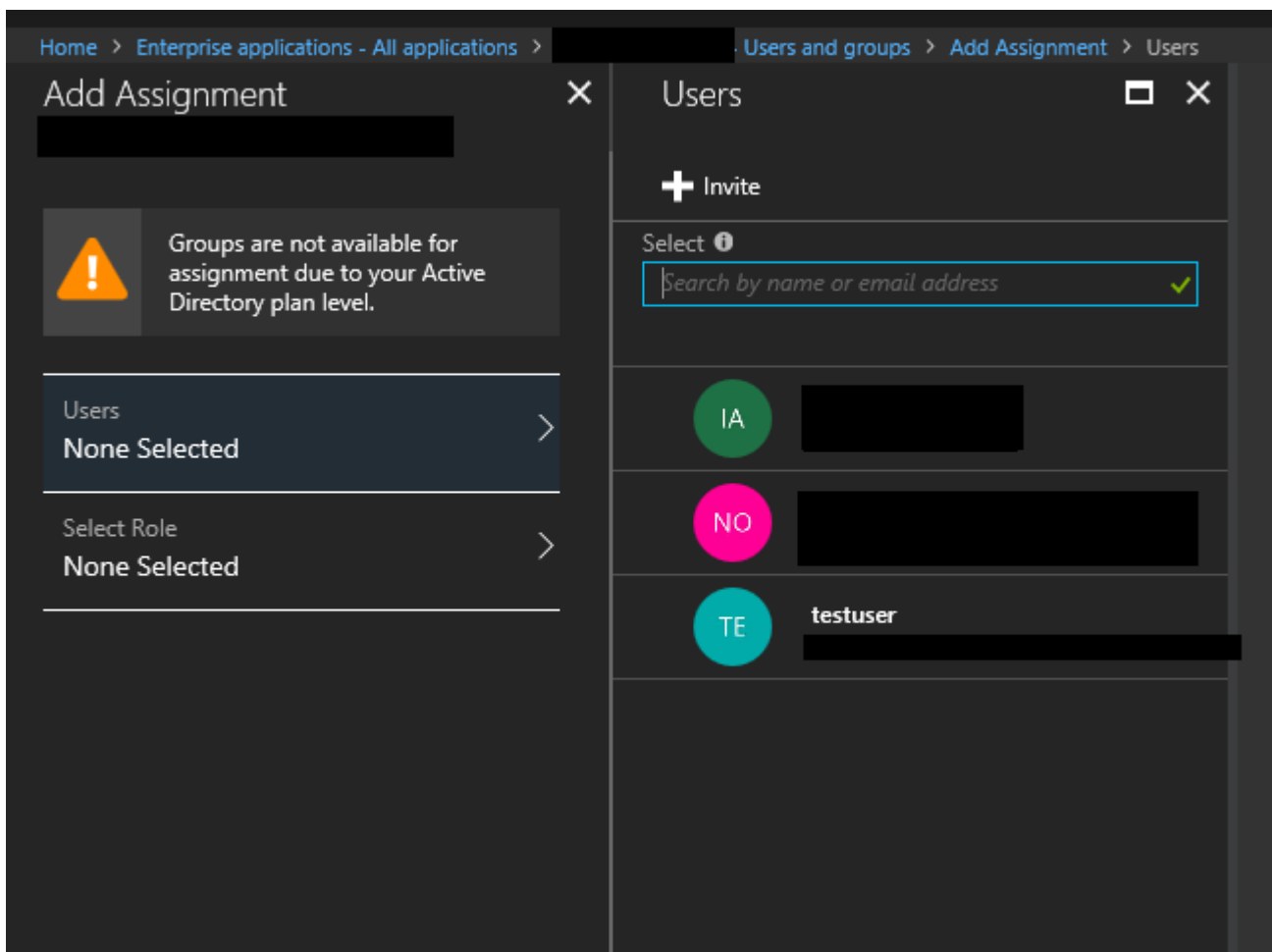
Next you need to assign users to a particular role. In the azure portal search for "Enterprise Applications"



Select your data service application and then click on Users and Groups



Click on the Add User button and assign users to roles



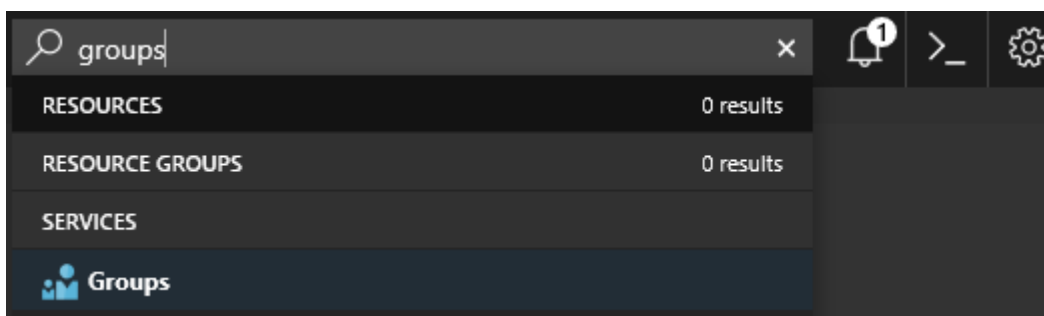
Now once you have a claims principle object you can use the `IsInRole` method to check role membership.

[Hide](#) [Copy Code](#)

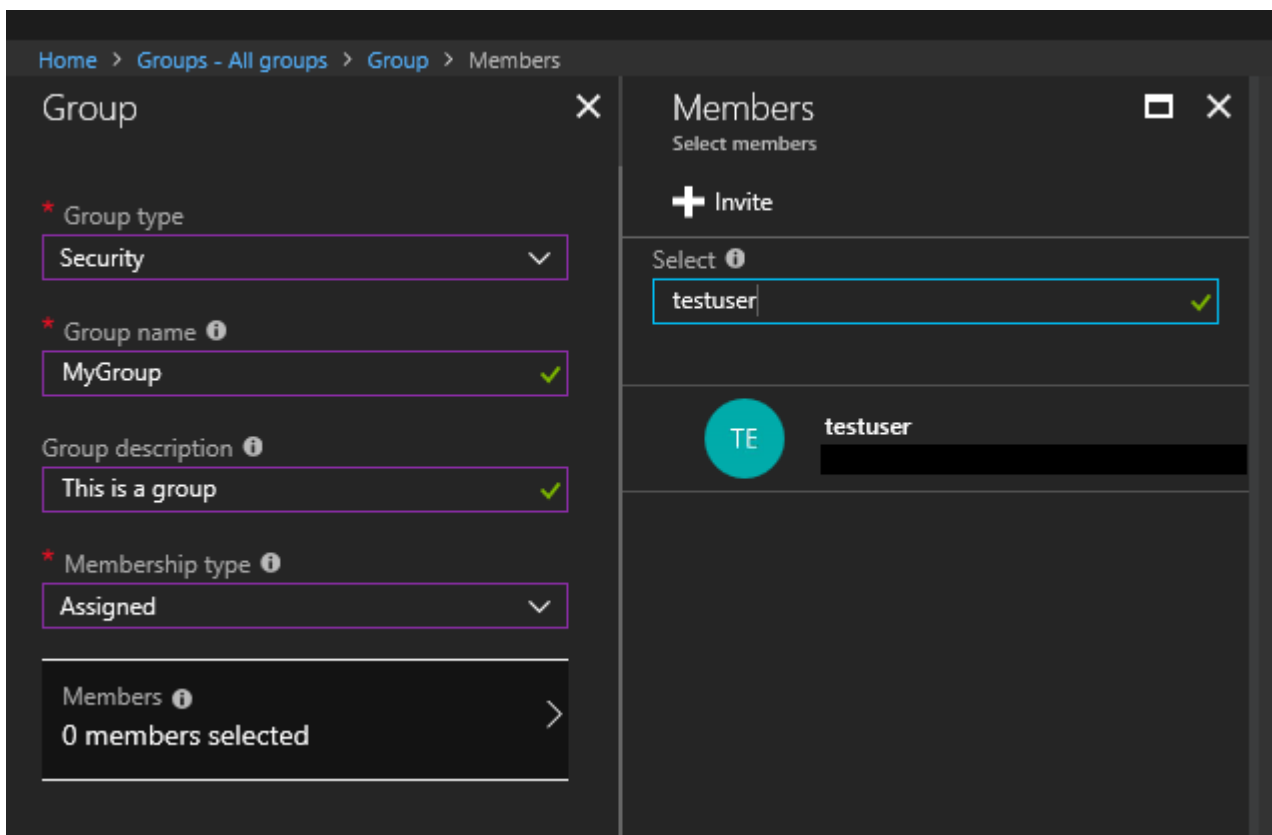
```
claimsPrincipal.IsInRole("GroupUser")
```

## AD Groups

In order to have groups in the access token and thus be available from the validated claims principle firstly you need to create some groups and assign some users. In your azure portal search groups



Then create groups and assign users



In order for the groups to appear in the access token and resulting validated claims principle you must edit the manifest of your data service application again. Below the approles field find the "groupMembershipClaims" field and change the value to "SecurityGroup"

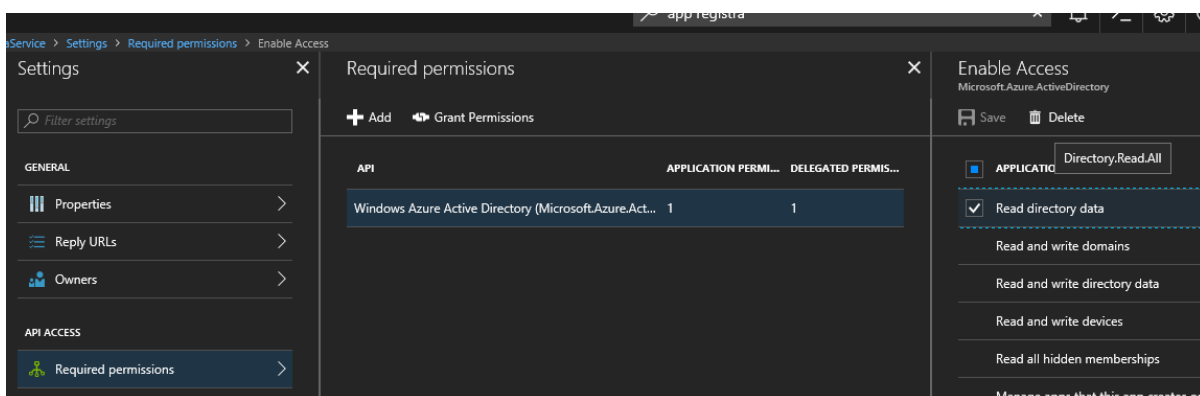
[Hide](#) [Copy Code](#)

```
"groupMembershipClaims": "SecurityGroup",
```

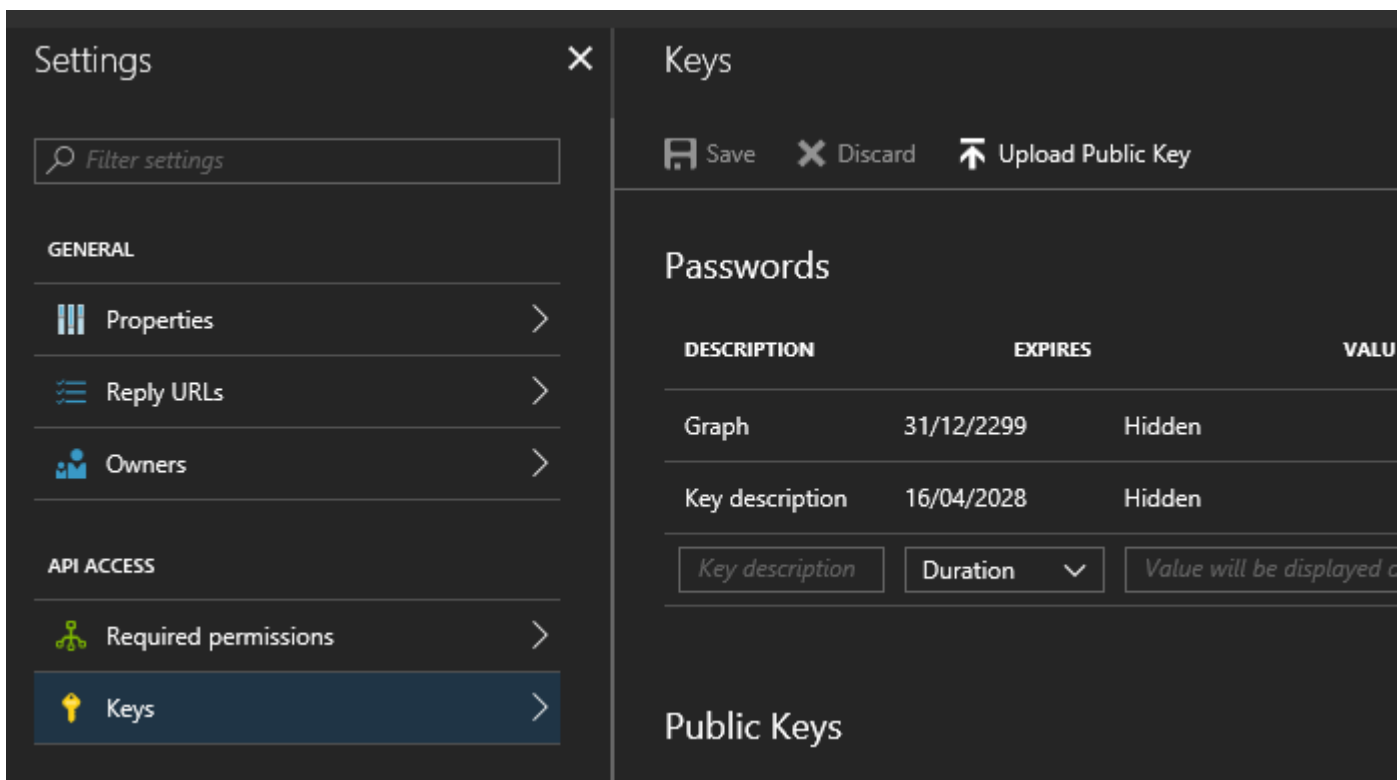
## Resolving group object Ids

Unfortunately what you get in the access token and the resulting claims principle is the group object id guid as opposed to the group name. In order to get the name of the group you must query the graph api.

Firstly you need to allow you dataservice the permission to query the graph api by itself. In the azure portal go to app registrations, select your application, go to settings and then required permissions. Open the Active directory API and enable the application permission. Directory.Read.All



Next go to the "Keys" blade which is directly below "Required Permissions". Create a new key, specify your own key name and desired expiry. Make a note of the key before it gets hidden.



Back in the data service you need to install the MSAL client nuget package

```
PM> Install-Package Microsoft.Identity.Client
```

Then in the `OAuthAuthenticationProvider` class declare an instance of the msal confidential client and also the secret key from the azure keys blade.

Hide Copy Code

```
/// <summary>
/// MSAL client for graph API lookups.
/// </summary>
static ConfidentialClientApplication msaClient;

/// <summary>
/// The password from App Registrations - App Name - Settings - Keys
/// </summary>
static string clientSecret = "%Secret Key%";
```

Then add the following methods. The `AddGroupNameClaim` will accept a claims principle and for each group object id claim will add the corresponding `group_name` claim by querying the graph api.

Hide Shrink ▲ Copy Code

```
/// <summary>
/// Updates the claims principle with the group name claim.
/// </summary>
/// <param name="claimsPrincipal">The claims principle to update.</param>
public static void AddGroupNameClaim(ClaimsPrincipal claimsPrincipal)
{
    (claimsPrincipal.Identity as ClaimsIdentity).AddClaims(
        claimsPrincipal.FindAll("groups").Select(r =>
            new Claim("group_name", GetGroupNameById(r.Value))));
}

/// <summary>
/// Gets the group name for the specified object id.
/// </summary>
/// <param name="objectId">The guid for the group.</param>
public static string GetGroupNameById(string objectId)
```

```

{
    //Initialize graph client.
    ActiveDirectoryClient activeDirectoryClient = new ActiveDirectoryClient(new
Uri($"https://graph.windows.net/{tenant}"), async () =>
    {
        return await Task.Run(async () =>
        {
            //If the msal client does not exist.
            if (msaClient == null)
            {
                //Initialize the msal client.
                msaClient = new ConfidentialClientApplication(
                    audiences[0],
                    authority,
                    audiences[1],
                    new Microsoft.Identity.Client.ClientCredential(clientSecret),
                    new Microsoft.Identity.Client.TokenCache(),
                    new Microsoft.Identity.Client.TokenCache());
            }

            //Get the token.
            var authResult = await msaClient.AcquireTokenForClientAsync(new string[] {
"https://graph.windows.net/.default" });
            return authResult.AccessToken;
        });
    });

    IGroup group = null;
    Task.Run(async () =>
    {
        //Get the group object.
        group = await activeDirectoryClient.Groups.GetByObjectId(objectId).ExecuteAsync();
    }).Wait();

    //Return the group name.
    return group?.DisplayName;
}

```

And finally change the Authenticate method to call the new method so the validated claims principle is updated with the group\_name claim.

[Hide](#) [Copy Code](#)

```

// If the token is scoped, verify that required permission is set in the scope claim.
if (claimsPrincipal.FindFirst(scopeClaimType) != null &&
    claimsPrincipal.FindFirst(scopeClaimType).Value != "user_impersonation")
{
    statusCode = (int)HttpStatusCode.Forbidden;
    httpStatus = "403 Forbidden";
    wwwAuthenticateResponse = wwwAuthenticate;
}
else
{
    //Update the claims principle with the group names.
    AddGroupNameClaim(claimsPrincipal);

    //Set out parameter.
    principal = claimsPrincipal;

    //Indicate overall success.
    overallResult = true;
}

```

So If subsequently working with the claims principle you can get the group name via the group\_name claim.

[Hide](#) [Copy Code](#)

```
claimsprinciple.FindFirst("group_name)?.Value
```



## Store proc Entity with Multiple Result Sets.

I mentioned that my stored procedure `aspnet_GetUserCredentials` had multiple result sets. In order to achieve this, you must again edit the model that defines the stored procedure via the XML editor and make some changes.

In my case, my stored procedure accepted two input parameters (the application name and the username) and returned 3 result sets:

1. The first result set contained fields for `username`, `password` and `passwordsalt`.
2. The second result set contained fields for `rolename`.
3. The third result set contained fields for `permissionId`.

Below are the modifications I needed.

Hide Shrink ▲ Copy Code

```
<EntityContainer Name="AuthenticationEntities"

annotation:LazyLoadingEnabled="true" >
  <FunctionImport Name="aspnet_GetUserCredentials">
    <ReturnType Type="Collection(UtilitiesLightswitchModel.User_SprocResult)" />
    <ReturnType Type="Collection(UtilitiesLightswitchModel.Role_SprocResult)" />
    <ReturnType Type="Collection(UtilitiesLightswitchModel.Permission_SprocResult)" />
    <Parameter Name="ApplicationName" Mode="In" Type="String" />
    <Parameter Name="UserName"

      Mode="In" Type="String" />
  </FunctionImport>
</EntityContainer>
<ComplexType Name="User_SprocResult">
  <Property Type="String" Name="UserName"

    Nullable="false" MaxLength="256" />
  <Property Type="String" Name="Password"

    Nullable="false" MaxLength="128" />
  <Property Type="String" Name="PasswordSalt"

    Nullable="false" MaxLength="128" />
</ComplexType>
<ComplexType Name="Role_SprocResult">
  <Property Type="String" Name="RoleName"

    Nullable="false" MaxLength="256" />
</ComplexType>
<ComplexType Name="Permission_SprocResult">
  <Property Type="String" Name="PermissionId"

    Nullable="false" MaxLength="322" />
</ComplexType>
```

...

Hide Copy Code

```
<FunctionImportMapping FunctionImportName="aspnet_GetUserCredentials"

FunctionName="UtilitiesLightswitchModel.Store.aspnet_GetUserCredentials">
  <ResultMapping>
    <ComplexTypeMapping TypeName="UtilitiesLightswitchModel.User_SprocResult">
      <ScalarProperty Name="UserName" ColumnName="UserName" />
      <ScalarProperty Name="Password" ColumnName="Password" />
      <ScalarProperty Name="PasswordSalt" ColumnName="PasswordSalt" />
    </ComplexTypeMapping>
  </ResultMapping>
  <ResultMapping>
    <ComplexTypeMapping TypeName="UtilitiesLightswitchModel.Role_SprocResult">
      <ScalarProperty Name="RoleName" ColumnName="RoleName" />
    </ComplexTypeMapping>
```

```

</ResultMapping>
<ResultMapping>
  <ComplexTypeMapping TypeName="UtilitiesLightswitchModel.Permission_SprocResult">
    <ScalarProperty Name="PermissionId" ColumnName="PermissionId" />
  </ComplexTypeMapping>
</ResultMapping>
</FunctionImportMapping>

```

## Filtering Data

You might want to Filter the entity based on user. Now that we have set the user context via the authentication module, we can now use the user context to filter the result set. You can filter entities by adding a query interceptor. This code below needs to be added to the main service class in my case *WcfDataService1.svc*. This example shows filtering the **Groups** entity depending on what role the user belongs to.

[Hide](#) [Copy Code](#)

```

/// <summary>
/// Intercept entity query.
/// </summary>
/// <returns>Filtered recordset.</returns>
[QueryInterceptor("Groups")]
public Expression<Func<Group, bool>> OnQueryGroups()
{
    //If this is a group user.
    if (HttpContext.Current.User.IsInRole("GroupUser"))
    {
        //Filter for the specific group id.
        return (Group e) => e.GroupID == HttpContext.Current.User.Identity.Name;
    }
    //If this is a local user.
    else if (HttpContext.Current.User.IsInRole("LocalUser"))
    {
        //Filter for the group containing their site id.
        return (Group e) => e.Sites.Any(r => r.SiteID == HttpContext.Current.User.Identity.Name);
    }
    else
    {
        //Return all.
        return (Group e) => true;
    }
}

```

You may also want to base your data access rules on the user. To do that, we can add a **ChangeInterceptor**. In the code below, the user requires the permission **CanAddOrEditGroups** in order to make changes to the **Groups** entity.

[Hide](#) [Copy Code](#)

```

[ChangeInterceptor("Groups")]
public void OnChangeGroups(Group group, UpdateOperations operations)
{
    //Unbox the user principle.
    var u = (BasicAuthenticationProvider.CustomPrincipal)HttpContext.Current.User;

    if (!u.HasPermission("CanAddOrEditGroups"))
    {
        throw new DataServiceException(400, "You do not have permission to add or edit new groups.");
    }
}

```

## Alternative Hosting

You might for whatever reason want to host the data service offline. It is possible to host the webservice in any .NET program such as a windows service or console application via the `WebServiceHost` class.

1. Start by adding a new windows service or console application project to the existing solution.
2. Add the entity framework provider for OData via nugget command line which should also add the Microsoft.Data.Services APIs.

Hide Copy Code

```
PM> Install-Package Microsoft.OData.EntityFrameworkProvider -Pre
```

3. Add an application configuration file to the project (if it doesn't already exist) and copy the `configSections`, `system.serviceModel`, `connectionStrings`, `entityFramework` and `runtime` configuration sections from the web.config file in the dataservice project to the app.config of the new project.

4. Within the `system.serviceModel` section add the following element.

Hide Copy Code

```
<bindings>
  <webHttpBinding>
    <binding>
      <security mode="TransportCredentialOnly">
        <transport clientCredentialType="Basic" />
      </security>
    </binding>
  </webHttpBinding>
</bindings>
```

The full app.config should look something like this. Note in this example I've blanked the connection strings for both the data entity model and the authentication entity model but obviously in reality they would remain intact.

Hide Shrink ▲ Copy Code

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <configSections>
    <!-- For more information on Entity Framework configuration, visit http://go.microsoft.com/fwlink/?
    LinkID=237468 -->
    <section name="entityFramework" type="System.Data.Entity.Internal.ConfigFile.EntityFrameworkSection,
    EntityFramework, Version=6.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089"
    requirePermission="false" />
    <!-- For more information on Entity Framework configuration, visit http://go.microsoft.com/fwlink/?
    LinkID=237468 -->
  </configSections>
  <startup>
    <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.5.2" />
  </startup>
  <system.serviceModel>
    <behaviors>
      <serviceBehaviors>
        <behavior>
          <!-- To avoid disclosing metadata information, set the values below to false before deployment -->
          <serviceMetadata httpGetEnabled="true" httpsGetEnabled="true" />
          <!-- To receive exception details in faults for debugging purposes, set the value below to true.
          Set to false before deployment to avoid disclosing exception information -->
          <serviceDebug includeExceptionDetailInFaults="true" httpHelpPageEnabled="true" />
        </behavior>
      </serviceBehaviors>
    </behaviors>
    <protocolMapping>
      <add binding="basicHttpsBinding" scheme="https" />
    </protocolMapping>
    <serviceHostingEnvironment aspNetCompatibilityEnabled="true" multipleSiteBindingsEnabled="true" />
  </system.serviceModel>
  <bindings>
    <webHttpBinding>
      <binding>
        <security mode="TransportCredentialOnly">
```

```

        <transport clientCredentialType="Basic" />
    </security>
</binding>
</webHttpBinding>
</bindings>
</system.serviceModel>
<connectionStrings>
    <add name="dfsdffsEntities" connectionString="" />
    <add name="AuthenticationEntities" connectionString="" />
</connectionStrings>
<entityFramework>
    <defaultConnectionFactory type="System.Data.Entity.Infrastructure.LocalDbConnectionFactory,
EntityFramework">
        <parameters>
            <parameter value="mssqllocaldb" />
        </parameters>
    </defaultConnectionFactory>
    <providers>
        <provider invariantName="System.Data.SqlClient"
type="System.Data.Entity.SqlServer.SqlProviderServices, EntityFramework.SqlServer" />
    </providers>
</entityFramework>
<runtime>
    <assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">
        <dependentAssembly>
            <assemblyIdentity name="Microsoft.Data.Services" publicKeyToken="31bf3856ad364e35"
culture="neutral" />
            <bindingRedirect oldVersion="0.0.0.0-5.7.0.0" newVersion="5.7.0.0" />
        </dependentAssembly>
        <dependentAssembly>
            <assemblyIdentity name="Microsoft.Data.Services.Client" publicKeyToken="31bf3856ad364e35"
culture="neutral" />
            <bindingRedirect oldVersion="0.0.0.0-5.7.0.0" newVersion="5.7.0.0" />
        </dependentAssembly>
        <dependentAssembly>
            <assemblyIdentity name="Microsoft.Data.Edm" publicKeyToken="31bf3856ad364e35" culture="neutral" />
            <bindingRedirect oldVersion="0.0.0.0-5.7.0.0" newVersion="5.7.0.0" />
        </dependentAssembly>
        <dependentAssembly>
            <assemblyIdentity name="System.Spatial" publicKeyToken="31bf3856ad364e35" culture="neutral" />
            <bindingRedirect oldVersion="0.0.0.0-5.7.0.0" newVersion="5.7.0.0" />
        </dependentAssembly>
    </assemblyBinding>
</runtime>
</configuration>

```

4. Add a reference to your dataservice library i.e. the dll produced by the original data service project, to the new project.

5. Here is complete code for a console application.

Hide Shrink ▲ Copy Code

```

using System;
using System.Linq;
using System.ServiceModel;
using System.ServiceModel.Web;
using System.ServiceModel.Channels;
using System.IdentityModel.Selectors;
using System.Configuration;

namespace WcfConsole
{
    class Program
    {
        static WebServiceHost serviceHost = null;

        static void Main(string[] args)
        {
            // Create a ServiceHost for the data service type and

```

```

        // provide the base address.
        serviceHost = new WebServiceHost(typeof(MyWcfServiceNameSpace.MyWcfDataService), new Uri[] {
new Uri("http://localhost:8195/MyDataService.svc") });
        serviceHost.Authentication.ServiceAuthenticationManager = new MyAuthentication();
        serviceHost.Credentials.UserNameAuthentication.UserNamePasswordValidationMode =
System.ServiceModel.Security.UserNamePasswordValidationMode.Custom;
        serviceHost.Credentials.UserNameAuthentication.CustomUserNamePasswordValidator = new
CustomUserNamePasswordValidator();

        // Open the ServiceHostBase to create listeners and start
        // listening for messages.
        serviceHost.Open();

        //Keep service alive.
        Console.ReadKey();

        //Close the service.
        serviceHost.Close();
    }

    /// <summary>
    /// Custom username and password validation.
    /// </summary>
    public class CustomUserNamePasswordValidator : UserNamePasswordValidator
    {
        /// <summary>
        /// Validate the username and password.
        /// </summary>
        /// <param name="userName">The username delivered in the message header.</param>
        /// <param name="password">The password delivered in the message header.</param>
        public override void Validate(string userName, string password)
        {
            //Store the custom principle.
            System.Security.Principal.IPrincipal principal;

            //If the user is authenticated.
            if (MyWcfServiceNameSpace.BasicAuthenticationProvider.TryGetPrincipal(new string[] {
userName, password }, out principal))
            {
                //Store the custom principle.
                System.Threading.Thread.CurrentPrincipal = principal;
            }
            else
            {
                //Throw an error message.
                throw new FaultException("Unknown Username or Incorrect Password");
            }
        }
    }

    public class MyAuthentication : ServiceAuthenticationManager
    {
        /// <summary>
        /// Store the custom authorization principle in the
        /// message properties so its available to query intercepters.
        /// </summary>
        /// <param name="authPolicy">The authorization policy collection.</param>
        /// <param name="listenUri">The uri.</param>
        /// <param name="message">The message received.</param>
        /// <returns></returns>
        public override
System.Collections.ObjectModel.ReadOnlyCollection<System.IdentityModel.Policy.IAuthorizationPolicy>
Authenticate(
System.Collections.ObjectModel.ReadOnlyCollection<System.IdentityModel.Policy.IAuthorizationPolicy>
authPolicy,
        Uri listenUri, ref Message message)
        {
            //Store the custom principle in the message properties so its available to query
interceptors.

```

```

        OperationContext.Current.IncomingMessageProperties.Add("Principal",
System.Threading.Thread.CurrentPrincipal);

        //Return the policy.
        return authPolicy;
    }
}
}
}

```

5. Here is complete code for a windows service

Hide Shrink ▲ Copy Code

```

using System;
using System.Linq;
using System.ComponentModel;
using System.Diagnostics;
using System.ServiceProcess;
using System.ServiceModel;
using System.Configuration;
using System.Configuration.Install;
using System.ServiceModel.Web;
using System.ServiceModel.Channels;
using System.IdentityModel.Selectors;

namespace WcfWindowsService
{
    public partial class Service1 : ServiceBase
    {
        /// <summary>
        /// Instance of web service host.
        /// </summary>
        WebServiceHost serviceHost = null;

        /// <summary>
        /// Default constructor.
        /// </summary>
        public Service1()
        {
            InitializeComponent();
        }

        /// <summary>
        /// On startup.
        /// </summary>
        /// <param name="args">The starting arguments for the service.</param>
        protected override void OnStart(string[] args)
        {
            //If the service is not null, close it.
            serviceHost?.Close();

            // Create a ServiceHost for the data service type and
            // provide the base address.
            serviceHost = new WebServiceHost(typeof(MyWcfServiceNameSpace.MyWcfDataService), new Uri[] {
new Uri("http://localhost:8195/MyDataService.svc") });
            serviceHost.Authentication.ServiceAuthenticationManager = new MyAuthentication();
            serviceHost.Credentials.UserNameAuthentication.UserNamePasswordValidationMode =
System.ServiceModel.Security.UserNamePasswordValidationMode.Custom;
            serviceHost.Credentials.UserNameAuthentication.CustomUserNamePasswordValidator = new
CustomUserNamePasswordValidator();

            // Open the ServiceHostBase to create listeners and start
            // listening for messages.
            serviceHost.Open();
        }
    }
}

```

```

    /// <summary>
    /// Close the service.
    /// </summary>
    protected override void OnStop()
    {
        serviceHost?.Close();
        serviceHost = null;
    }
}

/// <summary>
/// Custom username and password validation.
/// </summary>
public class CustomUserNamePasswordValidator : UserNamePasswordValidator
{
    /// <summary>
    /// Validate the username and password.
    /// </summary>
    /// <param name="userName">The username delivered in the message header.</param>
    /// <param name="password">The password delivered in the message header.</param>
    public override void Validate(string userName, string password)
    {
        //Store the custom principle.
        System.Security.Principal.IPrincipal principal;

        //If the user is authenticated.
        if (MyWcfServiceNameSpace.BasicAuthenticationProvider.TryGetPrincipal(new string[] { userName,
password }, out principal))
        {
            //Store the custom principle.
            System.Threading.Thread.CurrentPrincipal = principal;
        }
        else
        {
            //Throw an error message.
            throw new FaultException("Unknown Username or Incorrect Password");
        }
    }
}

/// <summary>
/// Custom authentication manager to embed authentication principle in message properties.
/// </summary>
public class MyAuthentication : ServiceAuthenticationManager
{
    /// <summary>
    /// Store the custom authorization principle in the
    /// message properties so its available to query intercepters.
    /// </summary>
    /// <param name="authPolicy">The authorization policy collection.</param>
    /// <param name="listenUri">The uri.</param>
    /// <param name="message">The message received.</param>
    /// <returns></returns>
    public override
System.Collections.ObjectModel.ReadOnlyCollection<System.IdentityModel.Policy.IAuthorizationPolicy>
Authenticate(
System.Collections.ObjectModel.ReadOnlyCollection<System.IdentityModel.Policy.IAuthorizationPolicy>
authPolicy,
Uri listenUri, ref Message message)
    {
        //Store the custom principle in the message properties so its available later to query
interceptors.
        OperationContext.Current.IncomingMessageProperties.Add("Principal",
System.Threading.Thread.CurrentPrincipal);

        //Return the policy.
        return authPolicy;
    }
}

```

```

// Provide the ProjectInstaller class which allows
// the service to be installed by the Installutil.exe tool
[RunInstaller(true)]
public class ProjectInstaller : Installer
{
    private ServiceProcessInstaller process;
    private ServiceInstaller service;

    public ProjectInstaller()
    {
        process = new ServiceProcessInstaller();
        process.Account = ServiceAccount.LocalSystem;
        service = new ServiceInstaller();
        service.ServiceName = "WcfWindowsService";

        Installers.Add(process);
        Installers.Add(service);
    }
}

```

Note there is some new authentication hooks required when using the **WebServiceHost**. The code is still leveraging the **BasicAuthenticationProvider** class to validate the user and build the custom principle however this principal is now being retrieved in the **Validate** method of the WebServiceHost **UserNamePasswordValidator**. Also its now calling the **TryGetPrincipal** method directly as opposed to the **Authenticate** method. This is because the **Authenticate** method of the **BasicAuthenticationProvider** involves **HttpContext** and when using the webservice host there is no **HttpContext**. Additionally because there is no **HttpContext** the security principal can't be stored in the **HttpContext.Current.User** property so I'm creating a new custom message property called **Principal** to store this object instead.

6. Finally any query or change interceptors need to be modified to access the principal from the appropriate context depending on the environment.

Hide Shrink ▲ Copy Code

```

/// <summary>
/// Intercept entity query.
/// </summary>
/// <returns>Filtered recordset.</returns>
[QueryInterceptor("Groups")]
public Expression<Func<Group, bool>> OnQueryGroups()
{
    //Unbox the security principle object.
    //f there is no httpcontext then we are not operating in a web enviroment.
    //In that case we will asume authorization is being handled by the service host.
    var u = HttpContext.Current != null ?
        HttpContext.Current.User :

    (System.Security.Principal.IPrincipal)OperationContext.Current.IncomingMessageProperties["Principal"];

    //If this is a group user.
    if (u.IsInRole("GroupUser"))
    {
        //Filter for the specific group id.
        return (Group e) => e.GroupID == u.Identity.Name;
    }
    //If this is a local user.
    else if (u.IsInRole("LocalUser"))
    {
        //Filter for the group containing their site id.
        return (Group e) => e.Sites.Any(r => r.SiteID == u.Identity.Name);
    }
    else
    {
        //Return all.
        return (Group e) => true;
    }
}

```



```
}  
}
```

[Hide](#) [Copy Code](#)

```
[ChangeInterceptor("Groups")]  
public void OnChangeGroups(Group group, UpdateOperations operations)  
{  
    //Unbox the security principle object.  
    //f there is no httpcontext then we are not operating in a web enviroment.  
    //In that case we will asume authorization is being handled by the service host.  
    var u = HttpContext.Current != null ?  
        (BasicAuthenticationProvider.CustomPrincipal)HttpContext.Current.User :  
  
    (BasicAuthenticationProvider.CustomPrincipal)OperationContext.Current.IncomingMessageProperties["Principal"]  
};  
  
    if (!u.HasPermission("LightSwitchApplication:CanAddOrEditGroups"))  
    {  
        throw new DataServiceException(400, "You do not have permission to add or edit new groups.");  
    }  
}
```

## History

- 2016-03-25: Initial upload
- BasicAuthenticationProvider HttpContext
- 2018-05-09: OAuth Authentication

## License

This article, along with any associated source code and files, is licensed under [The Code Project Open License \(CPOl\)](#)

## Share

[TWITTER](#)[FACEBOOK](#)

## About the Author



**NobsterTheLobster**

No Biography provided

l  
r  
e  
l  
a  
n  
d

[Follow  
this Member](#)

## You may also be interested in...

[Angular 5 and .NET Core 2 with Visual Studio 2017](#)

[C# Linq GroupBy ExtensionMethod Explained](#)

[Debugging with Chrome in Visual Studio 2017](#)

[Implementing Custom XAML Intellisense VS2017 Extension](#)

[Random Number Generator Recommendations for Applications](#)

[Visual Studio Collection Visualizers](#)

## Comments and Discussions

Add a Comment or Question



Search Comments



First Prev Next

### There is no Template for WCF Data Service

Ahmad El Kerdi 11-May-18 6:23

I am using Visual Studio 2017, and I cannot add an item of type WCF Data Service, I don't have that item ! I followed your instructions and Created a new WCF project but after that I stuck.

[Reply](#) · [Email](#) · [View Thread](#)



### Re: There is no Template for WCF Data Service

NobsterTheLobster 15-May-18 13:18

Sincere apologies. You are correct. I wrote that part of the article when I was testing under vs 2015.

There is an article about the issue here [WCF data service template missing in VS2017 - Developer Community](#)^]

I will update the article as soon as possible with workaround or at the very least redirect to the link above.

[Reply](#) · [Email](#) · [View Thread](#)



### Failed to use Odata + EF6

ARIA 5 9-Jul-16 3:00

Can you please see [this](#)^] and [this](#)^] ?



[Reply](#) · [Email](#) · [View Thread](#)



[Refresh](#)

1

[General](#) [News](#) [Suggestion](#) [Question](#) [Bug](#) [Answer](#) [Joke](#) [Praise](#) [Rant](#) [Admin](#)

[Permalink](#) | [Advertise](#) | [Privacy](#) | [Terms of Use](#) | [Mobile](#)  
Web04-2016 | 2.8.180515.1 | Last Updated 16 May 2018

  ▼

Layout: [fixed](#) | [fluid](#)

Article Copyright 2016 by NobsterTheLobster  
Everything else Copyright © [CodeProject](#), 1999-2018