**Contents**



The **Windows Store Developer's Tour** provides an introduction to developing a Windows Store app with DevForce 2012.

What you'll learn:

- How to write n-tier LINQ queries in DevForce
- How to handle asynchronous queries and saves
- Simple navigation techniques in Windows Store apps
- Abstracting data management into a DataService

This version of the tour uses a **Code First** model. See **here** for the Database First version.

- **Platform:** Windows Store
- **Language:** C#
- **Download:** Windows Store Dev Tour (Code First)
- **Prerequisites:** Windows 8 RTM or above

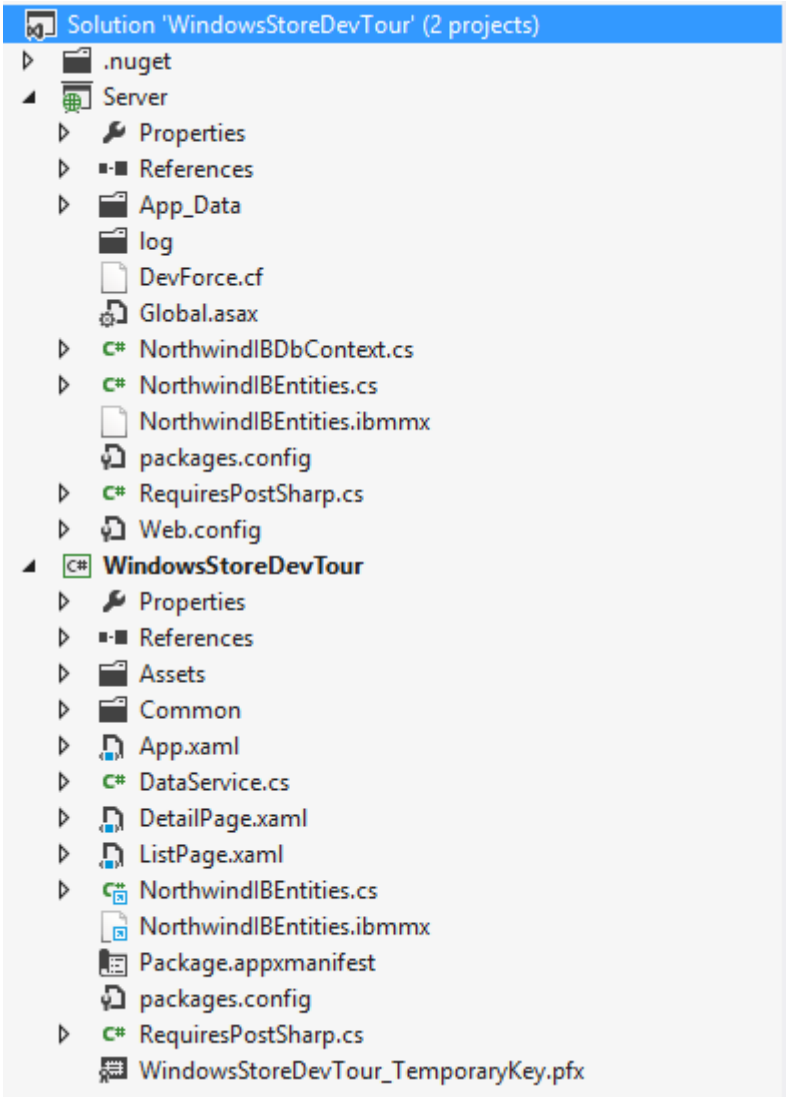
# Getting started

The Windows Store Developer's Tour demonstrates a simple two page master/detail application. The first page lists all customers in the NorthwindIB sample database and provides search capability. Tap a customer and it takes you to the detail page where you can edit the customer and save. Tapping the back button takes you back to the list.

You must enable NuGet package restore within Visual Studio in order to restore the required **DevForce NuGet packages** and other dependencies. The first time you build the application, the dependent NuGet packages are installed.

The Developer's tour includes a SQL CE version of the *NorthwindIB* sample database.

Let's first take a look at the solution structure:




A DevForce Windows Store application is an n-tier application, and uses an EntityServer to perform all backend data access. We see that reflected in the solution structure: *WindowsStoreDevTour* is the client application project, and *Server* is the web application project hosting the *EntityServer*.

Next, let's look at the model.

We've added the DevForce Code First NuGet package to both projects. This package adds the necessary DevForce and PostSharp dependencies, and is always required in a model project, *Server* here; and in the "linked" client project, *WindowsStoreDevTour* here.

File *NorthwindIBEntities.cs* contains our model: here a single entity *Customer*, and a custom *EntityManager*.

```
[ProvideEntityAspect]
[DataContract(IsReference = true)]
public abstract class BaseEntity {

...
}

/// <summary>
/// The domain-specific EntityManager for your domain model.
/// </summary>
public partial class NorthwindIBEntities : EntityManager {

  public EntityQuery<Customer> Customers { get; set; }
}

[DataContract(IsReference = true)]
public partial class Customer : BaseEntity {

...
}
```

See Apply DevForce AOP attributes, Add a custom EntityManager, and Add a custom DbContext for more information on writing a Code First model.

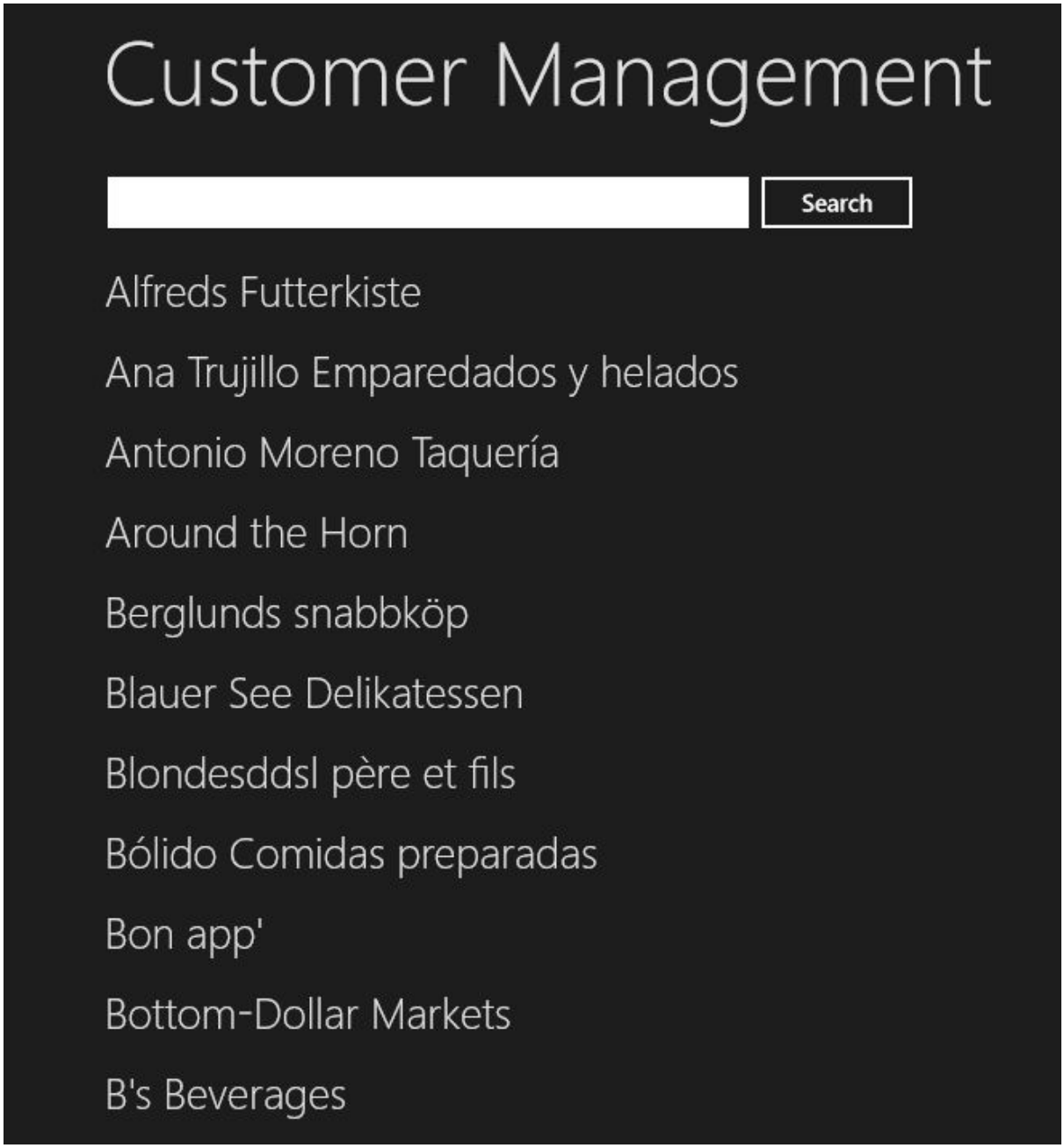
Our custom *EntityManager* and *Customer* classes in file *NorthwindIBEntities.cs* are also required in the Windows Store project, so the file has been added as a link. When we build the solution, a metadata file, here *NorthwindIBEntities.ibmmx* is added to the *Server* project, and also added as a link to the *WindowsStoreDevTour* project. In both cases, the file is given a build action of "Embedded Resource". See Generate metadata for more information on how to work with *ibmmx* files in DevForce.

## Overview

1. In order to query data from the server, you need to specify the URL of the *EntityServer*. You can do this either with an embedded app.config file, or programmatically, as show in the application constructor:

```
public App()
{
  this.InitializeComponent();
  this.Suspending += OnSuspending;
   IdeaBladeConfig.Instance.ObjectServer.RemoteBaseUrl = "http://localhost";
   IdeaBladeConfig.Instance.ObjectServer.ServerPort = 57209;
   IdeaBladeConfig.Instance.ObjectServer.ServiceName = "EntityService.svc";
}
```

2. ListPage.xaml is the master/search page. It displays all customers and provides search capability by customer name.




Looking at the code, notice the Start method, called from the *OnNavigatedTo* handler:

```
public async void Start()
{
  try
  {
    IsBusy = true;
    var customers = await DataService.Instance.GetAllCustomersAsync();
    Customers = new ObservableCollection<Customer>(customers);
  }
  finally
  {
    IsBusy = false;
  }
}
```

A few things to note here: data retrieval is asynchronous using the *async/await* keywords, and data management activities are performed by a *DataService* class.

The *IsBusy* flag is used, via data binding, to display a busy indicator:

```
ProgressRing Grid.Row="2" IsActive="{Binding IsBusy}"/>
```

Let's also look at the *search* button event handler:

```
private async void Search(object sender, RoutedEventArgs e)
{
  try
   {
      IsBusy = true;
     if (string.IsNullOrEmpty(SearchText))
      {
          Start();
        return;
      }
      var customers = await DataService.Instance.FindCustomersAsync(SearchText);
      Customers = new ObservableCollection<Customer>(customers);
   }
  finally
   {
      IsBusy = false;
   }
}
```

Again, we see asynchronous data retrieval using the *DataService* which encapsulates the DevForce LINQ queries.

3. When a customer is selected, a detail page is displayed. *DetailPage.xaml* shows customer details, and allows editing with save and undo capabilities.


Navigation to the detail page uses *Frame.Navigate*:

```
private void NavigateToDetailPage()
{
  Frame.Navigate(typeof (DetailPage), SelectedCustomer.CustomerID);
}
```

Looking at the code for DetailPage.xaml.cs, we see the *Start* method is called from the *OnNavigatedTo* handler, which receives the ID of the customer to be displayed in the *NavigationEventArgs*.

```
public async void Start(Guid customerId)
{
    Customer = await DataService.Instance.GetCustomerAsync(customerId);
}
```

And here's the *Save* button handler:

```
private async void Save(object sender, RoutedEventArgs e)
```

```
{
  await DataService.Instance.SaveAsync();
}
```

4. The *DataService.cs* contains a singleton data service used throughout the application. Both the *ListPage* and *DetailPage* work directly with the *DataService*, which encapsulates the DevForce EntityManager and performs all queries and saves.

Here's a sample method, this one to retrieve all customers:

```
public Task<IEnumerable<Customer>> GetAllCustomersAsync()
{
    return _entityManager.Customers.OrderBy(x => x.CompanyName).ExecuteAsync();
}
```

The *Task* returned is awaited upon by the caller, the *ListPage Start* method we saw above.

## Additional resources

- Windows Store Dev Center