Contents

The **Windows Universal app tour** provides an introduction to developing a Windows 10 Universal platform app with DevForce 2012.

What you'll learn:

- How to write n-tier LINQ queries in DevForce
- How to handle asynchronous queries and saves
- Abstracting data management into a DataService
- **Platform:** Windows 10 Universal
- **Language:** C#
- **Download:** Windows Universal Dev Tour
- **Prerequisites:** Windows 10

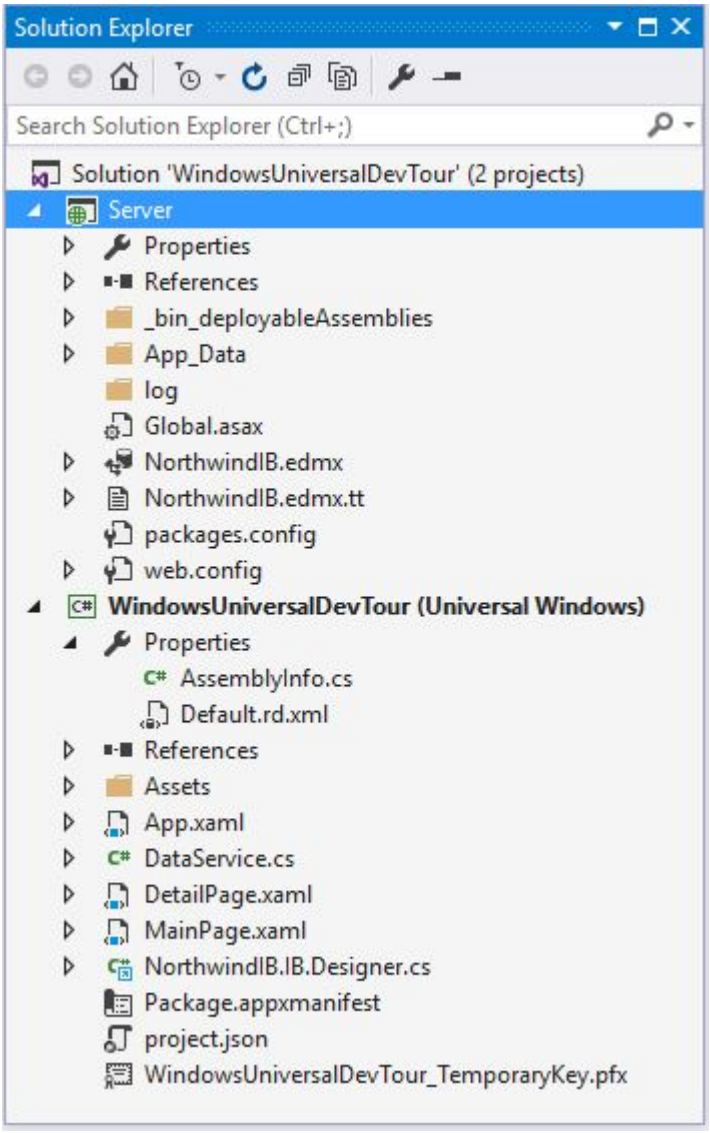
Getting started

The Windows Universal 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.

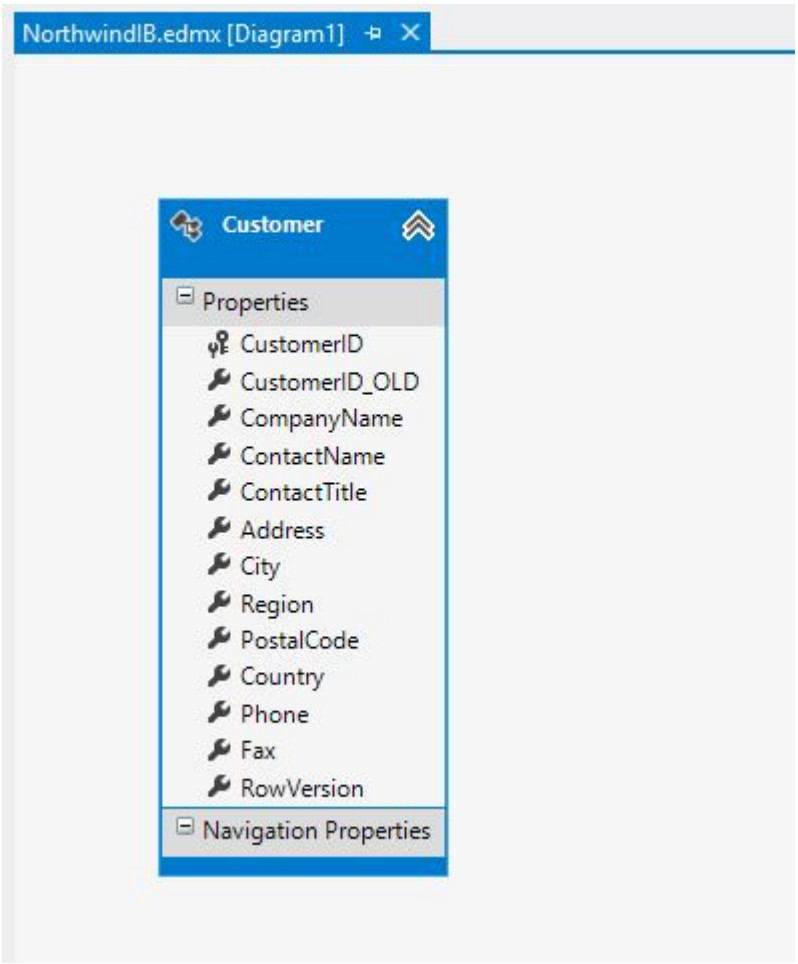
The Developer's tour includes a SQL CE version of the *NorthwindIB* sample database.

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

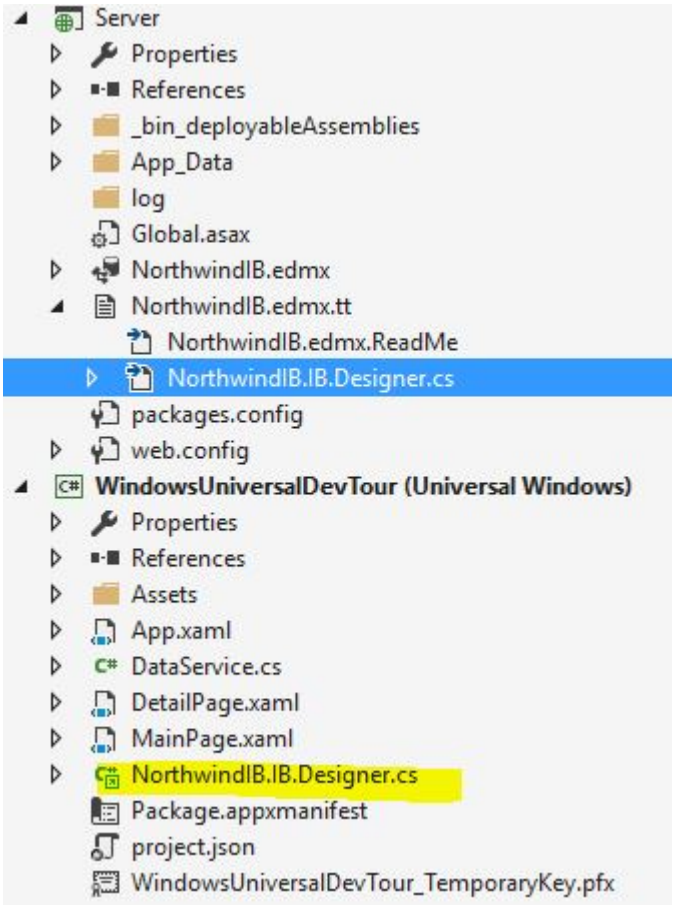


A DevForce Universal application is an n-tier application, and uses an EntityServer to perform all backend data access. We see that reflected in the solution structure: *WindowsUniversalDevTour* is the client application project, and *Server* is the web application project hosting the *EntityServer*.

Next, let's look at the model. The Entity Data Model was added to the *Server* project, and contains only a single entity type, *Customer*:



Finally, note that while the entity model is located in the *Server* project, the generated code for the entities, in the *NorthwindIB.IB.Designer.cs* file, is linked in the Windows Universal project so that these entities (and any business logic you add to the entities) can execute on the client:



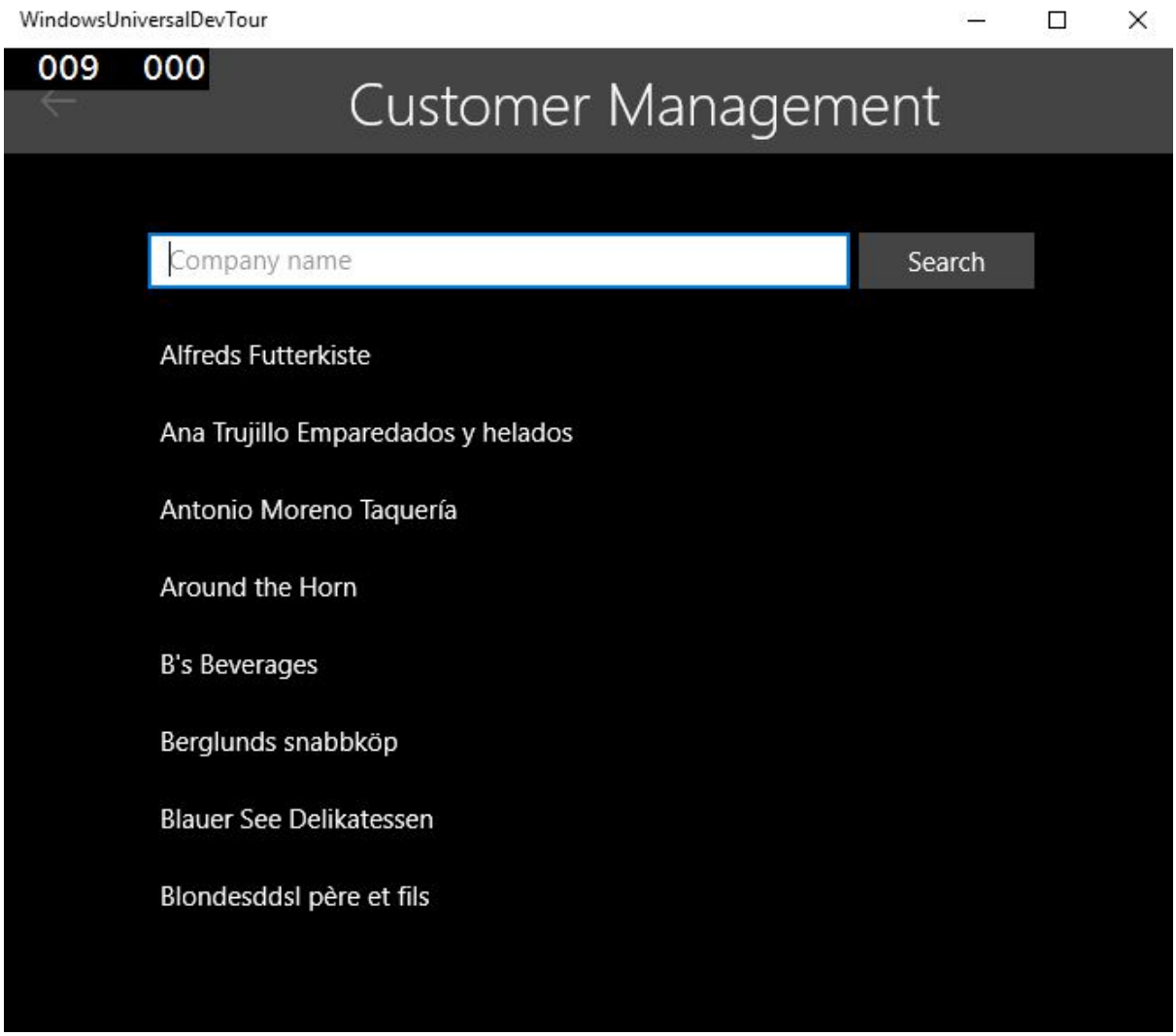
You can easily extend these entities by declaring a partial class for any of them and linking that file to the client too. These techniques are common to DevForce client development.

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 app.config file, or programmatically, as shown in the application constructor:

```
public App() {
  this.InitializeComponent();
  this.Suspending += OnSuspending;
   IdeaBladeConfig.Instance.ObjectServer.RemoteBaseUrl = "http://localhost";
   IdeaBladeConfig.Instance.ObjectServer.ServerPort = 49541;
   IdeaBladeConfig.Instance.ObjectServer.ServiceName = "EntityService.svc";
 }
```

2. MainPage.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}" Foreground="White" Width="64" Height="64"/>
```

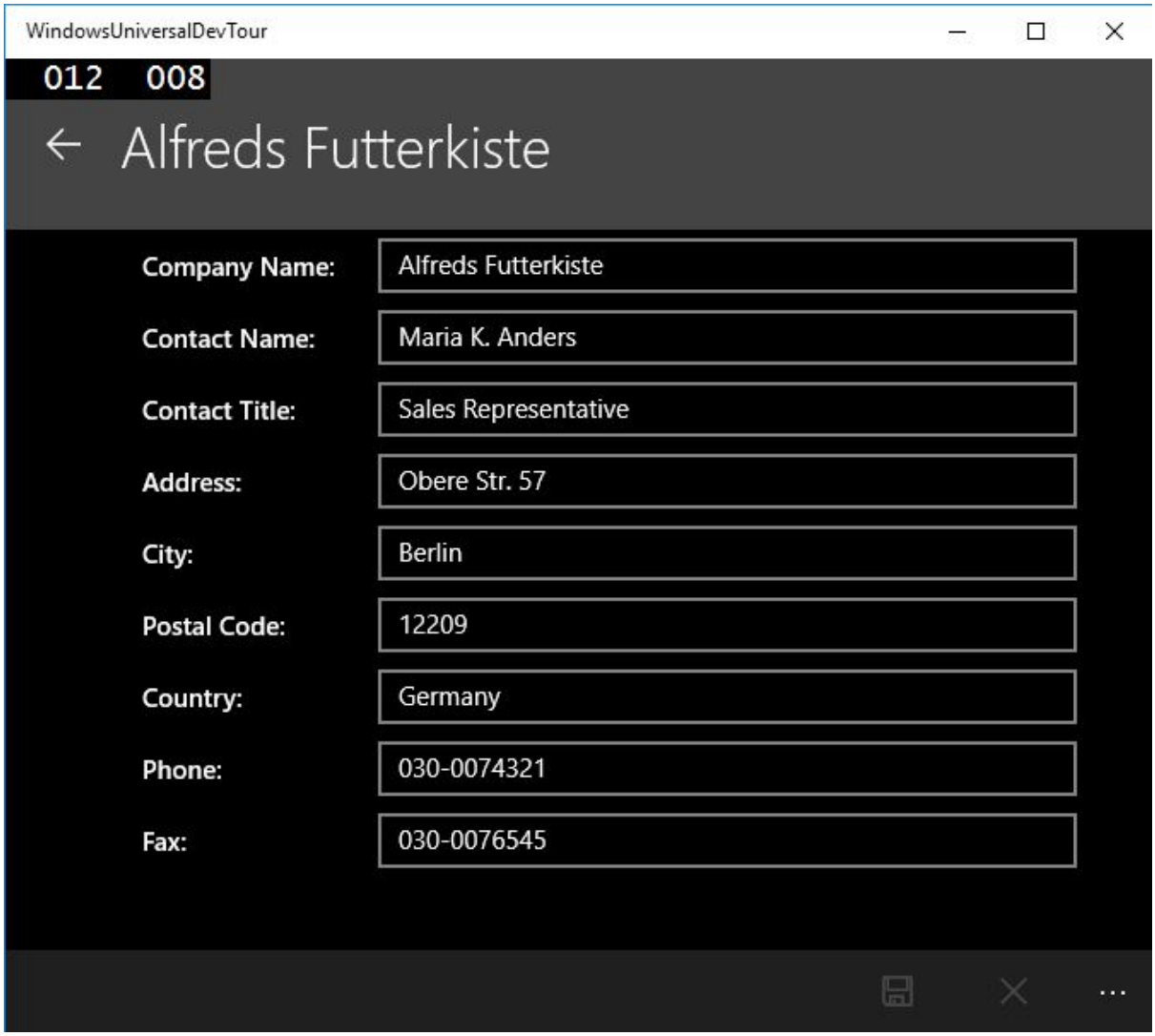
Let's also look at the *search* logic:

```
private async void DoSearch()
{
  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 *MainPage* 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 *MainPage Start* method we saw above.

Release builds

When building a Universal Windows Platform app in *release* mode the .NET Native tool chain is used to compile your app to native code for the target platforms specified. This build removes most dependencies on external runtimes and libraries and heavily optimizes code for maximum performance. To ensure your DevForce app still works in .NET native you must use runtime directives to specify its serialization and reflection requirements.

Here is the runtime directive file, Default.rd.xml, for the WindowsUniversalDevTour:

```
<Directives xmlns="http://schemas.microsoft.com/netfx/2013/01/metadata">
 <Application>
   <Assembly Name="*Application*" Dynamic="Required All" />

   <Type Name="System.Collections.Generic.List{WindowsUniversalDevTour.Customer}" DataContractSerializer="Required Public"/>
   <Type Name="IdeaBlade.EntityModel.RelatedEntityList{WindowsUniversalDevTour.Customer}" DataContractSerializer="Required Public"/>
   <Type Name="IdeaBlade.EntityModel.EntityQueryProxy{WindowsUniversalDevTour.Customer}" DataContractSerializer="Required Public"/>

 </Application>
</Directives>
```

Note that for **every** entity type in your model you must include a *Type* directive for **List<TEntity>**, **RelatedEntityList<TEntity>** and **EntityQueryProxy<TEntity>**, as we see above for the *Customer* entity.

Additional resources

- Build UWP apps with Visual Studio
- Getting Started with .NET Native
- How-to Guides for UWP apps
- Runtime Exceptions in .NET Native apps
- Runtime Directives Reference