Documentation - Windows 10 Universal Apps

Contents

You can build Windows 10 Universal apps beginning with DevForce 7.4.0. We'll show you how to get started here.

First, some nomenclature. We generally use "UWP" (for Universal Windows Platform) to indicate a Windows 10 Universal app. You'll also sometimes see the abbreviation "UAP" (Universal App Platform) in other documentation.

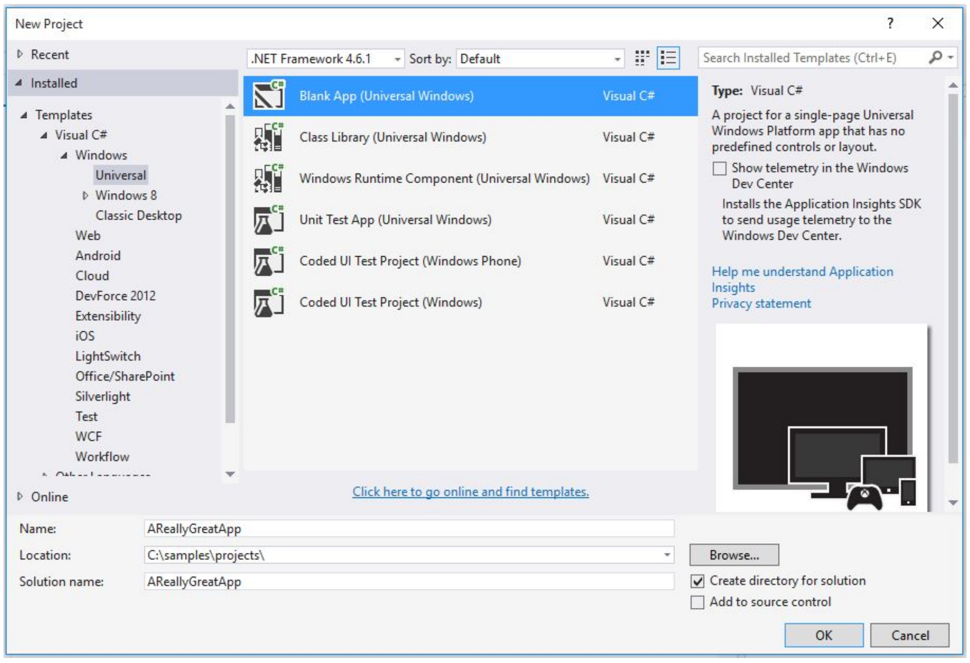
Requirements

- Windows 10
- Visual Studio 2015 Update 1 with Visual Studio Tools for Universal Windows Apps - Although UWP apps can be developed with any edition of Visual Studio 2015, DevForce 2012 requires the Professional edition or above.
- Windows 10 SDK - This will be installed with VS 2015 Update 1.
- To build a UWP app with DevForce you'll need either a "Universal" or "Windows Store" license.

Getting started

DevForce does not offer a project template to begin development of a DevForce UWP app, but it's simple to get started.

1. Create a new project and select the template for Windows Universal Blank App:



New Project

Type: Visual C#

A project for a single-page Universal Windows Platform app that has no predefined controls or layout.

Blank App (Universal Windows) — Visual C#

Name: AReallyGreatApp

Location: C:\samples\projects\

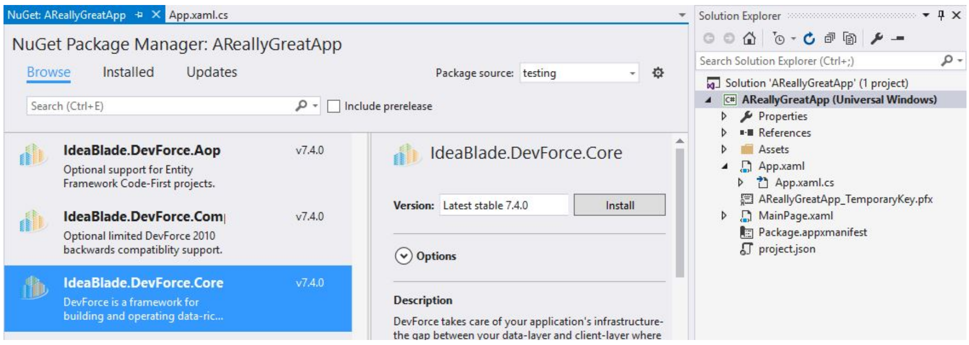
Solution name: AReallyGreatApp

☑ Create directory for solution

☐ Add to source control

Recent
Installed
Templates
Visual C#
Windows
Universal
Windows 8
Classic Desktop
Web
Android
Cloud
DevForce 2012
Extensibility
iOS
LightSwitch
Office/SharePoint
Silverlight
Test
WCF
Workflow
Online
.NET Framework 4.6.1
Sort by: Default
Search Installed Templates (Ctrl+E)
Class Library (Universal Windows) Visual C#
Windows Runtime Component (Universal Windows) Visual C#
Unit Test App (Universal Windows) Visual C#
Coded UI Test Project (Windows Phone) Visual C#
Coded UI Test Project (Windows) Visual C#
Show telemetry in the Windows Dev Center
Installs the Application Insights SDK to send usage telemetry to the Windows Dev Center.
Help me understand Application Insights
Privacy statement
Click here to go online and find templates.
Browse...
OK
Cancel

2. Install the IdeaBlade.DevForce.Core NuGet package:



NuGet Package Manager: AReallyGreatApp

Package source: testing

IdeaBlade.DevForce.Aop v7.4.0 — Optional support for Entity Framework Code-First projects.

IdeaBlade.DevForce.Com v7.4.0 — Optional limited DevForce 2010 backwards compatiblity support.

IdeaBlade.DevForce.Core v7.4.0 — DevForce is a framework for building and operating data-ric...

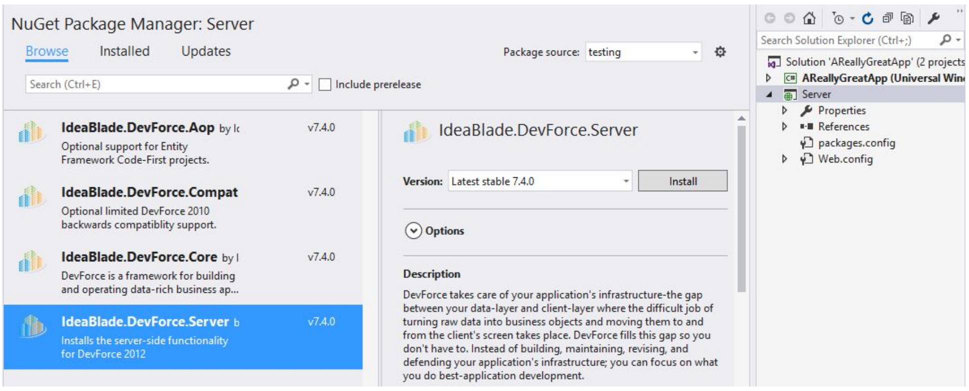
IdeaBlade.DevForce.Core

Version: Latest stable 7.4.0

Description

DevForce takes care of your application's infrastructure- the gap between your data-layer and client-layer where

3. Like other DevForce mobile environments, DevForce UWP apps are n-tier and require an *EntityServer*. The most common way to implement this is to add an empty web application project to your solution, and install the IdeaBlade.DevForce.Server package to it.



NuGet Package Manager: Server

Browse | Installed | Updates

Package source: testing

Search (Ctrl+E) | Include prerelease

IdeaBlade.DevForce.Aop by I — v7.4.0
Optional support for Entity Framework Code-First projects.

IdeaBlade.DevForce.Compat — v7.4.0
Optional limited DevForce 2010 backwards compatiblity support.

IdeaBlade.DevForce.Core by I — v7.4.0
DevForce is a framework for building and operating data-rich business ap...

IdeaBlade.DevForce.Server b — v7.4.0
Installs the server-side functionality for DevForce 2012

IdeaBlade.DevForce.Server

Version: Latest stable 7.4.0 | Install

Options

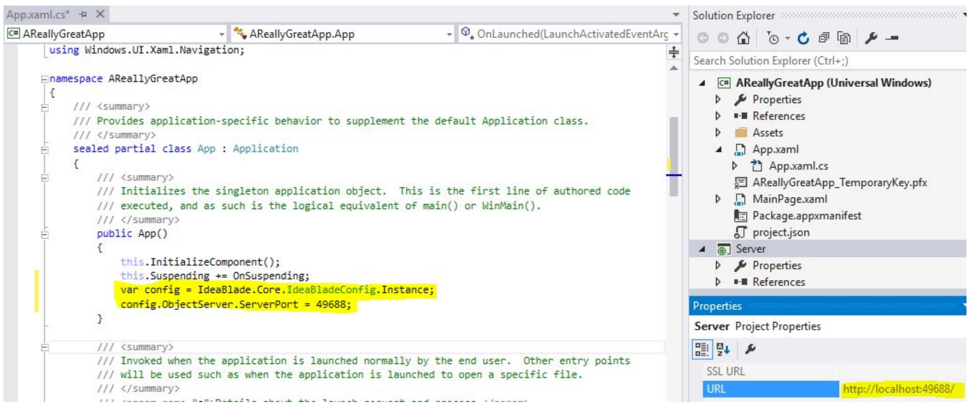
Description

DevForce takes care of your application's infrastructure-the gap between your data-layer and client-layer where the difficult job of turning raw data into business objects and moving them to and from the client's screen takes place. DevForce fills this gap so you don't have to. Instead of building, maintaining, revising, and defending your application's infrastructure; you can focus on what you do best-application development.

Search Solution Explorer (Ctrl+;)

```
Solution 'AReallyGreatApp' (2 projects
▷ AReallyGreatApp (Universal Win
◢ Server
    ▷ Properties
    ▷ References
      packages.config
    ▷ Web.config
```

4. Configure the app to communicate with the *EntityServer*.
You can use either an app.config or simple programmatic configuration. During development, it's easy to get started by specifying only the server port, allowing other settings to default, like we do here in the app startup logic:



App.xaml.cs* | AReallyGreatApp | AReallyGreatApp.App | OnLaunched(LaunchActivatedEventArg

```
using Windows.UI.Xaml.Navigation;

namespace AReallyGreatApp
{
    /// <summary>
    /// Provides application-specific behavior to supplement the default Application class.
    /// </summary>
    sealed partial class App : Application
    {
        /// <summary>
        /// Initializes the singleton application object.  This is the first line of authored code
        /// executed, and as such is the logical equivalent of main() or WinMain().
        /// </summary>
        public App()
        {
            this.InitializeComponent();
            this.Suspending += OnSuspending;
            var config = IdeaBlade.Core.IdeaBladeConfig.Instance;
            config.ObjectServer.ServerPort = 49688;
        }

        /// <summary>
        /// Invoked when the application is launched normally by the end user.  Other entry points
        /// will be used such as when the application is launched to open a specific file.
        /// </summary>
```

Solution Explorer

Search Solution Explorer (Ctrl+;)

```
◢ AReallyGreatApp (Universal Windows)
    ▷ Properties
    ▷ References
    ▷ Assets
    ◢ App.xaml
        ▷ App.xaml.cs
      AReallyGreatApp_TemporaryKey.pfx
    ▷ MainPage.xaml
      Package.appxmanifest
      project.json
◢ Server
    ▷ Properties
    ▷ References
```

Properties

Server Project Properties

SSL URL	
URL	http://localhost:49688/

(Hmm, Someone doesn't know how to use a highlighter.)

5. See the development guide for what you can do next.

Queries

- DevForce uses QueryStrategy.Normal by default when querying. This translates into an Optimized fetch strategy, which generally means that DevForce will determine how best to satisfy the query using local cache, the data source, or a combination of the two. In UWP apps, at this time queries may not be executed against both cache and the data source, which means that a *FetchStrategy* of *DataSourceThenCache* or *DataSourceAndCache* is not supported.
  In UWP apps, ***QueryStrategy.Normal*** means that DevForce will use the ***DataSourceOnly*** fetch strategy when the *EntityManager* is connected and the ***CacheOnly*** fetch strategy when disconnected. You can still use custom query strategies, but an exception will be thrown if you use an unsupported fetch strategy.
- There is reduced support for the *Nullable<T>* type in UWP apps. Most importantly, the List<Nullable<T>> cannot be used in queries. For example, you might want to use a List<int?> when using a query with a contains clause:

  ```
  var pids = new List<int?> {1, 2, 3};
  var query = _entityManager.Products
            .Where(p => pids.Contains(p.ID));
  ```

  This will fail, however, because the List<int?> cannot be serialized. To work around this restriction, use a List for the non-nullable value type, in this case List<int>.

  Note that you will only encounter an error when doing a *release* build or enabling the .NET Native tool chain. You may also encounter other edge conditions when using Nullable<T> in release builds. **It's very important to enable the .NET Native tool chain at times during development and testing to ensure there will be no unpleasant surprises at release.**

Release build

When building a UWP 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.**

Specifically, for every entity type in your model you must include a Type directive for List<TEntity>, RelatedEntityList<TEntity> and EntityQueryProxy<TEntity>. This is required even if you never query or save the entity.

For example, here's the runtime directive file, Default.rd.xml, for an app whose entity data model contains an entity type named "Customer":

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

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

  </Application>
</Directives>
```

Other known issues

- Code First models are not currently supported.
- Projection into an anonymous type is not supported. See the return part of an entity topic for how to work around this.

Additional resources

- DevForce Windows Universal app tour
- Get Started with Windows Apps
- 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