ALLAN THRAEN | 3 years ago | PDF |

Tips and Tricks Azure Addon Development Optimizely (Episerver)

HOW TO SETUP DEV & BUILD ENVIRONMENT FOR AN EPISERVER ADDON

Over the years I have been involved in quite a lot(!) of Episerver addons and integration projects. A key to a successful add-on is to get the entire project and environment correctly setup and working from the start. This is my recipe.

Introduction

Back in early october I was lucky enough to be invited to speak at the virtual Episerver community week, along with Episervers Nicholas Rohr and fellow EMVP, the amazing Marija Jemuović Delibašić about one of my favourite topics: Add-ons for Episerver and the Episerver Marketplace.

The recording can be seen here: https://episerver.wistia.com/medias/slboh4mo2p

In preparation for that I cleaned up my usual checklist for when I create a new Add-on project which I figured I'd share in detail here.

Note that Marija's approach is slightly different than mine, but both works well. Like Marija, I also plan to share a public repository soon with a starter kit for add-ons.

Prequel: Testing out the idea

To be completely honest, this is not really a required step - it is just what I usually end up doing. When I get a new idea for an add-on I usually get so excited about it that I just want to test it out really quickly - without too much setup. So it often starts out with a quick mock-up in an Alloy site, to see that the entire concept is valid and holds water, before I go through the following 'proper' steps.

Create the Repository

First things first, let's start by create the repository. I usually do this directly in Github, configure license, pick the "VisualStudio" gitignore, and add a readme to the initial repository. Then clone it locally and we're all set to start coding.

Creating the Project and Primary Solution

Fire up good ol' Visual Studio and create a new Class Library project (.NET Framework, C#) in a "src" folder inside your newly created repository.

I use the "src" folder as I might later add Docs folder, Examples folder or similar.

This is also a good time to ensure your have set the proper default namespace and assembly name in your project, and that you are targeting the proper .NET Framework version. Note that Episerver (up to CMS 11) supports .NET Framework 4.61 and later. If you can, do the same.

Now, add the Episerver nuget packages to your solution. Some packages I typically add are:

- EPiServer.CMS.Core
- EPiServer.CMS.AspNet
- EPiServer.CMS.UI (if you include UI extensions)
- EPiServer.ServiceLocation.StructureMap
- Others that your add-on might need - like Forms, if your addon extends forms, EPiServer.ApplicationModules if you have custom VisitorCriteria and so on.

I typically get started building with the latest packages, but then later - before I release I try to roll back to earlier versions of the Episerver packages to support as early an Episerver version as possible for my add-on.

Initial Project modifications

Remove the default 'Class1.cs".

Add a folder called "/modules/_protected/[name of your addon]"

In that folder add a module.config and a web.config.

If your add-on contains views, you can create a "Views" folder inside your module folder.

If your add-on contains dojo scripts, add a "ClientResources" folder inside your module folder.

Update module.config to register scripts, styles, assemblies or similar.

Adding your code

Now, you can start to add your code. Put cs files whereever you think is a good place, but make sure that views and other assets that needs to be installed on the site goes in the Module folder you created before.

You can create an XML file with **translation texts** if needed. Make sure to mark it as an Embedded

Resource in it's Visual studio properties.

You should probably also add a **web.config.transform** file. As a minimum this should register your module - but it's also a good idea to add app-settings or your own configuration section if needed, as well as any provider registrations needed.

Preparing the project for build

Add a **readme.txt** file that can be shown to Developers after they install your add-on with whatever instructions they need.

Add a **Nuget.Config** file to the root of the project. This is needed for Azure Devops to know how to access the Episerver nuget feed when it restores packages in order to build. Here is the nuget.config I usually use:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <packageRestore>
    <!-- Allow NuGet to download missing packages -->
    <add key="enabled" value="True" />

    <!-- Automatically check for missing packages during build in Visual Studio -->
    <add key="automatic" value="True" />
  </packageRestore>
  <packageSources>
    <add key="NuGet official package source" value="https://api.nuget.org/v3/index.json" />
    <add key="EPiServer Feed" value="https://nuget.episerver.com/feed/packages.svc/" />
  </packageSources>
</configuration>
```

Add a **[Project name].nuspec** file in order to define the elements needed for your nuget file to be created. Make sure to setup proper dependencies (Episerver packages at least) as well as copying the addon module folder into content.

Here is a nuspec I have used. It should be updated a bit though:

```
<?xml version="1.0"?>
<package xmlns="http://schemas.microsoft.com/packaging/2010/07/nuspec.xsd">
  <metadata>
    <id>EpiserverAddonStarterKit</id>
    <version>$version$</version>
    <title>Friendly Title</title>
    <authors>Your name</authors>
    <owners>Your companys name</owners>
    <requireLicenseAcceptance>false</requireLicenseAcceptance>
    <iconUrl>https://www.codeart.dk/favicon.ico</iconUrl>
    <projectUrl>https://www.codeart.dk/Jupiter</projectUrl>
    <releaseNotes>
    </releaseNotes>
    <description></description>
    <tags>EPiServerPublicModulePackage</tags>
    <dependencies>
      <dependency id="EPiServer.CMS.AspNet" version="[11.17.0,12.0.0)" />
      <dependency id="EPiServer.Framework" version="[11.17.0,12.0.0)" />
      <dependency id="EPiServer.CMS.UI.Core" version="[11.27.0,12.0.0)"/>
      <dependency id="Newtonsoft.Json" version="11.0.0"/>
    </dependencies>
  </metadata>
  <files>
    <file src="web.config.transform" target="content\" />
    <file src="modules\_protected\EpiserverAddonStarterKit\**\*" target="content\modules\_protected\Ep
  </files>
</package>
```

If everything builds correctly so far, commit and push the code.

Adding a sandbox site

This is one of the places where Marijas approach differs from mine. I terribly lazy and prefer to be able to just hit run in VS and step through all the code - from the site to the addon.

Typically I do this by creating a new solution with an out-of-the-box episerver Alloy site (unless it's commerce, then Quicksilver) somewhere else on disk - so not below the add-on repository.

Once created I make sure to upgrade it to the **latest Episerver** and check that it runs as expected.

Now, in Visual studio **add existing project to solution**. Navigate and find the add-on project and add it to this solution as well as it's own.

When that is done, **add a reference** from the site to the add-on.

The reference will take care of getting the assembly over, but we also need to move the Module folder. In order to do that I typically configure a build-event in my sandbox site, that simply does an xcopy. Something like:

xcopy "[path to add-on folder]\modules_protected\[Add-on name]" "$(ProjectDir)modules_protected\[Add-on name]\" /s /y

Also make sure to manually do the **web.config changes** the transform would normally take care of.

Note this this site is purely a development sandbox. It won't be useful as a test site.

Now it's all set up and you can run it and see if it works with the add-on. And if it doesn't you can debug all the way through easily.

Configuring the Nuget package to build in Azure Devops

I've grown quite fond of Azure Devops - especially for add-ons. It's so nice to have it automatically setup

so any updates to the Master branch will force a build.

We've already prepared the nuspec and nuget.config files, so we can go right ahead and create a new project in Azure Devops. you can remove "Boards and Repos" from the project if you want - we don't need them right now.

Next, define a new empty YML build pipeline, connected to your Github (or Bitbucket or whatever) repo. There is a wizard to guide you the way all the way from connecting to the repo to having a complete pipeline.

Here is an example of a pipeline I typically use:

```
name: $(majorMinorVersion).$(semanticVersion) # $(rev:r)

trigger:
- master

pool:
  vmImage: 'windows-latest'

variables:
  solution: '**/EpiserverAddonStarterKit.sln'
  buildPlatform: 'Any CPU'
  buildConfiguration: 'Release'
  majorMinorVersion: 0.1 #Change this!
  semanticVersion: $[counter(variables['majorMinorVersion'], 0)]

steps:
- task: NuGetToolInstaller@1

- task: Assembly-Info-NetFramework@2
  inputs:
    Path: '$(Build.SourcesDirectory)'
    FileNames: '**\AssemblyInfo.cs'
    InsertAttributes: false
    FileEncoding: 'auto'
    WriteBOM: false
    Configuration: '$(buildConfiguration)'
    VersionNumber: '$(majorMinorVersion).$(semanticVersion)'
    FileVersionNumber: '$(majorMinorVersion).$(semanticVersion)'
    LogLevel: 'verbose'
    FailOnWarning: false
    DisableTelemetry: false

- task: NuGetCommand@2
  inputs:
    command: 'restore'
    restoreSolution: '$(solution)'
    feedsToUse: 'config'
    nugetConfigPath: 'EpiserverAddonStarterKit/EpiserverAddonStarterKit/Nuget.config' #Update path her

- task: VSBuild@1
  inputs:
    solution: '$(solution)'
    platform: '$(buildPlatform)'
    configuration: '$(buildConfiguration)'


    # Package a project
- task: NuGetCommand@2
  inputs:
    command: 'pack'
    packagesToPack: 'EpiserverAddonStarterKit/EpiserverAddonStarterKit/EpiserverAddonStarterKit.csproj
    versioningScheme: 'byBuildNumber'

- task: PublishBuildArtifacts@1
  inputs:
    PathtoPublish: '$(Build.ArtifactStagingDirectory)'
    ArtifactName: 'drop'
    publishLocation: 'Container'

- task: NuGetCommand@2
  inputs:
    command: 'push'
    packagesToPush: '$(Build.ArtifactStagingDirectory)/**/*.nupkg;!$(Build.ArtifactStagingDirectory)/
    nuGetFeedType: 'internal'
    publishVstsFeed: 'GUIDTO-VSTS FEED' #Update this with the guid of the feed to push to
```

You should also setup a release pipeline that will copy the package built onto a local Nuget feed from that Azure repo. That way you can always test the packages easily in your local visual studio - just by adding that nuget feed.

Ensure you have properly configured versioning in a way that auto-increments and sets the assembly version as well as the nuget version. Nothing worse than having a hard time keeping track of which version a certain bug is fixed in, without being able to see the version reported.

Once the pipelines are completed, do a run and see if it works. It usually takes a couple of tries to get adjusted right.

Try out package locally

Download Foundation or some other hopelessly overcomplicated site :-), follow the instructions to get it to run - and if you success, try to install your add-on from the Devops Feed.

Now, test it properly. Be aware that if you test against Foundation there'll be a lot of other add-ons installed that could potentially conflict with yours in ways you might not imagine, so you should also check that they still work as intended.

When all this is done, it's time to move on to the last step...

Upload to Episerver nuget feed

Now, if this is a community contributed add-on, just log in to nuget.episerver.com and upload it. Within a few days Episervers staff will have a look and probably approve it and it'll be out in the official Episerver feed. You won't be notified when it's out, so it's a matter of checking regularly for it. When it's out and available a good idea is to do a blog post on it, and if it's open source, make the repository public (remember a good Readme.md before doing that). Sit back, relax and wait for the glory.

If it's a commercial add-on, you'll probably want to get it on the official marketplace. So, follow the instructions to submit to marketplace. Submit it. Wait for it to be tested. Fix the errors the testers have found and repeat. Keep up the faith, eventually you'll get it out :-)

Tips and Tricks | Azure | Addon Development | Optimizely (Episerver)

RECENT POSTS

CodeArt ApS
Teknikerbyen 5, 2830 Virum, Denmark
Email: info@codeart.dk
Phone: +45 26 13 66 96
CVR: 39680688