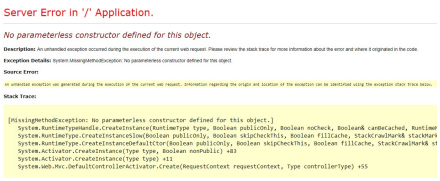
ALLAN THRAEN | 5 years ago | PDF |

Tips and Tricks .NET Development C# Optimizely (Episerver)

ERROR: NO PARAMETERLESS CONSTRUCTOR DEFINED FOR THIS OBJECT



Ever started a site from scratch rather than the reference site and run into this classic error? Here's a hint for you.

Call me oldfashioned, but when I start a new web project, I do the same as when I get a new laptop. Start from scratch with a clean platform and then add the stuff I want carefully, rather than start with a bunch of bloatware that you then have to adjust and remove.

In Episerver terms I often do this by using the Empty website template in visual studio. And it works pretty good. But once my site grows a bit and I start adding some addons or writing code more complex than just outputting the models, I often encounter this error.

At first it puzzled me a bit - I mean, Dependency Injection is native to Episerver - and usually works beautifully - so why now this?

Obviously it happens because somewhere a class expects to be able to use dependency injection through it's constructor like this:

```
public class StandardPageController : PageController<StandardPage>
{
    private IContentRepository _repo { get; set; }

    public StandardPageController(IContentRepository Repo)
                                                    {
        _repo = Repo;
    }

    public ActionResult Index(StandardPage currentPage)
                                                    {
        /* Implementation of action. You can create your own view model class that you pass to the
                * you can pass the page type for simpler templates */

        return View(currentPage);
    }
}
```

Sometimes it's a bit more hidden - in one project I noticed because the icons for the wonderful Geta project Epi.FontThumbnail (check it out!) were not showing - they used dependency injection in their controller.

This happens because Episerver CMS typically uses a different dependency injector than default in ASP.NET. Sample sites like Alloy and Quicksilver contains classes to change the default to the right one. One could wonder why something as essential as this is a bit of sample code, but my guess is that Episerver don't want to force a specific dependency injection on a poor unsuspecting developer, but instead give them the option of what to use. Anyways, since I've fallen down this trap myself a few times I figured I'd share the code from alloy here, slightly modified to take out Alloy specific stuff.

```
using System.Web.Mvc;
using EPiServer.Framework;
using EPiServer.Framework.Initialization;
using EPiServer.ServiceLocation;
using EPiServer.Web.Mvc;
using EPiServer.Web.Mvc.Html;

namespace DependencyInjection
{
  [InitializableModule]
  public class DependencyResolverInitialization : IConfigurableModule
  {
    public void ConfigureContainer(ServiceConfigurationContext context)
    {
    }

    public void Initialize(InitializationEngine context)
    {
      DependencyResolver.SetResolver(new ServiceLocatorDependencyResolver(context.Locate.Advanced));
    }

    public void Uninitialize(InitializationEngine context)
    {
    }

    public void Preload(string[] parameters)
    {
    }
  }
}
```

DependencyResolverInitialization.cs hosted with ♥ by GitHub view raw

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web.Mvc;
using EPiServer.ServiceLocation;

namespace DependencyInjection
{
  public class ServiceLocatorDependencyResolver : IDependencyResolver
  {
```

```
    readonly IServiceLocator _serviceLocator;

    public ServiceLocatorDependencyResolver(IServiceLocator serviceLocator)
    {
      _serviceLocator = serviceLocator;
    }

    public object GetService(Type serviceType)
    {
      if (serviceType.IsInterface || serviceType.IsAbstract)
      {
        return GetInterfaceService(serviceType);
      }
      return GetConcreteService(serviceType);
    }

    private object GetConcreteService(Type serviceType)
    {
      try
      {
        // Can't use TryGetInstance here because it won't create concrete types
        return _serviceLocator.GetInstance(serviceType);
      }
      catch (ActivationException)
      {
        return null;
      }
    }

    private object GetInterfaceService(Type serviceType)
    {
      object instance;
      return _serviceLocator.TryGetExistingInstance(serviceType, out instance) ? instance : null;
    }

    public IEnumerable<object> GetServices(Type serviceType)
    {
      return _serviceLocator.GetAllInstances(serviceType).Cast<object>();
    }
  }
}
```

ServiceLocatorDependencyResolver.cs hosted with ♥ by GitHub view raw

Tips and Tricks .NET Development C# Optimizely (Episerver)

RECENT POSTS

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