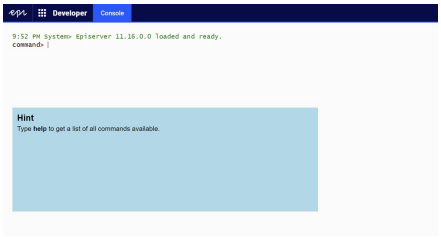
ALLAN THRAEN | 4 years ago | PDF |

Vision Demos & Prototypes C# Optimizely (Episerver) Addon Development

INTRODUCING THE DEVELOPERCONSOLE FOR EPISERVER



Often you find yourself creating Scheduled Jobs in Episerver, only intending them to be run manually - and often annoyed that they don't support parameters. This blog post introduces a new approach to solving the same problem: The Developer Console.

It's very often I find myself in the above situation. Perhaps I need to write some custom code that imports a dataset into Episerver, maybe I need to traverse the content tree and update certain content - or maybe I need to programmatically read some settings. Anyway - I often find myself falling back to the easy Scheduled Jobs.

But there are problems related to that - first of all you'll end up with a lot of scheduled jobs, that you don't really want the editors to run by accident. Secondly, you can't have parameters in any easy way - so you have to find ways to work around that constantly. Either by hardcoding values (but that's not nice in production) or by having a content item keeping the parameters. Neither are ideal solutions.

The DeveloperConsole is a concept that has been on the way on and off for a few years. I think the idea originated between Valdis and myself at a party in 2018 - and I'm pretty sure there was beers involved - which might explain some of the weirder aspects of this :-) The idea isn't completely new though. In 2011 Adam Najmanowich started a powershell console for Episerver and 10 years ago Magnus Stråle and I did a demo of running an entire Episerver in a console (https://www.codeart.dk/archive/episerver-labs/2009/8/Codemania-Run-EPiServer-CMS-in-the-Console/).

The basic idea is simple: Provide a classic command prompt to Episerver where you can execute commands with parameters much like you are used to. And create new commands as easily as creating a class.

Quickly the idea grew, and it became obvious that if piping was supported, then you could have multiple uses for the same command - passing output from one command to input for the next.

To get started just install the CodeArt.Episerver.Console package from the Episerver nuget feed and you can instantly try out the console under the Developer menu in Episervers backend.

You can also find the repository here: https://github.com/CodeArtDK/CodeArt.Episerver.Console feel free to do pull requests or suggest improvements. Let me know if you want to be part of the team building and extending this in the future.

Note: This is still an experimental release, so I would not advice using this in production yet.

Working with commands

An example of a very simple command is the "8ball" command. This is really silly, and just included to illustrate a point. It simulates the classic magic 8ball providing you with an answer to lifes big questions. The code for it is pretty straight forward:

```
using CodeArt.Episerver.DevConsole.Attributes;
using CodeArt.Episerver.DevConsole.Interfaces;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace CodeArt.Episerver.DevConsole.Commands
{
    [Command(Keyword ="8ball", Description = "Throw the magic 8-ball and get your answer")]
    public class Magic8BallCommand : IConsoleCommand
    {
        public string Execute(params string[] parameters)
                                                        {
            Random r = new Random();
            return new string[] {"Absolutely","I will think about it", "Oh, you know I'll say yes","n
        }
    }
}
```

The command must implement IConsoleCommand which specifies that it needs to have an Execute method that returns a string. The string returned will be put in the console log as output, unless it's null.

Also, the [Command] attribute specifies that this is a command and its keyword is "8ball". Let's see an example use of this:

```
command>8ball Should I upgrade my Episerver installation?
11:11 AM Magic8BallCommand> Absolutely
command>
```

Here we pass along a lot of parameters - but they don't really matter, as nothing in the code interacts with them. Instead a random answer will be returned.

Another simple command is the "HelloWorldCommand" which takes a parameter, the Name.

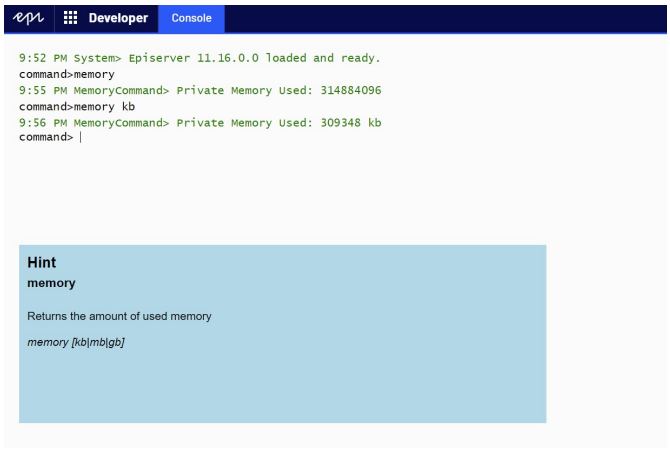
```
using CodeArt.Episerver.DevConsole.Attributes;
using CodeArt.Episerver.DevConsole.Interfaces;
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;

namespace CodeArt.Episerver.DevConsole.Commands
{
    [Command(Keyword ="hello", Description ="The classic hello world example command")]
    public class HelloWorldCommand : IConsoleCommand
    {

        [CommandParameter]
        public string Name { get; set; }

        /// <summary>
        /// Returns a string that will be returned as output.
        /// </summary>
        /// <returns></returns>
        public string Execute(params string[] parameters)
                                                        {
            if (string.IsNullOrEmpty(Name) && parameters.Length == 1) Name = parameters.First();
            return "Hello " + Name;
        }
    }
}
```

We simply specify this as a public property with the [CommandParameter] attribute attached. This way, we can call it like "hello -Name [name]" and the [name] will be assigned to the Name property before execution. Alternatively, if the user prefers to not use named parameters they can just include the name as the first parameter and we have code to handle that.

```
command>hello -name Allan
9:57 PM HelloWorldCommand> Hello Allan
```

Much the same way is the memory command:



It will output to console the amount of memory currently used by the Episerver application process - and by attaching kb|mb|gb as unnamed parameters it will show it as that. By the way, notice how the Hint box below detects the commands you are using and is automatically suggesting syntax as you type. And yes - just as in visual studio, you can use CTRL+. to use the suggested hint in your command.

As we start piping stuff, it gets a bit more fun. An example of a command that can output data in the pipe is the ListDescendentsCommand. It will list content descendents below a certain point, and pass each of them on to the next commands in the pipe.

```
using CodeArt.Episerver.DevConsole.Attributes;
using CodeArt.Episerver.DevConsole.Interfaces;
using EPiServer;
using EPiServer.Core;
using EPiServer.ServiceLocation;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace DeveloperTools.Console. Commands
{
    [Command(Keyword = "listdescendents", Description ="Lists all descendents below the node")]
    public class ListDescendentsCommand : IOutputCommand
    {
        public event CommandOutput OnCommandOutput;

        [CommandParameter]
        public string Parent { get; set; }

        private readonly IContentRepository _repo;

        public ListDescendentsCommand(IContentRepository contentRepository)
                                                                    {
            _repo = contentRepository;
        }

        public string Execute(params string[] parameters)
                                                        {
            int cnt = 0;
            ContentReference start = ContentReference.StartPage;
            if (!string.IsNullOrEmpty(Parent))
            {
                if (Parent.ToLower() == "root") start = ContentReference.RootPage;
                start = ContentReference.Parse(Parent);
            }
```

```
            foreach(var r in _repo.GetDescendents(start))
            {
                OnCommandOutput?.Invoke(this, _repo.Get<IContent>(r));
                cnt++;
            }

            return $"Done, listing {cnt} content items";
        }
    }
}
```

It can do that, since it implements IOutputCommand - and hence has an event "OnCommandOutput" that the following command can attach and listen to.

```
command>listdescendents 5|filter PageTypeName = ArticlePage|select PageName|dump
9:59 PM DumpCommand> "Alloy Saves Bears"
9:59 PM DumpCommand> "Enhances Risk Management"
9:59 PM DumpCommand> "Top Collaboration Technology"
9:59 PM DumpCommand> "Trek Selects Alloy Plan"
```

In this case I'm asking it to list all descendents below the start page (with ID 5), then a filter commands listens to the command and filters content with the pagetype ArticlePage, passes those results on to a select command which then picks the PageName and finally sends it to the Dump command that basically tries to output whatever it gets sent to the console in the best way possible.

Now, you might start to see how this potentially can be powerful.

And we don't have to limit ourselves to content. How about looking at the assemblies loaded in the app domain?

```
command>assemblies|skip 10| take -count 1|dump -type Json
10:03 PM DumpCommand> { "Data": "System.Data, Version=4.0.0.0, Culture=neutral, PublicKe
"System.Data, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" }
```

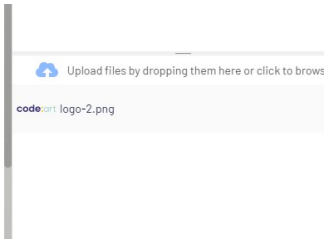
If we take this on to the more extreme, we could also imagine creating content programmatically. In this example, I'm using a command to create a piece of content, pass it on to have a few properties set, then save and publish it and pass along the content reference which I will then get the friendly url for and dump it to console. Easy peasy ;-)

```
command>createcontent -contenttypename StandardPage -parent 5|setproperty -name PageName -value "This is
cool"|setproperty -name MainBody -value "Hey there"|savecontent -action publish|getfriendlyurl|dump
11:12 AM DumpCommand> /en/this-is-cool/
```

I could also use commands to let the server download an asset and put it in my media library - to save me the trouble of downloading it locally and then uploading it. In this case I download the codeart logo in png:

```
command>import-asset -url "https://www.codeart.dk/contentassets/ec584942f4c84114aed03c3ec8161130/logo-2.png"
2:06 PM ImportAssetCommand> Successfully downloaded asset https://www.codeart.dk/contentassets/ec584942f4c84114aed03c3ec8161130/logo-2.png
command>
```

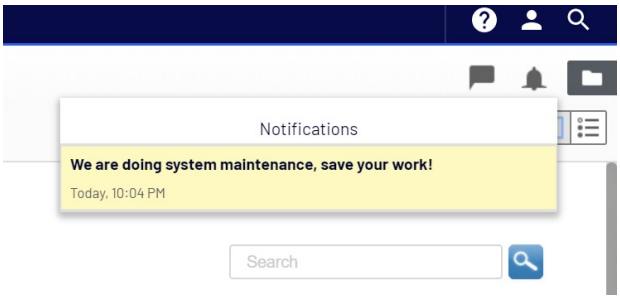
And when I check my media library I'll see this:



I could of course also do other interactions with the Episerver CMS. For instance send a notification to all editors:

```
command>notify -Subject "We are doing system maintenance, save your work!"
10:04 PM NotifyCommand> User notified
```

which would result in them seeing this notification:



Coming up...

I think this is enough introduction for now. Head over to the repository to check it out, or try it out yourself on a local Episerver installation near you.

I plan to post more about this - for example how to work with session variables, advanced piping, and async commands.

Please - if you enjoy this, leave a comment below - or even better, contribute to the project!

Vision Demos & Prototypes | C# | Optimizely (Episerver) | Addon Development

RECENT POSTS

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