ALLAN THRAEN | 14 years ago | PDF |

BUILDING A TWITTER WORKFLOW

For a long time I've wanted to play more with Windows Workflow Foundation (WWF) in conjunction with EPiServer, and today I finally did it. And luckily it turned out to be a lot less scary than I had imagined.

First of all, I decided to check out a couple of guides online. As you can tell, lots of good stuff in the guides. However, there was still a couple of things I was uncertain about after reading them – like how to interact with episerver pages and workflow parameters. So here's a description of the steps I took.

Twittering

I've noticed that a lot of people tweet it, whenever they have written a blog post – and I thought: Why not automate that with a workflow. True, it'll be a fairly simple one – but then again, it felt fairly safe to start out small. Some people are opposed to automated tweets and regard it spam – but in my eyes the API and the ability to automate tweets is a key feature of Twitter. And I appreciate being updated when there are relevant blog posts for me to read – I don't always check my rss reader – but I usually follow twitter pretty closely. Anyway – tweeps (people on twitter) who complain about other peoples tweets should probably just stop following the sources of their frustration – much like people who get offended by certain tv broadcasts should consider changing the channel :-)

1. First I created a Workflow Library project in visual studio, referenced a couple of EPiServer assemblies (including EPiServer.WorkflowFoundation) and I connected the project to my web site to make sure the code would be deployed when I compiled.
2. In the workflow project I created a SequentialWorkflowActivity called "SendTwitterNotification".
3. In the designer I dragged in an IfElse (because we might only want to tweet when the page is published for the first time – not in any subsequent fixes). In one of the IfElseBranches I put a custom code activity.
4. Then I created empty handler methods for both the IfElseBranch and the custom code block. The IfElse branch requires you to create a method matching a certain signature and then assign it as a property, while the custom code let's just setup an event-handler for when it should execute.
5. I'm the kind of developer who likes to see stuff working as soon as possible, so already at this point I put in some dummy code (in the IfElse I just returned true, and in the custom code I wrote a line to a file), compiled and decided to try it out in Admin mode.
6. And to my big surprise it seemed to work in the first try. I published a page in edit-mode and could see the file the dummy code was writing to, grow!
7. Next step was to make sure I from my code would know which page it was being invoked on. This turned out to be really, really simple. Just add a public property to your workflow class of the type WorkflowPageEventArgs and call it "PageArgs". It will automatically be set when the workflow is invoked. *Just the same way you could create a property called FileSystemArgs of the type WorkflowFileSystemEventArgs and it would be assigned filesystem info if the workflow was started from a filesystem event.*

```
public WorkflowPageEventArgs PageArgs { get; set; }
```

8. Now, that the page information was here, it wasn't that hard to write the method for the IfElseBranch – basically it checks if there are previouslypublished versions of the page.

```
protected void CheckIfItIsFirstPublish(object sender, ConditionalEventArgs args)
{
    args.Result = (bool)(DataFactory.Instance.ListVersions(PageArgs.PageLink).Where(pv => pv.Status == VersionStatus.Pre
}
```

9. Now I reached a point where I needed to access some parameters for the workflow. Like which username and password to use for Twitter, and how the tweets should look like. So, first I created a user control to hold the UI (initially I created the user control in the Web Site project – and then moved it to my Workflow project), added asp.net code for showing the input boxes and implemented the IWorkflowStartParameterHandler interface on the user control. That lets you define a method for loading parameters and one for saving them – in a dictionary. Fairly easy stuff. I ended up with a code-behind for the UserControl that looked like this:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
using EPiServer.WorkflowFoundation.UI;

namespace TwitterWorkflow.WorkflowUI
{
    public partial class TwitterWorkflowUI : System.Web.UI.UserControl, IWorkflowStartParameterHandler
```

And the .ascx like this:

```
<%@ Control Language="C#" AutoEventWireup="true" CodeBehind="TwitterWorkflowUI.ascx.cs"
    Inherits="TwitterWorkflow.WorkflowUI.TwitterWorkflowUI" %>
<h2>
    Twitter Workflow</h2>
<div style="width: 100%">
    <table class="epistandardtable">
        <tr>
            <td>
                Twitter Username
            </td>
            <td>
```

Of course this needs to be polished off a bit, language translations and all – but I'm sure you get the point.

You'll note that to test this with, I created a test-account on twitter – and logged it so only approved readers can follow. The message uses regular place-holders to insert various elements one might image people would want in their tweets, like Author name, page name, shortened url and the full friendly url. Finally I put in an option to send out tweets every time the page is published – as opposed to only tweet it once.

10. In order to access these parameters in the workflow we need to do 2 things:
Create public properties in our workflow with the same name and type as the dictionary entries (then they will magically be assigned)

```
public string TwitterUsername { get; set; }
public string TwitterPassword { get; set; }
public string TwitterMessage { get; set; }
public bool TweetAlways { get; set; }
```

and attach the user control to the workflow through an attribute.

```
[WorkflowPlugIn(Area = PlugInArea.None, Url = "~/WorkflowUI/TwitterWorkflowUI.ascx")]
public sealed partial class SendTwitterNotification : SequentialWorkflowActivity
{
```

11. Now, all I had to do was to write the code that actually creates the tweet, shortens the URL and sends out the tweet.
In this post there is a nice example of sending tweets from C# without external libraries and it turns out that both TinyUrl and bit.ly has pretty decent API's for shortening urls. I went with TinyUrls – mostly because it was so simple – just send a request to
http://tinyurl.com/api-create.php?url=http://labs.episerver.com and see how it returns a short url. I also wrote a bit of code to try to keep the messages under 140 – if they are too big, it will try to cut the end of the PageName – but leave the url intact.

12. In the end, my Workflow class ended up looking like this:

```
[WorkflowPlugIn(Area = PlugInArea.None, Url = "~/WorkflowUI/TwitterWorkflowUI.ascx")]
   public sealed partial class SendTwitterNotification : SequentialWorkflowActivity
   {
       //TODO: Digg, Reddit och Delicious, stumpleupon

       public string TwitterUsername { get; set; }
       public string TwitterPassword { get; set; }
       public string TwitterMessage { get; set; }
       public bool TweetAlways { get; set; }

       public WorkflowPageEventArgs PageArgs { get; set; }
```

So – in the end, it turns out to be fairly easy to create workflows. I'll admit this was a very basic one, but definitely not impossible. One might argue that this would have been significantly easier to put directly as an eventhandler to the PagePublished event in the system, but you have to keep in mind that this way I get:

- Admin UI for selecting which pages / page types that should be affected
- Admin UI for setting up start parameters – and even defining different tweets for different pages
- A workflow, that can be used as a building block in other workflows.
- A learning experience :-)

Download User Control + assembly – just register them in Admin mode and you are good to go.

Enjoy it!

P.S.

Naturally this post was automatically tweeted by @EPiServerLabs when it was published :-)

RECENT POSTS

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