The Chris Kent

Quick Fixes, Workarounds, and Other Neat Tricks

Use Local Files in CefSharp

April 21, 2014January 14, 2015 theChrisKent C#, CefCustomScheme, CefSettings, CefSharp, Chromium, HTML, ISchemeHandler, ISchemeHandlerFactory, WinForms, WPF
Applies to: CefSharp, C#
CefSharp (https://github.com/cefsharp/CefSharp) is an open source project which provides Embedded Chromium for .NET *(WPF & WinForms)*. It's a great way to get Chrome hosted inside of your .NET application. Recently I had the need of loading web files directly from the file system. The easiest way to do that is to provide a custom scheme. You do this by creating an instance of **CefSettings** and using the **RegisterScheme** method with a **CefCustomScheme** wrapper object which requires an implementation of the **ISchemeHandlerFactory** which will in turn require an implementation of the **ISchemeHandler**. Got it?! It's actually much less confusing than it seems. It will make much more sense in a moment.

Quick Note: *This isn't a tutorial on how to get CefSharp working or integrated into your product. You'll want to install the packages using NuGet and go to the CefSharp project page (https://github.com/cefsharp/CefSharp) for details.*

ISchemeHandler

SchemeHandler (https://github.com/cefsharp/CefSharp/blob/master/CefSharp/ISchemeHandler.cs) objects process custom scheme-based requests asynchronously. In other words, you can provide a URL in the form *"**customscheme**://folder/yourfile.html"*. The SchemeHandler object takes the request and gives you a chance to provide a custom response. In our case, we want to take anything with the scheme *local* and pull it from the file system. Here's what that looks like:

```
1  using CefSharp;
2  using System.IO;
3  public class LocalSchemeHandler : ISchemeHandler
4  {
5      public bool ProcessRequestAsync(IRequest request, ISchemeHandlerRespo
6      {
7          Uri u = new Uri(request.Url);
8          String file = u.Authority + u.AbsolutePath;
9
10         if (File.Exists(file))
11         {
12             Byte[] bytes = File.ReadAllBytes(file);
13             response.ResponseStream = new MemoryStream(bytes);
14             switch (Path.GetExtension(file))
15             {
16                 case ".html":
17                     response.MimeType = "text/html";
18                     break;
19                 case ".js":
20                     response.MimeType = "text/javascript";
21                     break;
22                 case ".png":
23                     response.MimeType = "image/png";
24                     break;
25                 case ".appcache":
26                 case ".manifest":
27                     response.MimeType = "text/cache-manifest";
28                     break;
29                 default:
30                     response.MimeType = "application/octet-stream";
31                     break;
32             }
33             requestCompletedCallback();
34             return true;
35         }
36         return false;
37     }
38 }
```

I've named my implementation **LocalSchemeHandler** because I'm creative like that. You can see we are implementing the **ISchemeHandler** interface in line 3 *(If this isn't recognized be sure to include the using statements in lines 1 and 2 above).* This interface only requires a single method, **ProcessRequestAsync**. Our job is to translate the request into a response stream, call **requestCompletedCallback** and indicate if we handled the request or not *(return true or false).*

The first thing we do is translate the request URL into a local file path *(lines 7-8)*. If the file doesn't exist, there isn't any way for us to handle the request so we return false *(line 36)*. Otherwise, we set the **response.ResponseStream** to a MemoryStream from the file's bytes *(lines 12-13)*. We then guess the Mime Type based on the file's extension *(lines 14-32)*. This is a pretty limited list but it was all I needed – Just expand as necessary.

Once we've got everything we need, we call the **requestCompletedCallback** and return *true* to indicate we have handled the request.

ISchemeHandlerFactory

SchemeHandlerFactory (https://github.com/cefsharp/CefSharp/blob/master/CefSharp/ISchemeHandlerFactory.cs) objects create the appropriate SchemeHandler objects. I've also added a convenience property to help out during registration. Here's mine:

```
1  using CefSharp;
2  public class LocalSchemeHandlerFactory : ISchemeHandlerFactory
3  {
4      public ISchemeHandler Create()
5      {
6          return new LocalSchemeHandler();
7      }
8
9      public static string SchemeName { get { return "local"; } }
10 }
```

This one is called LocalSchemeHandlerFactory *(surprise!)* and you can see we are implementing the ISchemeHandlerFactory interface in line 2 *(If this isn't recognized be sure to include the using statement in line 1)*. This interface also only requires a single method, **Create**. All we have to do is return a new instance of our custom SchemeHandler *(line 6)*.

The **SchemeName** property in line 9 is just there to make things easier during registration.

Registering Your Custom Scheme

Now that we have our custom SchemeHandler and the corresponding Factory, how do we get the Chromium web browser controls to take advantage of them? This has to be done before things are initialized *(before the controls are loaded)*. In WPF this can usually be done in your ViewModel's constructor and in WinForms within the Form Load event. Here's what it looks like:

```
1  CefSettings settings = new CefSettings();
2  settings.RegisterScheme(new CefCustomScheme()
3      {
4          SchemeName = LocalSchemeHandlerFactory.SchemeName,
5          SchemeHandlerFactory = new LocalSchemeHandlerFactory()
6      });
7  Cef.Initialize(settings);
```

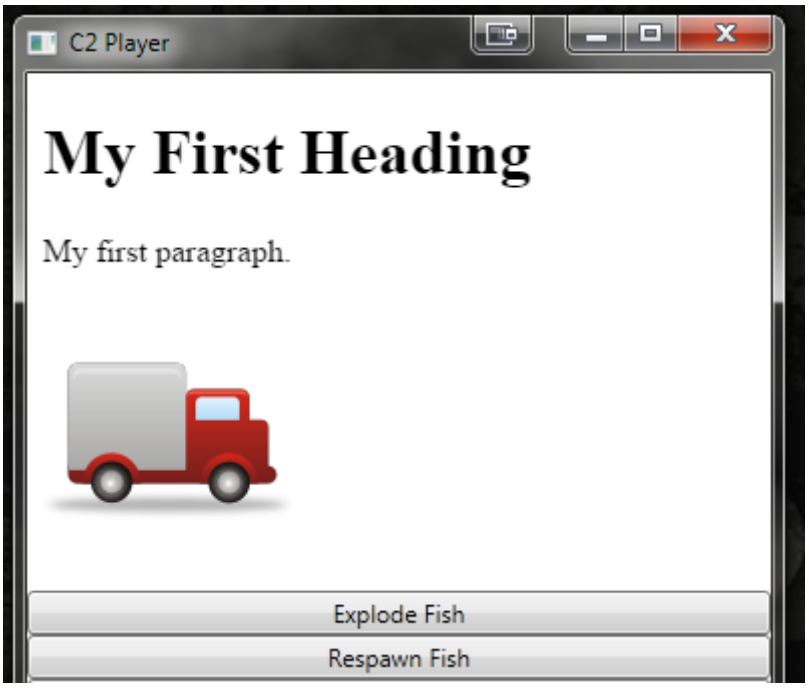
You'll create a new instance of a **CefSettings** object *(line 1)* and call the **RegisterScheme** method which takes a **CefCustomScheme** object *(lines 2-6)*. the **CefCustomScheme** object needs to have its **SchemeName** set to whatever you are going to be using in the URLs to distinguish your custom scheme *(we are using local)* and its **SchemeHandlerFactory** should be set to a new instance of your custom SchemeHandlerFactory. Then call the **Cef.Intialize** with the settings object and all the plumbing is hooked up.

Giving it a go

In my application's directory *(bin/x64/debug)* I have created a folder called web. Inside the folder is an html file *(index.html)* and an images folder with a single image *(truck.png)*. The html page is very simple:

```
<!DOCTYPE html>
<html>
<body>
<h1>My First Heading</h1>
<p>My first paragraph.</p>
<img src="images/truck.png"/>
</body>
</html>
```

Now if I set my ChromiumWebBrowser *(WPF)* or WebView *(WinForms)* **Address** property to ***local://web/index.html*** I get the following:



(https://thechriskent.files.wordpress.com/2014/04/localschemehandler.png)

You can see the web page loaded both the HTML and the linked image from the file system. There's even a couple of dummy buttons underneath the truck to show you that it's just a standard WPF window. This is all you need for simple web files stored locally. In some advanced cases you will need to adjust the **BrowserSettings** for your control to adjust permission levels *(specifically FileAccessFromFileUrlsAllowed and UniversalAccessFromFileUrlsAllowed)*.

4 thoughts on "**Use Local Files in CefSharp**"

1. **Use Embedded Resources in CefSharp | The Chris Kent** says:
   January 15, 2015 at 9:51 am

   […] my last post, Use Local Files in CefSharp, I showed you how to create a CefCustomScheme to pull web files directly from the file system. This […]

   Reply (https://thechriskent.com/2014/04/21/use-local-files-in-cefsharp/?replytocom=19131#respond)
2. **Embedded Chromium in WinForms | The Chris Kent** says:
   February 2, 2015 at 4:23 pm

[…] a previous article, Use Local Files in CefSharp, I showed you how to make the necessary objects to register a CefCustomScheme that would load […]

Reply (https://thechriskent.com/2014/04/21/use-local-files-in-cefsharp/?replytocom=20153#respond)

3. **Bartosz Baczek (@bbbonczek)** says:
April 24, 2018 at 4:09 pm

Great article ! It helped me a lot with my problem – but problem is that it is a bit outdated now. I wrote a blog post on how to use local files in CEF sharp, with newer version of CEF.

https://bbonczek.github.io/jekyll/update/2018/04/24/serve-content-in-cef-without-http-server.html (https://bbonczek.github.io/jekyll/update/2018/04/24/serve-content-in-cef-without-http-server.html)

Hope you don't mind Chris 🙂

Reply (https://thechriskent.com/2014/04/21/use-local-files-in-cefsharp/?replytocom=30498#respond)

4. **Running Local Web Pages in CefSharp.WPF – Bloodforge Blog** says:
February 19, 2020 at 11:18 pm

[…] files, so it had to be modified. Fortunately, I found some posts on how to enable that feature (here and here) – however, the first link was for an older version of CefSharp, and the […]

Reply (https://thechriskent.com/2014/04/21/use-local-files-in-cefsharp/?replytocom=58195#respond)

Ⓦ (https://wordpress.com/?ref=footer_custom_svg)