Taking the new WebView2 Control for a Spin in Markdown Monster

The `WebView2` control is a new Chromium based Web Browser control that aims to replace the Internet Explorer based Web Browser control in WPF and WinForms. There briefly was a `WebView` control too - based on the now discontinued original Edge browser, but that has now been superceded by the new WebView2. These two controls are **very different** so if you used the old `WebView` expect to require changes.

Why WebView2?

The WebBrowser control in WPF and WinForms is a built in control, and it's actually very capable and compared to the other controls - very light weight in terms of memory and resource usage. The old Web Browser control is based on Internet Explorer however and that's starting to become a problem. Although the control defaults to IE 7.0 there are ways you can make the control use IE 11 which provides HTML5 browsing functionality. But... IE 11 is discontinued and the latest HTML and JavaScript features are no longer updated and haven't been for the last 5 years or so and slowly but surely even IE 11 is falling so far behind that some library updates no longer work with it.

I use the WebBrowser control in my **Markdown Monster Markdown Editor** and it's actually been very reliable, fast and very resource friendly. MM **heavily relies on HTML rendering** as it uses a JavaScript based editor (ACE editor) for its editing surface, and a browser for previewing the HTML generated in near real time. A lot of people talk disparagingly about IE 11, but it actually works very, very well for this particular use case.

The problem however is that JavaScript libraries are starting to no longer provide updates for IE 11 - which is understandable. The world is moving to ECMAScript 2015+ and IE is not welcome for that party. Although ACE editor continues to have good support for IE 11, some of the other support libraries I use like **HighlightJs** no longer works in recent versions. Some support libraries that can be added like **Mermaid diagrams** in recent versions no longer support IE 11. While I can stick to older libraries, eventually IE 11 is going to be a problem and end up on the final trash heap of history.

The WebView2 control was announced last year and has quietly and very slowly been updated. After the initial hype about a Chromium WebBrowser control that doesn't require massive run time installs in most cases, things have slowed down as Microsoft loves to do. Make a big splash and then disappoint with trickle out features. While the control works overall it's still in pre-release mode for the WPF and WinForms controls, and is missing quite a few features some of which I'll discuss here.

The WebView2 control is exciting nevertheless because it provides:

- A Chromium based Web browser control
- Can use Evergreen features of the Edge Browser
- Doesn't require a browser runtime if Edge Chromium is installed
- Browser runtime can be installed if desired to support older (pre-Win10) clients

Chromium is the open source core engine used in most modern Web Browsers. Chrome, Edge (Chromium), Brave, Vivaldi, Opera and many more smaller browsers all use the core WebKit/Chromium based browser engine in their desktop browsers and now you can use it too easily in your .NET Desktop applications.

`WebView2` can use an existing Edge installation using an evergreen browser, which means on Windows 10 it'll be ready to run without having to install the Chromium runtime which is quite large (somewhere around 30 or so megs in an install). Optionally it's also possible to ship a browser runtime with your application which ensures that you get a fixed version of Chromium and that it works on older machines that may not have Edge Chromium installed.

Real World Experiment

Markdown Monster WebView Chromium Previewer Addin

Problems

Parameters need to be passed as string or object

For example, here's a method that expects an `int` and `bool` parameter.

However, the following does not work:

```
public void gotoLine(int editorLine, bool noRefresh)
```

but it does if parameters are converted to object.

```
public void gotoLine(object editorLine, object noRefresh)
```

It looks like the control can work with string parameters, but it doesn't do well with other types and you'll either have to pass them using `object` parameters or pass a string with JSON data.

Using JSON for Complex Data

The IE control was very easy to work with when it came to interop - you could define objects in JavaScript and call these objects easily from .NET using natural parameter syntax and even simple object structures. The reverse also worked easily: You could pass an object to a JavaScript function, and store the object in the document, and then call methods and set properties on that object using JavaScript. In both directions that just worked.

With the `WebView2` this doesn't work the same way and requires some pretty funky workarounds. JavaScript execution from .NET involves calling an `ExecuteScript()` method on an Interop object and this method only takes a single string parameter that holds an expression that is 'executed' on the document.

While it does allow you to call code in JavaScript it's pretty ugly:

```
await WebBrowser.CoreWebView2.ExecuteScriptAsync(codeToExecute);
```

The tricky part with this is if you need to parse parameters you need to properly encode them.

I created some helpers to facilitate this using a `CallMethod()` method in an interop object:

```
/// <summary>
/// Calls a method with simple parameters: String, number, boolean
/// This version returns no results.
/// </summary>
/// <param name="method"></param>
/// <param name="parameters"></param>
/// <returns></returns>
public async Task CallMethod(string method, params object[] parameters)
{
    StringBuilder sb = new StringBuilder();
    sb.Append(method + "(");

    if (parameters != null)
    {
        for (var index = 0; index < parameters.Length; index++)
        {
            object parm = parameters[index];
            var jsonParm = JsonSerializationUtils.Serialize(parm);
            sb.Append(jsonParm);
            if (index < parameters.Length - 1)
                sb.Append(",");
        }
    }
    sb.Append(")");

    await WebBrowser.CoreWebView2.ExecuteScriptAsync(sb.ToString());
}
```

So I can use something like this from C# code:

```
string selText = await JsInterop.CallMethod("getSelection");
JsInterInterop.CallMethod("setSelection","**" + selText + "**", true);
```

Both of these calls are encoded into the appropriate `ExecuteScriptAsync()` string values with the parameters properly JavaScript encoded (using JSON strings).

It works but it's tedious as heck and easy to make mistakes with.

Problem Child

So now we come to the trade-off part of using this control. It's clearly the way forward for Microsoft and Web Content rendering, but there are a few prices to pay:

- Memory Usage is high due to Chromium Engine
- Currently Runtime Installs are required
- Environment control is very limited

Runtime Installs are Required

This is probably the biggest problem with this control - it requires explicit runtime support - it's not enough to just have Microsoft Edge Chromium installed and an additional WebView runtime has to be installed.

Runtimes can be installed from here:

https://developer.microsoft.com/en-us/microsoft-edge/webview2/

The runtimes are not small (the global runtime installer is an 80mb download, while the local installable runtime is a 120mb cab file). So not something you really want to ship with 1k or 2k .NET application.

In Markdown Monster I'm side stepping shipping the runtime by checking for an installed version and if not found, directing users to download and install the runtime, then re-try. That works but yeah, it's fucking ugly.

Runtime Versions are a Pain

It gets worse though: You can't just use any runtime. The runtime **has to be of the same version or larger than the installed runtime your code is binding to** which happens to be the NuGet Packages Build number.

Not only do you need to check for whether it's installed but also check for the version and potentially install a newer version.

What this amounts to is back to fucking DLL hell. Leave it to the Windows folks to bring that back.

It sure would be nice if there was a true evergreen mode where you could just be OK with whatever version of the control is installed. But no the runtime has to be specifically matched to the library version or higher. It's not clear whether that's a function of the library → Runtime binding or the Chromium version.

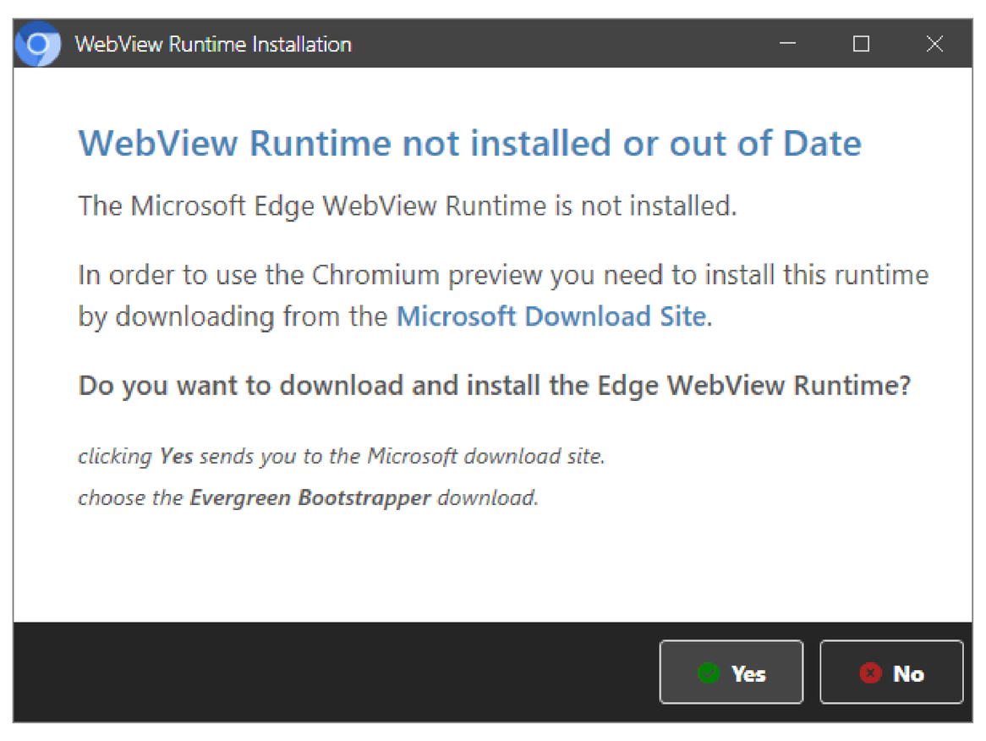
Either way this is big downside to this control as it requires very large, extra components to install that a typical non-developer is not likely to have already installed.

Booo!

Checking and Downloading

In Markdown Monster I have a check routine that checks whether the runtime is installed.

If it isn't installed I prompt the user to download and install the WebView runtime. Since the runtimes are very large (80-120mb) I don't want to ship it as part of the stock download. Rather I let people install it if it's not there or out of date:



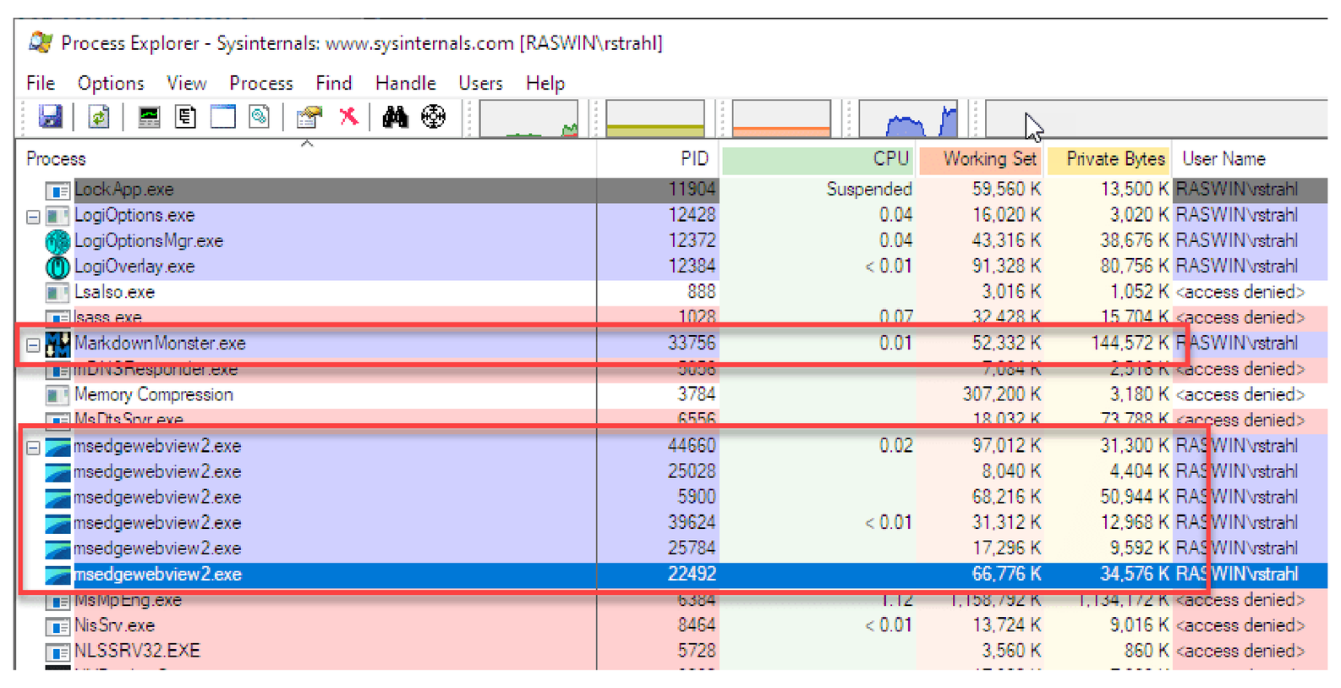

```
private bool IsWebViewVersionInstalled(bool showDownloadUi = false)
        {

            string versionNo = string.Empty;
            try
            {
                versionNo = CoreWebView2Environment.GetAvailableBrowserVersionString();

                // strip off 'canary' or 'stable' version
                versionNo = StringUtils.ExtractString(versionNo, "", " ", allowMissingEndDelimiter: true)?.Trim();
                var ver = new Version(versionNo);

                var assVersion = typeof(CoreWebView2Environment).Assembly.GetName().Version;

                if (ver.Build >= assVersion.Build)
                    return true;
            }
            catch {}

            if (!showDownloadUi)
                return false;

            var form = new BrowserMessageBox();

                var markdown = $@"
### Edge WebView Runtime not installed or out of Date
The Microsoft Edge WebView Runtime is either not installed
or is the wrong version { (versionNo != null ? " (" + versionNo + ")" : "") }.

In order to use the Chromium preview you need to install this runtime.

Do you want to download and install the Edge WebView Runtime?";
                var okButton = form.AddButton("Yes", FontAwesomeIcon.CheckCircle, Brushes.Green);
                form.ClearButtons();
                form.AddButton("No", FontAwesomeIcon.Remove, Brushes.Firebrick);
                form.ShowMarkdown(markdown);
                form.ShowDialog();

                if (form.ButtonResult == okButton)
                {
                    mmFileUtils.OpenBrowser("https://developer.microsoft.com/en-us/microsoft-edge/webview2/");
                }

                return false;

        }
```

which is then called by the startup and toggle routines:

```
// addin loader of Preview Browser
public override IPreviewBrowser GetPreviewBrowserUserControl()
{
    if (!IsActive || !IsWebViewVersionInstalled())
        return null;  // use the default IE control

    return new WebViewPreviewControl();
}

// toggle between IE and WebPreview2
public override void OnExecute(object sender)
{
    if (Model.Window == null || !IsWebViewVersionInstalled(true))
        return;

    IsActive = !IsActive;
    Model.Window.LoadPreviewBrowser();

    if (IsActive)
        Model.Window.ShowStatusSuccess("Switched to Chromium based Preview Browser.");
    else
        Model.Window.ShowStatusSuccess("Switched to Internet Explorer based Preview Browser.");
}
```

WebView and Memory: It's not LightWeight

The WebView2 control is a big ass memory hog. Check out this memory view for an MM instance with 2 documents open:



Using the old IE WebBrowser control MM would run roughly around 20mb for smallish docs and maybe somewhere around 100mb for many sizable documents open at the same time. The WebView does that old max memory usage as a bare minimum.

That's not surprising though - if you check Chrome, Edge, Brave etc. and look at their memory usage it's similar - it's an artifact of the Chromium Web Browser engine that uses available memory liberally. With some experimenting and opening a bunch of documents in MM I was unfortunately able to get it use 500mb of memory. Holy shit...

I'm not sure what to do about that - that's just a price to pay for using Chromium as a rendering engine I guess.