现代的 WEB 浏览器控件

Christof Wollenhaupt
foxpert GmbH
Ulzburger Straße 352
22846 Norderstedt, Germany
Voice: +49-40-6053373-70
Website: http://www.foxpert.com
Email: christof.wollenhaupt@foxpert.com

翻译：xinjie
QQ：411618689

多年来，在 Windows 32 位桌面应用程序中，仅有一种方法可以查看网页内容：基于 IE 的网页浏览器控件。网页浏览器控件极难驯服（除非您是 Rick Strahl），特别是当你无法控制应该显示的内容时。

现在，情况已经发生改变。微软在 2018 年宣布了 Edge Chromium 的可嵌入版本，我们在本次会议中会予以简单介绍。本次会议的重点将集中在另一个控件：Chromium 嵌入式框架。这是一个用于在应用程序中嵌入 Chromium 浏览器的开源项目。它也是 Google Chrome、Microsoft Edge 和 Opera 的开源基础。

CefSharp 项目使 CEF 成为一个 .NET WinForm 和 WPF 控件，我们可以在 Visual FoxPro 应用程序中使用它。本白皮书将介绍如何在 VFP 程序中嵌入 CefSharp，如果查看远程和本地文件，如果扩展控件，如何与网页交互，如何从 WEB 控件中运行 VFP 代码，如何与 VFP 交互数据，如何调试应用程序中的 JavaScript 代码，以及如何将 CefSharp 与应用程序一起发布。

简介

当我考虑在应用程序中嵌入一个 WEB 浏览器控件时，我是从一个业务应用程序开发人员的角度来看这个问题。我认为 WEB 浏览器控件是显示更复杂报表的一种方式，它提供更好的 UI 来显示数据，并与其他 WEB 应用（如目录系统、状态页面、预约系统等）进行交互。

微软以类似的方式使用 web browser controls 。Outlook 中的一些 UI 元素，例如侧边栏，都是 WEB 视觉元素，而不是 WPF 。

但是，至少在我看来，还有些不是很不明显的用例。似乎游戏行业依赖于呈现到游戏中的 HTML 内容。Epic Software 公司的主要游戏引擎之一虚幻引擎（Unreal Engine）包含了 CefSharp ，它可以将 HTML 内容渲染成可以在游戏中使用的纹理。大多数游戏的启动器和聊天工具也都是基于浏览器开发的。

许多应用程序是 WEB 和桌面应用。而桌面应用通常只是重新打包的 WEB 应用。如果您运行 Slack、Trello、Skype、Spotify、Amazon Music、Evernote、Signal、BaseCamp、WhatsApp、Github Desktop、Microsoft Teams、Yammer，甚至 Visual Studio Code 或 Xbox ，那么您运行的 Chromium 要么作为 Electron 的一部分，要么是 Chromium 嵌入式框架（CEF），后者，也是我们在本白皮书中论述的重点。

您应该听说过 Chrome 浏览器，即使您可能更喜欢 FireFox、Edge、Safari 或 Opera，设置是不知名的其他浏览器。Chromium 与 Chrome 浏览器到底有什么不同呢？

Chrome 是闭源的，而 Chromium 则是开源的。Chrome 浏览器与 Edge 浏览器类似，两者都是以 Chromium 为基础，但各自添加了不开源的特有功能。

Chrome 支持需要授权费的编码/解码器，如 AAC、H.264 和 MP3。这意味着 Chromium 不支持一些依赖这些现代流媒体编码/解码的视频和音乐网站。当然，如果您的应用中使用了 Chromium，也同样不支持。因为 CefSharp 也是基于 Chromium 的，但没有任何编码/解码器。如果您想知道什么时候您会需要这些解码器，让我说一个词：Netflix。

在 Chrome 中，已将 Adobe Flash 沙盒化，但是 Chromium 中默认并未安装 Adobe Flash。我们后续的讨论将说明如何克服 Chormium 的这种局限性。

Chrome 已融入 Google 基础架构。它具有自动更新、一个复杂的 UI、一个应用商店

（它是 Chrome 插件的唯一来源）、以及一个错误报告（它只能迫使风扇疯狂旋转）。大多数 Chrome 的特有功能都集中在 UI 体验上，例如输入预测、同步、反钓鱼支持。

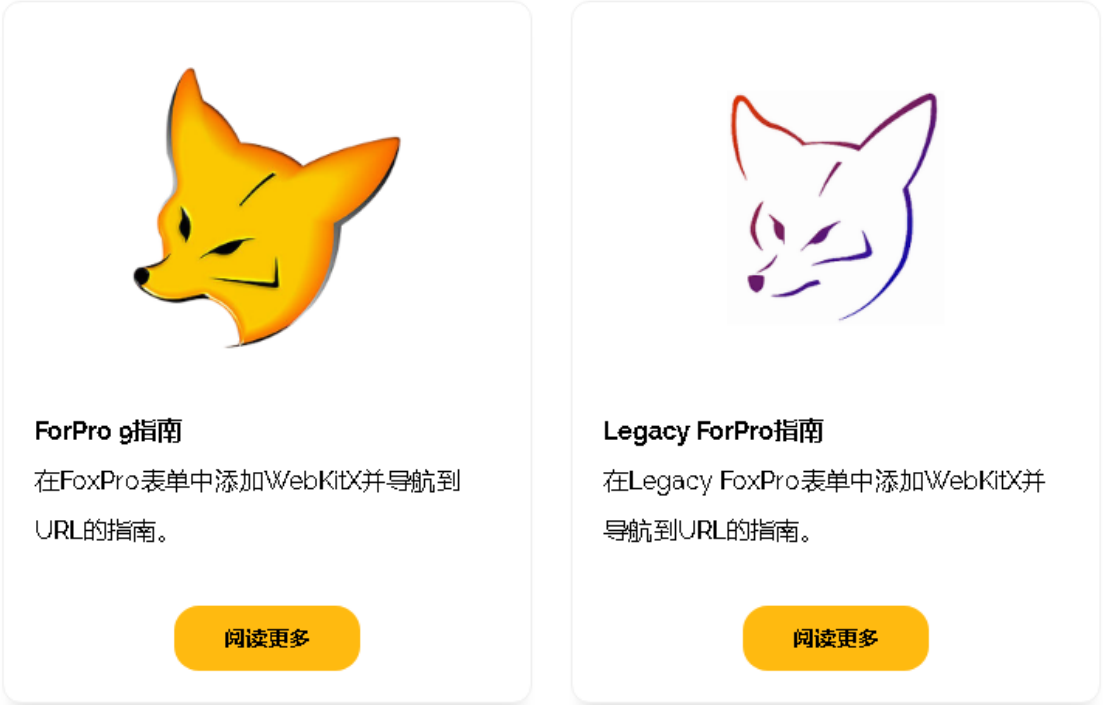
Chromium 是 Chrome、Opera、Edge 和 Amazon Silk 的基础。它基于 Safari 的基础 webkit 。webkit 是整合了 Gecko 的 KDE 浏览器引擎的基础，也就是 FireFox 的渲染引擎。几十种浏览器都是基于这些引擎的变体。而消失的引擎只有一个，那就是 Internet Explorer。

Chromium 嵌入式框架(CEF)是基于 Chroium 的。它是一个库，允许我们把 Chromium 浏览器嵌入到我们自己的应用程序中。我提到了各种基于 CEF 的应用。这里，有一家公司比任何其他公司都重要，瑞典的 Spotify ，它正在维护 Linux、Mac 和 Windows 版本。

CEF 是用 C/C++ 语言编写的，并且它们在众多语言中很容易使用（如果您认为 C++ 很容易的话）。然而，世界上大多数人，只是在商业领域，并没有人使用 C++。他们依赖 FoxPro、.NET、Jave 和 JavaScript 等工具（您没有看错，在 CEF 的维基百科文章中提到了 FoxPro）。

在 Visual FoxPro 中，有几种方法可以使用 CEF 。最近添加的一个方法是 Webkit ActiveX，这也是维基百科提到 Visual FoxPro 的原因。该控件支持自 VFP6 开始的所有 VFP 版本。有趣的是，它们的 DEMO 都是从全新安装的 VFP 开始：

https://www.webkitx.com/doc/light/Getting%20Started%20with%20FoxPro%209.html



不用感到困惑，Visual FoxPro 有时也被称为 FoxPro，我们都知道，真正的专业人士使用的都是 FoxPro 。

WebkitX 是一个商业产品，需要许可证。桌面应用的免税版许可证是 599 英镑，看清除，单位是英镑。在写这个白皮书时，这个价格折合成美元大概在 700 到 800 美元之间。如果您需要在服务器或非桌面设备上运行此控件，那需要单独购买额外的许可证。

如果您正在寻找一个受支持的商业产品，可以像其他 ActiveX 控件那样易用，那么，这个选择不会很坏。

我之所以没有走这条路而是选择使用 CefSharp，有两个原因。首先，是时间问题。我在 2015 年开始研究 CefSharp ，并在 2018 年在第一个商业应用中实现了它。那一年 WebkitX 还是一个新产品，还没有任何的业绩。

另一个原因，在切换到 wwwDotNetBridge 之后，我真的希望尽量减少我必须处理的 ActiveX 控件的数量。由于对于那些我无法控制源代码的控件。我的解决方案确实需要 ActiveX 控件，我只需几行代码就可以实现需求。但它必须是通用的，而且必须有源代码，并且，不限制我在 CefSharp 中做任何的事。

当然，CefSharp 的缺点就是比较难用，而且，学习曲线陡峭，这和其他的开源软件没太大的区别。

使用浏览器控件的注意事项

摘要

如果您仅仅想替换 Microsoft Web Browser 控件，用于显示 HTML 文档或远程的内容，那么本章节您就需要认真阅读。

文件附件包含 CefSharp V84 版本。也就是 2020 年 9 月的版本。将各种文件复制到您的项目文件夹中：

cef-V84.4.10——此文件夹必须是项目根目录的子文件夹。整个文件夹的内容需要作为 EXE 的子文件夹部署到您的客户端。是的，它很大，152MB。但是，请看看 CefSharp 消耗了多少内存。毕竟，它是 Chrome。

ftDotNet——该文件夹的内容您可以随意放置。但是，由于这是一个 ActiveX 控件，因此，您必须在计算机上注册 DLL，而且，在您的客户端计算机上也需要注册。注册的时候，需要管理员权限。这是一个 .NET 4.0 组件，这意味着您注册时要使用 regasm 而不是 regsvr32。

C:\Windows\Microsoft.NET\Framework\v4.0.30319\regasm fpDotNet.dll /tlb fpDotNet.tlb /codebase

在 CMD 窗口以管理员身份运行上面这行代码。如果您在安装 OS 时更改了默认文件夹，那么，您需要做相应的修改。**不要用您自己的副本替代 wwDotNetBridge.dll。这是一个修改后的特定版本。**

ClrHost.dll、wwDotNetBridge.dll——这些文件必须位于您的搜索路径中，以便 VFP 可以加载它们。**不要用您自己的副本替代 wwDotNetBridge.dll。这是一个修改后的特定版本。**

cefSharpBrowser.prg 和各种 .h 文件——如果您将 PRG 文件包含在项目中，那么这些文件可以放置在项目的任意位置。

其他的所有文件都是演示文件，您可以根据需要自行修改。如果要显示 HTML 内容，您需要两个步骤。

VFP 中的可视控件是 ActiveX 控件。打开需要添加浏览器的类或者表单，然后添加一个 OLEControl 实例，并从可用类列表中选择 fpDot.DotNetContainer 。如果您没有在可用列表中找到，那么请您从头重新阅读本白皮书。

接下来，将 CefSharpBrowser 的实例绑定到 UI 控件。这仅需两行简单的代码。我选择将对象存储在名为 oCefSharpBrowser 的属性中：

```
This.oCefSharpBrowser = NewObject ("CefSharpBrowser", "CefSharpBrowser.fxp")
This.oCefSharpBrowser.BindToHost (This.oleBrowser, m.lcUrl)
```

示例中的 lcURL 是包含完整的 WEB 地址（包括 http、https 或文件）的变量。运行此代码，经过短暂的延迟后，您应该在浏览器中看到 HTML 文档。

如果您遇到任何问题，或者想知道还能做什么，那么，请继续阅读本白皮书。

技术问题

在文件附带的示例中包含文件 CefSharpBrowser.prg，该类是 CefSharp 的 VFP 封装。如果您觉得您需要的这个类已经提供，那么，您完全可以跳过本节。其余各节介绍如何使用它，如何响应 HTML 文档中的事件，以及如何在应用程序中分发 cefSharp 组件。

在 VFP 中托管 WinForm 控件

在您的应用程序中使用浏览器控件主要是为了能够显示 HTML 文档。cefSharp 有三个

选项来显示 UI：OffScreen、WPF 和 WinForms。

OffScreen 将页面渲染成一个位图，理论上我们可以用 VFP 中的 image 控件来显示。对于一个静态页面，我甚至考虑过这种想法。大多数网页允许滚动，并与输入进行交互，一般来说，需要在短时间内对 UI 进行多次更新。VFP 并不特别适合显示频繁变化的位图。您需要不断的更改文件名，并频繁的调用 CLEAR RESOURCES 。根据我的经验，当使用大量图片时，屏幕更新会变得相当迟缓。我也害怕自己不得不处理包括键盘、鼠标和触摸屏的输入。

另外两个选项是 WPF 和 WinForm ，它们都是 .NET UI 技术，无法直接在 VFP 中工作。然而，我们可以使用 COM 技术使之与 VFP 应用程序进行交互。这两种技术都可以在 VFP 中工作，因为它们都可以作为 OLEContainer 控件托管在 VFP 的表单中。我选择使用 WinForm，因为 WInForm 在渲染方面更接近 VFP 。

.NET 控件用于 VFP 的经典方法是编写一个包装程序，该包装程序提供您需要从 VFP 调用的所有方法。这次，我也是别无选择。我真的不想在应用程序中附带更多的 ActiveX 控件，不想处理因它产生的注册、更新以及其他所有问题。

无法完全避免使用 ActiveX 控件，因为 COM 是 VFP 支持的唯一接口，该 UI 可以充分集成到 VFP UI 中。其他的选择包括创建可以自己处理所有 UI 活动的低级窗口，要使它们集成到表单上的 TAB 顺序、在 pageframe 内表现正确、在容器内进行剪辑、以及处理具有焦点时的键盘事件，这将是一件相当耗时繁琐的事。

因此，我决定在 .NET 中再创建一个 ActiveX 控件。这是一个空的 UserControl，充当加载任何 WinForm 控件的外壳。这也就是 fpDotNet.DotetContainer 类。我希望它的功能尽可能的少。当然，我不想用 C# 实现 CefSharp 的所有接口，然后从 VFP 中调用。

相反，我使用 wwDotNetBridge 与 CefSharp 进行“通信”。我还使用 wwDotNetBridge 将控件加载到应用程序中，然后将其添加到我的 DotNetContainer 类中。SetControl 方法接收 WinForm 控件，它可以是继承自 Control 的任何控件，并将其作为唯一的控件添加到容器中。添加的控件将自动填充整个容器。

此代码非常简单。唯一要克服的障碍是 VFP 没有在 ActiveX 控件列表中将其显示出来，因为默认的 .NET COM 注册不包含“Control”键。

```
/// <summary>
/// VFP won't add a COM object to a form unless it has a Control key.
/// </summary>
/// <param name="keyName"></param>
```

```
        [ComRegisterFunction]
        public static void RegisterClass(string keyName)
        {
            using (RegistryKey key =
Registry.ClassesRoot.OpenSubKey(keyName.Replace(@"HKEY_CLASSES_ROOT\", ""), true))
                key?.CreateSubKey("Control")?.Close();
        }

        [ComUnregisterFunction]
        public static void UnregisterClass(string keyName)
        {
            using (RegistryKey key =
Registry.ClassesRoot.OpenSubKey(keyName.Replace(@"HKEY_CLASSES_ROOT\", ""), true))
                key?.DeleteSubKey("Control", false);
        }
```

当我明白症结所在后，很快就用自己的 ComRegisterFunction 解决了这个问题。

什么是应用程序域（AppDomain）？

在经历了重启 VFP 几次之后，我终于成功了。我拥有了第一个版本的 .NET 程序集。我将其注册为控件，甚至可以将其添加到新的 VFP 表单中。

接下来，我要开始向表单添加控件。我初始化了 wwDotNetBridge，并成功加载了所有需要的程序集，实例化了 WinForm 浏览器控件。但是，当我将其添加到我的 host 时，出现了各种奇怪的错误信息…程序集未加载、类型找不到…简直是一团糟。

.NET 具有常规 Windows 应用程序所没有的另一种隔离级别。您只能将一个 .NET Runtime 副本加载到一个进程中。但是，您可以将多个应用程序域(AppDomain)加载到同一个进程。应用程序域（AppDomain)就像是进程中的进程一样。.NET 中的所有对象和设置都在应用程序域（AppDomain）级别上进行隔离。

如果您在一个应用程序域（AppDomain）中创建了一个对象，而在另一个应用程序域（AppDomain）中创建了另一个对象，那么这两个对象对话的方式只能是远程和 marshalling 。另一个对象也许是在另一台 PC 上。从技术上来说，这没有任何的区别（尽管速度比较慢)。

当 VFP 加载 .NET 程序集时，它实际上会加载 mscoree.dll，这是实现所有 COM InterOP 魔术的 .NET 宿主控件。加载 mscoree 时，它将为其承载的 COM 控件创建一个新的应用程序域（AppDomain)。加载 wwDotNetBridge 是，它会执行相同的操作。它为实

例化的类型创建一个新的应用程序域（AppDomain），然后创建 C# wwDontNetBridge 的实例。

如果要使用 wwDontNetBridge 操作表单上的对象，那么 wwDotNetBridge 必须与 COM 位于同一应用程序域（AppDomain）。非常幸运，修复这个问题非常简单：

```
public Object CreateBridge()
{
    var bridge = new wwDotNetBridge();
    return bridge;
}
```

我在 wwDotNetBridge 中添加了一个新的方法，该方法用于创建 wwDotNetBridge 实例。当然，这也意味着我将 wwDotNetBridge 的版本与 fpDotNet.dll 紧密结合在一起。请不要忘记，如果不做一些额外的工作，您将无法独立更改两个 DLL。具体说，就是您需要为 EXE 提供一个清单，将 fpDotNet.dll 请求的 wwDotNetBridge 版本重定向到您应用程序中使用的版本。

在 VFP 方面我做了两个额外的变动。第一，我保留了我自己的 wwDotNetBridge 实例副本。

```
This.oDotNetBridge = m.toHost.CreateBridge()
```

此代码调用我上面定义的 C# 方法。然后，每当我创建 wwDotNetBridge VFP 类实例时，我使用下面的代码来进行切换：

```
Local loBridge as wwdotnetbridge of wwdotnetbridge.prg
loBridge = NewObject("wwDotNetBridge", "wwDotNetBridge.fxp", "", "v4")
loBridge.oDotNetBridge = This.oDotNetBridge
```

.NET 世界的噩梦：版本控制

让我们先明确一下…用 .NET 进行版本管理比之前我们生活在 DLL 地狱的时候要好很多。不过，这次，我遇到一个挑战，这是我之前没有考虑到的。先让我们回顾下 .NET 中的版本管理是如何工作的。

一个项目只能访问项目中定义的类型。对于任何其他类型，包括 .NET 框架类型，您必须让编译器知道这些类型。这个过程被称为向项目添加引用。可以通过多种方式添加引用。您可以选择磁盘上某个位置的 DLL，也可以引用全局程序集缓存，可以引用另一个项目的输出，可以选择 COM 对象，甚至是一个 nuget 包。

无论您以何种方式添加引用，最终，它们都指向一个特定的 EXE 或 DLL。具体可能是值版本号，甚至是由哈希值标识的实际内容。每当访问特定 DLL 中的类型时，它将在程序文件夹、全局程序集缓存和您可能指定的文件夹中查找与该 DLL 完全相同的版本。

如果您具有名称相同、类型相同但版本号不同的 DLL，那么 .NET 框架将不会加载此 DL，而是报告缺失 DLL 的错误。如果您使用的 DLL 也依赖于项目中使用的同一其他 DL，但又需要不同的版本，那么，您真的很值得同情。

这就是我们在 .NET 中进行版本重定向的原因。您可以在应用程序清单中为每个程序集指定可接受的版本范围。每当另一个程序集或您的应用程序请求此 DLL 的版本在配置的版本范围内时，.NET 框架就会将此文件解析为清单文件中指定的特定版本，并于您在项目中应用的版本相同。

换句话说，您可以告诉 .NET 框架，只请求版本介于 1.0 和 12.0 之间的版本，而您只需要 5.7.1 版本。这个版本，是您应用程序附带的版本。

对于 CefSharp ，我需要的恰恰相反。CefSharp 是一个巨大的包，平均每隔一个月就会发布一个新的版本。对于大多数情况，我运行的是 74、79 还是 85 版并不重要。我们定期向几千名用户发送这个应用程序的更新，并在两次更新之间根据支持需求向个别的客户单独发送。在我们的应用程序中，我必须额外交付 150 MB 才能快速修复我们的应用程序这一想法，好像并不是特别的吸引人。

这有点像 Un-Fox。在 VFP 中，我可以加载使用任何磁盘上可用的库或者文件。如果我需要一个特定的版本，我会检查代码。所以，我设想，我会把 CefSharp 的所有东西都放置在一个文件夹中，比如 cef-bin-v84.4.10 。在我的代码中，我会有一个支持的版本列表，在列表中找到最高的版本，然后简单的加载它。

这样做的好处是，当我们遇到问题时，我们很容易的安装一个较新或者较旧的 CefSharp 版本。当我仅用 CefSharp 来显示页面时，这种方法工作的很好。事实上，我甚至都没有一丁点儿关于版本管理方面的担心。

但是，我后来不得不添加处理回调的代码。这要求我实现 CefSharp 的某些接口，这就要求我必须为我的项目添加一个引用，也就意味着我被绑定到一个特定的版本。

不幸的是，版本重定向是一个多对一的关系。您可以指定每当有任何介于 v79 和 v85 之间的东西被请求时去使用 v79 版本。但是你不能反其道而行之，指定当要求使用 v79（我编译项目时的版本）时，返回 v79 和 v85 之间的任何可用的版本。

既没有多对一的关系，也没有多对多的关系来进行版本重定向。所以，我只能有几个不

太理想的选择:

我可以修改应用程序的配置文件，来配置客户端计算机上安装的版本。这要求我们具有对程序文件夹的写访问权限（我们目前的方案，因为它是一个古老的应用程序），并且在修改清单文件后重新启动该应用程序。

此外，我还可以针对我们支持的任何版本的 CefSharp 编译 DLL，这会让构建过程变的复杂。现在，我不仅要构建项目，还需要重新配置 nuget 软件包，卸载当前的版本并安装一个新的版本，然后构建项目，然后对所有其他受支持的版本重复上述步骤。或者，我备份项目，每当我们需要 DLL 中的新功能时，都在几个位置进行代码的维护。

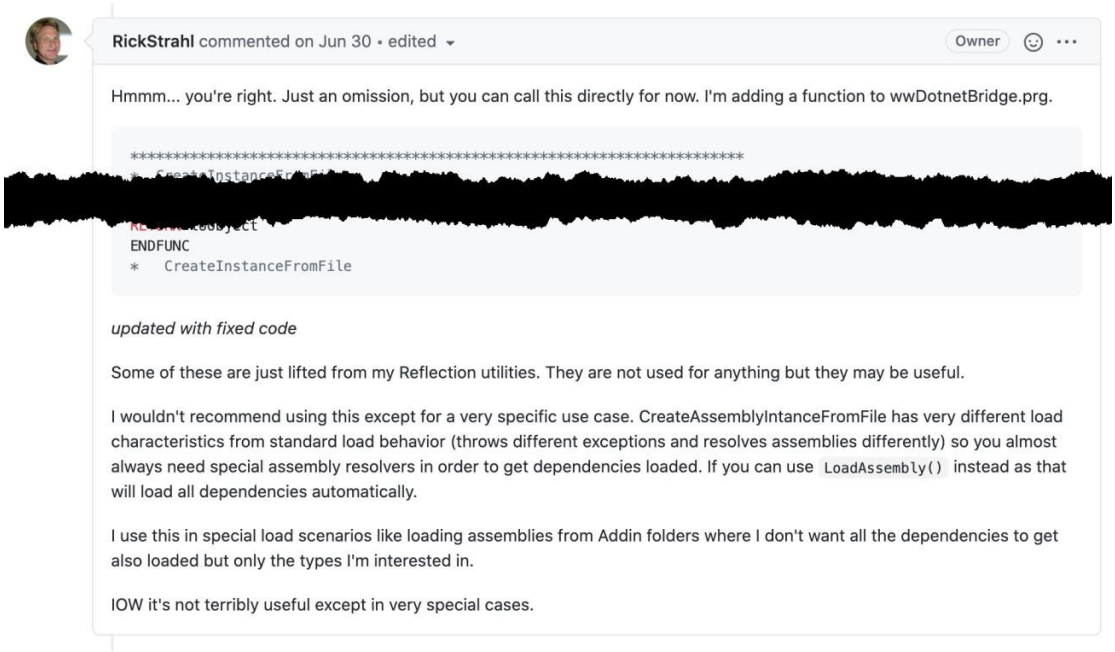
加载程序集，一段迷茫的旅程

在我现在的 CefSharp 的第一个版本中，我犯了一个错误。或者说，我学到了一些新的东西。这个项目始于一个客户需求。所以，我像之前概述的那样创建了 CefSharp 文件夹，但是我把 CefSharp 集成代码添加到了我们已经使用并与 EXE 一起提供的现有 .NET DLL 中。

为了避免版本问题，我最初决定仅支持 CefSharp 的单一版本。因为我需要在本次会议之前完成此功能。目前来说是可行的，在发布第一个版本之后，我有时间来弄清楚版本问题。除非这个版本不可用。

我从 CefSharp 文件夹加载 CefSharp DlIs，而 CefSharp 文件夹正是我构建库的同一版本。接下来，我将加载引用 CefSharp Dll 的 Dll，它失败了。而且仅在 EXE 中发生，在 VFP 开发环境中却不会。该错误消息是可怕的“Missing assembly”，尽管前面的代码行正是这个 DLL 的 LoadAssembly 调用，并且是成功的。

这时我才意识到（在 Rick Strahl 向我指出了明显的问题之后）什么是 AssemblyLoadContext 。



RickStrahl commented on Jun 30 • edited — Owner

Hmmm... you're right. Just an omission, but you can call this directly for now. I'm adding a function to wwDotnetBridge.prg.

```
ENDFUNC
*   CreateInstanceFromFile
```

updated with fixed code

Some of these are just lifted from my Reflection utilities. They are not used for anything but they may be useful.

I wouldn't recommend using this except for a very specific use case. CreateAssemblyIntanceFromFile has very different load characteristics from standard load behavior (throws different exceptions and resolves assemblies differently) so you almost always need special assembly resolvers in order to get dependencies loaded. If you can use `LoadAssembly()` instead as that will load all dependencies automatically.

I use this in special load scenarios like loading assemblies from Addin folders where I don't want all the dependencies to get also loaded but only the types I'm interested in.

IOW it's not terribly useful except in very special cases.

在 VFP 中，一旦您加载了一个库，它也会被加载。在 Windows 中，一旦加载了 DLL，也也会被加载。在.NET 中，当您将 DLL 加载到应用程序域（AppDomain）中时，它也会被加载，但并不是所有人都可以访问。

有两种不同的 AssemblyLoadContexts：Default Load 和 Load-From。从技术上讲，还有一个 No Context 上下文，您甚至可以自己创建自己的上下文。但这些与现在的问题无关。

任何被加载到 Load-From 上下文的程序集都可以看到 Load-From 和 Default Load 上下文中的任何程序集。

装入 Default Load 上下文的程序集看不到 Load-From 上下文中的程序集。如果 Default Load 上下文中的任何程序集需要一个已经加载到 Load-From 上下文中的程序集，那么，该程序集就不能加载到 Default Load 上下文中。

一个标准的 .NET 应用程序是用 .NET 编写的 EXE（或 .NET Core 编写的 DLL）。所有的依赖关系通常是自动解析的，因为它们是应用程序已知的，并且与应用程序一起位于程序文件夹中。基本上，.NET 应用程序将程序集加载到 Default Load 上下文中。仅在特殊情况下，您才能以 .NET 开发人员的身份处理 Load-From 上下文。例如，您可以在搜索路径之外的文件夹中加载插件。

FoxPro 开发者都是从本地应用程序开始。我们使用 wwDotNetBridge 来加载我们的程序集。

如果在 VFP 的 IDE 里运行代码，那么 .NET 的探测路径为 VFP 程序文件夹而不是项目文件夹。因此，在测试中，不会自动解决 DLL 的依赖关系，最终您只会从非默认目录中使

用 LoadAssembly 加载所需的所有 DLL。它们最终都能在 Load-From 上下文找到彼此。但是，如果在 EXE 中，则项目文件夹下的所有 DLL 都位于所谓的探测路径中。而这时情况就会变得混乱。从 CefSharp 文件夹中加载的 DLL 最后都在 Load-From 上下文。

我将自己的 DLL 放在项目文件夹中，因为这是作为 .NET 开发人员的习惯。LoadAssembly 方法最终在 .NET 中调用 Assembly.LoadFrom()。LoadFrom 将程序集加载到两个不同的上下文中。当 DLL 位于程序文件夹（探测路径）中时，程序集将在 Default Load 上下文中结束，否则，将在 Load-From 上下文结束。

当我们在 EXE 中加载时，我的 DLL 最终出现在 Default Load 上下文中，而从 VFP 加载时，我的 DLL 最终出现在 Load-From 上下文中。它取决于 CefSHarp 特定的版本。将 DLL 加载到 Load-From 上下文之后，它将解析 CefSharp 作为其依赖项，因为 Load-From 上下文的程序集对 Load-Fromm 上下文中的其他程序集是可见的。

当将它加载到 Default Load 上下文中时，.NET 框架尝试通过仅查看 Default Load 上下文中的程序集来解决依赖关系。但是 CefSharp 并不会加载到那里。CefSharp 在探测路径中均不可用。因此，我收到一条错误消息。

我最终将 DLL 分为两部分，并将 CefSharp 依赖部分放入 CefSharp 文件夹中，除了始终加载到 Default Load 上下文中的 .NET 框架库外，它没有对其他 DLL 的引用。

用 wwDotNetBridge 解决程序集

从理论上讲，还有另一种处理加载程序集的方法。您可以通过绑定到 AssemblyLoadContext.Default.Resolving 事件，然后将程序集从子文件夹馈送到 Default Load 上下文中。但是，这种方法我还没有进行测试。

打印内容

您的浏览器最终将显示所需的内容，运行 JavaScript 代码就能提供现代浏览器应支持的所有功能。

无论您在屏幕上看到的效果有多棒，总有用户需要打印其中的内容。这并不意味着只有砍树才能造纸。Windows 已经开始使用 PDF 打印机驱动程序了。所显示的文档很可能需要发送给客户，与文档管理系统一起归档或添加到其他工作流程中。

实际上，生成 HTML 比 VFP 的报表灵活的多，因为您可以直接使用所有流布局和格

式。使用自定义 CSS，您甚至可以让用户设置报表格式。如果您可以从 HTML 内容自动生成 PDF 文件，那不是更好吗？

您可能还记得，用微软的 WEB 浏览器控件打印是多么的不容易。基本上，任何在用户界面上可以实现的事情都发生在一个名为 ExecWB 的方法中。例如，您将 IDM_EXECPRINT 传递给该方法，它就执行与 Internet Explorer 中的“文件-打印”。

如果您希望这次有一个更好的界面，很抱歉，您可能会失望。不过换个界面毕竟会有一些不同。

我们使用 ChromiumWebBrowser 类来显示内容。这个类实现了 IWebBrowser（不要和 IBrowser 混淆）。如果您查看文档，您会发现该控件有一个 Print 方法。这似乎很容易调用，但是，当您尝试的话，您会得到一个错误信息，该方法不存在。

很多时候您会发现线索其实就在文档中。这里也一样，Print 就是 C# 所谓的扩展方法。为什么一定要这么复杂呢？

CefSharp 是一个独立使用 UI 技术的浏览器实现。WinForm（我们正在使用的）中有一个，WPF 也有一个，甚至还有一个可以渲染成位图的实现。以中立的方式与浏览器窗口交互的接口是 IWebBrowser。这个 “I” 告诉我们，这是一个没有任何附加实现的接口。

然而，对于不同的技术而言，相当多的实现实际上是相同的。实现是在一个名为 cefSharp.WebbowserExtension 的帮助类中。这个类的名称来源于您在 C# 中可以做一个快捷方式。当您拥有一个具有静态方法的静态类，就意味着没有对象参与，您需要使用一种特殊的语法来调用这些扩展方法，就好像它们是它们扩展的类的本地方法。

在 C# 中这是透明的，而在 VFP 中，我们需要显式的处理。

与浏览器控件的交互

到目前为止，我们已经在浏览器中加载了 HTML，并允许用户与文档的交互就像真正的浏览器中一样。

这对于 HTML 报表、所嵌入的独立工具或显示文档来说是非常有用的。例如，在一个应用程序中，我们扫描图片或文档。我们提供的选项是创建一个多页的 HTML 文档，将所有图片合并到一个文档中。我们在应用中使用浏览器控件来显示这个文档，但我们不需要以任何方式与之交互。

当我们可以对文档中的事件做出响应时，浏览器控件就具有了更强悍的功能。这需要两个步骤，第一步是能够对浏览器中所发生的事件做出反应，也就是说用户执行操作时得到通知；第二步是对这种反应做出反馈，例如更新内容。

创建自己的协议

大多数开发人员会使用 WEB 浏览器控件中的 BeforeNavigate2 事件来拦截来自卢兰其的操作。您会从 HTML 表单中得到接收的 URL、头信息、附加的 POST 数据；基本上，这些信息与 HTTP 请求中的信息是一样的。然后，您可以针对这些数据采取一些行动，决定取消请求或者让浏览器控件将其发送给实际的接收者。

为了拦截 URL，最常见的方法是对 URL 进行编码，创建一个假的协议。例如，导航到下一条记录的按钮将执行“app:next”。这样就不会干扰任何实际的链接，也很容易检查出一个特殊的链接。我们只需要检查前四个字符是否是“app:”就可以了。

这是我最熟悉的方法，也是我使用 CefSharp 时尝试的第一个方法。WEB 浏览器控件会给我呈现一个简单的界面，但是选择有限，而 CefSharp 则提供了丰富的对象、方法、控件、命名空间，以及你所能想到的一切。到目前为止，我找到两种创建自己的协议的方法。

比较常见的是为您的新协议创建一个处理程序。在 CefSharp 中，协议有时也被称为方案（schemes）。

```
*=============================================================================
* Creating your own protocol...
*=============================================================================
    #include Acodey.h
    *-----------------------------------------------------------------------------
    * Create a factory instance. cefBrowser calls the factory to create a new
handler
    * instance. This is a special implementation that calls the OnProcessRequest
    * method.
    *-----------------------------------------------------------------------------
    Local loFactory
    loFactory = toBridge.CreateInstance ;
     ("fpDotNet.fpCefSharp.fpSchemeHandlerFactory")
    #If __DEBUGLEVEL >= __DEBUG_REGULAR
             Assert Vartype (m.loFactory) == T_OBJECT
    #Endif
    *-----------------------------------------------------------------------------
    * Create a new protocol (scheme)
```

```
*-----------------------------------------------------------------------------
Local loScheme
loScheme = toBridge.CreateInstance ("CefSharp.CefCustomScheme")
loScheme.SchemeName = "app"
toBridge.SetProperty (m.loScheme, "SchemeHandlerFactory", m.loFactory)
toBridge.InvokeMethod (m.toSettings, "RegisterScheme", m.loScheme)
```

该代码直截了当。在应用程序中配置 CefSharp 时，您需要注册一个 SchemeHandlerFactory。它必须是实现 ISchemeHandlerFactory 的类。每当处理一个请求时，CefSharp 就会调用这个对象。然而，顾名思义，工厂对象并不会直接处理请求。

相反，它创建一个新的 SchemeHandler 对象实例来处理请求。这需要在 C# 中编写一些额外的类，我们无法通过 wwDotNetBridge 轻松实现。稍后我们将详细研究这个类。

您需要谨记，每个进程只能初始化一次 CefSharp 。这意味着您必须在您的应用程序中一次性注册所有您需要的协议。您不能将您的应用程序分解成独立的部分，这些部分可以根据当时的需要注册和配置浏览器窗口。

在简单的测试页面中，我这个方案可以很好的工作。我能够显示数据和一个按钮，这个按钮调用我自己的方案。当我点击按钮时，VFP 中的 OnProcessRequest 方法就会被调用，在那里，我可以做任何我想做的事。

但是，我需要与另一个 WEB 应用程序进行交互。我们的应用程序将把用户登录到一个远程目录系统中，并在我们的应用程序中以表单的形式显示。用户能够根据目录系统处理的标准从目录中选择项目。最后，我们需要检索这个项目列表，并将其导入到我们的系统中。

系统提供了一个回调的 URL 和一个让用户启动数据传输的方法。我们传递一个自定义 URL。它们使用 JavaScript 创建新的 WEB 请求，将所有项目组合成 XML 字符串，然后将 POST 请求发送到我们指定的 URL。

但是它不起作用。

因为对于 JavaScript 请求，有一个名为 CORS 的东西，即跨域资源共享。JavaScript 和浏览器有严格的规则，可以从代码连接到哪些服务器。已经从一个服务器下载的代码无法使用另一种协议将数据发送到另一台服务器。

file:// vs http:// vs https://

原来，CefSharp 对待我自己的 schemes 更像 file:// 。以 file:// 开头的 URL 指向的是您电脑上的的一个本地文件。因为它可能会被滥用，从您的计算机向一些服务器发送恶意的

数据，所以，文件 schemes 对其他服务器的作用非常有限。反之亦然。从服务器加载的网页无法本地文件，比如我们刚刚创建的方案。

虽然有几个选项可以配置自己的协议，但仍旧存在一个问题，即，从一个基于 https 协议的网站到“app://import”的 URL 改变了协议，会产生各种跨协议的影响。例如，CefSharp（实际上是 CefSharp 背后的 CEF）对每一个 schemes 和域名的组合都会运行一个渲染的过程。

有些功能只适用于 http:// 和 https:// 但不适用于任何自定义 schemes 或任何 file:// schemes。这包括跨域发布数据和如摄像头和麦克风这样的隐私相关的服务。

当文档都说你尽可能选择 http 或 https 而不是创建自己的 achemes 时，您应该迷途知返。

创建自己的服务

实际上，CefSharp 提供了一种比较方便的方式来挂钩 http 和 https 请求，基本上，就是创建一个只为您的应用服务的虚拟 WEB 服务。这种方法还有一个巨大的优势。

当您使用 WEB 浏览器控件时，您必须将 HTML 加载到控件中。当您显示一个远程网站并将其集成到您的应用程序时，这一切都易如反掌。您只需简单的导航到正确的 URL 。通常情况下，您希望在 WEB 浏览器控件中显示应用程序特定的内容，或者是应用程序生成的内容。

有两种常见的方法可以将任意一个 HTML 文档显示在 WEB 控件中。您可以创建一个临时的 HTML 文件，并使用 file:// 协议加载；或者，您可以加载一个页面，比如“about:blank”，然后操作 DOM 来加载字符串。

这两种方法虽然都可以，但是，它们都不在 internet ，而且，代码可以做的事都受到限制。对于基于 Internet Explorer 的控件来说，这可能不是个大问题，因为无论如何它都不支持很多现代的 API。然而，对于 CefSharp 来说，同样的方法会严重限制 HTML5 的功能以及 WEB 服务的互操作性。

语法与创建自己的协议一样，但是你不需要命名您的协议，而是简单的指定“https”：

```
*-------------------------------------------------------------------------------
* Create and configure a schema. MyApp acts as a virtual server.
*-------------------------------------------------------------------------------
Local loScheme
loScheme = toBridge.CreateInstance ("CefSharp.CefCustomScheme")
```

```
loScheme.SchemeName = m.tcProtocol
loScheme.DomainName = This.cHostName
toBridge.SetProperty (m.loScheme, "SchemeHandlerFactory", m.toFactory)
```

神奇的事情发生在为 DomainName 赋值上。您在这里可以使用任何在语法上有效的域名。该名称不一定是真实的域名或实际的计算机名称，您也可以使用“ MyApp”、您的应用程序的实际名称、甚至是进程名称，如：“<<name>>.EXE”。

对外部服务器或本地网络中的计算机的任何 http 请求都会像正常看到的那样处理。指向您指定的域名的 URL 会被拦截，并发送到您自己的 schemes 处理程序。像这样的请求

https://myapp/some-folder/somefile.app?name1=value1

会发送到您的代码中，在那里，您可以得到网络服务器收到的所有东西，例如 POST 和 PUT 的数据、HTTP verb、Internet headers、cookies 等等。代码中提供了处理请求所需的一切。请求的来源无关紧要。无论是跟踪链接的用户、JavaScript 库的源属性、图像控件、HTML 表单 submission 或来自某个库的 AJAX 请求。这一切都会触发您的代码。

CefSharp 使用一个小小的 .NET 实用程序类 fpCefSharp 来处理这些调用。fpCefSharp 是开源的，您可在 Github 上得到：https://github.com/cwollenhaupt/fpCefSharp 。

fpCefSharp 和所有其他 .NET 程序集一样，是针对其他程序集的特定版本编译的。在这种情况下，它需要 CefSharp v84.4.10 和至少 4.7.2 版本的 Micorsoft .NET Framework 。fpCefSharp 的发行版本包含了所有需要的 CefSharp 二进制文件。如果您使用了不同版本的 CefSharp ，那么您必须重新编译项目，或者在您的应用程序清单文件中添加版本重定向。

fpCefSharp 仅有几个类型。fpSchemeHandlerFactory 是 ISchemeHandlerFatory 的实现。对于你的 VFP 应用程序打开的每一个浏览器窗口，您可以为每个请求指定一个不同的对象和方法。fpSchemeHandlerFactory 管理回调的集合，这些回调将链接到它们各自对应的窗口。

每当 CefSharp 请求一个新的 SchemeHandler 时，它就会得到一个新的 SchemeHandler，这个 SchemeHandler 会链接到相应的基于浏览器窗口的 VFP 回调对象。

fpHandler 是 ResoureHandler 的一个实现，它处理一个页面（意即资源）的任何请求，它将调用转发给 VFP 回调对象。

fpCallback 对象管理对 VFP 实现的引用，并且将任何的调用都转发给它。这个对象在浏览器窗口打开时被维护，而 fpCallback 则为每个请求重新创建。

最后，这三种类型的组合导致 FoxPro 代码在每个请求中被调用。在文件

CefSharpBrowser.prg 中，CefSharpBrowser 类的 OnProcessRequest 方法用来处理这些请求。

OnProcessRequest 方法具有多种操作模式。它根据请求的资源进行区分。当请求指向一个具有扩展名的文件时，比如 index.html，那么 CefSharpBrowser.prg 假定这是一个属性；如果没有扩展名，那就被视为一个方法。我们还将在后面介绍第三种针对 REST API 的模式。

所有这些属性和方法都位于 config 对象，您将其作为第三个参数传入 BindToHost() 方法。让我们来看看最基本的实现。

在您还没有建立与外部 WEB 应用程序交互前，您只能使用可以在浏览器中显示的 HTML 文档。就像前面描述的那样，我们创建一个临时的 HTML 文件，并使用 file:// 协议。这样做有一个缺点，您不仅可能暴露敏感数据，比如嵌入到 JavaScript 的密码；还可能遇到现代浏览器控件的功能限制。

通过 SchemeHandler，您可以规避这些问题。首先，您必须在一个变量中生成一些 HTML，我们假设这个变量是 lcHtml 。

```
Local loConfig
loConfig = Createobject ("Empty")
AddProperty (m.loConfig, "index_html", m.lcHtml)
loBrowser.BindToHost (Thisform.Host, "https://MyApp/index.html", m.loConfig)
```

config 对象可以是任意的对象。您也可以将 Thisform 作为对象传递给您的表单，仅需保证在 Thisform 中拥有您需要的属性。您可以在 VCX 或 PRG 中创建一个 handler 类来处理请求。在本例中，我仅仅对静态内容有兴趣，所以，我使用了一个 Empty 对象。

CefSharpBrowser.prg 希望属性的名字和被请求的文件名一模一样。这时，属性名中的所有句点，都必须用下划线代替。因此，如果文件名是“index.html”，那么相应的属性名就会被命名为“ index_html”。

当请求这个文件时，将返回属性的内容。如果属性不存在，那么您将收到一条错误信息。当然，这还有改进的空间。

访问属性最快的方式是 GETPEM() 。它可以返回属性值。然而，CefSharpBrowser 使用了宏替换，这是因为 GETPEM() 不能用于 foxMock 的 mocked 对象。foxMock 使用 This_Access 方法来拦截所有方法和属性的调用。GETPEM() 不会触发 This_Access ，因此，永远也不会返回模拟的属性。

文件扩展名确定为请求返回的 MIME 类型。该列表当前状态是硬编码，默认值为：text/plain 。

编码（Encoding）这方面有一点小麻烦，因为我们在这里需要处理两种编码。每个请求都会返回一个字节流。您可以把它想象成您文件中的二进制数据。您必须要告诉 CefSharp 如何解释这个字节流，就像在 WEB 服务中那样。

CefSharpBrowser.prg 假设您传递的是 UTF-8 编码的文本，因为这是 WEB 开发的黄金标准。

如果您可以确保在将结果赋予 response 对象之前就调用了 STRCONV()，那么，此方法对于 HTML、文本或 JSON 来说，效果会非常好。对于 XML ，您则需要确保 XML 标头标记中的 XML 编码也设置为 UTF-8 。

对于像图像这样的二进制数据来说更加麻烦。理论上，您可以使用 MemoryStream 对象将二进制数据传递给 CefSharp。但是，对 .NET 代码的任何调用都会经过 COM InterOp ，并在其中进行参数转换。这意味着所有字符串都从 ANSI(或您的本地代码页)转换成了宽字符，也就是 16-bit Unicode 。处理这样的二进制数据，目前我还没有最终完成。

OnProcessRequesProperty 从 CefSharp 请求一个 UTF-8 编码的内存流，并将响应回写到资源处理程序。它还处理诸如状态码、Mime Type、Responselength 这样。

如果您处理的是静态内容，那么属性是最简单的方法。如果您要处理动态的，但是不依赖于浏览器中的任何参数，那么您仍旧可以通过添加一个带有 Access 方法的属性来解决。

但是，如果您需要将响应作为查询字符串参数或 POST 数据（甚至可能是 header 值）发送给您的数据，那么您需要一个方法而不是一个属性。每一个不以文件扩展名结尾的 URL 都会被视为一个方法。也就是说，https://MyApp/HandleRequest?param=value 会调用您传入 BindToHost 的 config 对象中的 HandleRequest 方法。

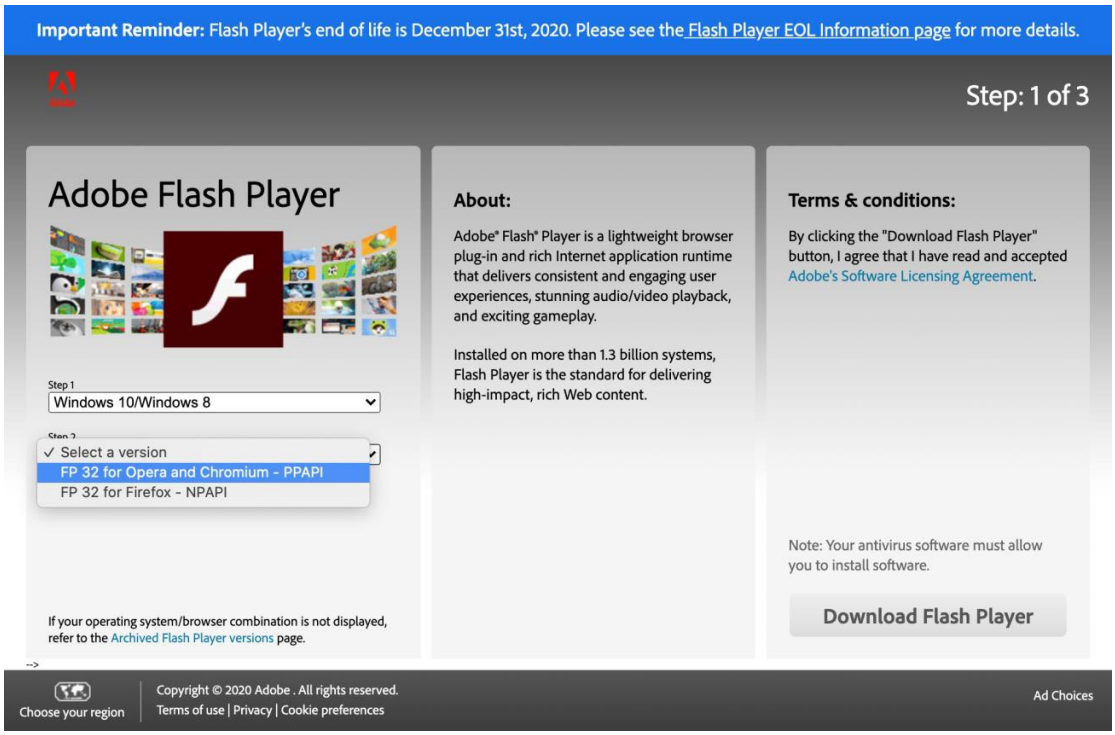
需要 Flash 的网站

幸运的是 Flash 仅仅是一个暂时的问题。Adobe 宣布将在 2020 年 12 月 31 停止对它的支持。这实际上也就意味着所有主要的浏览器供应商都将从它们的浏览器中删除 Flash 。微软宣布作为 Windows 更新的一部分将从其浏览器中删除 Flash 。Adobe 将从 2021 年 1 月 1 日起停止 Flash 的下载，并删除所有合法的下载链接。

您完全可以期待在 2021 年时，您无需再担心此问题。

但是，如果您仍要处理使用 Flash 的 WEB 应用程序，那么很快您就会发现 CefSharp 不支持 Flash 的开箱即用，这和它的大表兄 Chrome 有点儿不一样。

Chromium 支持 PPAPI，即 pepper API。您可以在 https://get.adobe.com/cn/flashplayer/otherversions/ 下载用于 Flash 的 pepper 插件（切记，必须在 2020 年 12 月 31 日之前）。



该插件可立即使用，但它需要您在您的应用程序中启用它。当您配置 CefSharp 的设置时，您必须启用 enable-system-flash 选项。更麻烦的是，默认情况下，它被设置为禁用。您需要首先删除默认的选项，然后再次添加它。传递“1”，可以启用该选项。

```
toBridge.InvokeMethod(m.loCefCommandLineArgs, "Remove", "enable-system-flash")
toBridge.InvokeMethod(m.loCefCommandLineArgs, "Add", "enable-system-flash", "1")
```

切记，对 CefSharp 的配置进行任何更改后，您必须重新启动 VFP !!!

部署与支持

系统需求

CefSharp 需要 .NET FrameWork 4.5.2 或更高版本。它很可能已经安装在用户的计算机上。如果没有安装，您可以从 Microsoft 网站下载 WEB 安装程序：https://www.microsoft.com/zh-CN/download/details.aspx?id=42643 。

CefSharp 是个内存大户。您的应用程序在使用 CefSharp 时会使用更多的内存。对桌

面应用程序来说，这通常不是问题，除非您的客户使用的是老旧的廉价硬件，其内存还达不到运行 Windows 的最低要求。

但是，在终端服务器环境，这也许是个问题。如果您的客户在服务器使用 WEB 浏览器时出现问题，并要求所有终端从各自的客户端计算机浏览，那么，您可能事先要和客户费一番唇舌。

如何在您的应用程序中分发 CefSharp?

如果您愿意，您可以创建您自己的 VS 项目，通过 nuget 获取 CefSharp，然后从 Release 文件夹中进行分发。如果您不使用任何的回调，那么，您也就只需要这些文件。

如果您依赖回调，那么分发 CefSharp 的最好方法就是使用 fpCefSharp 库。它包括 DLL 和它所编译的 CefSharp 版本。(译者注：作者似乎还没有释放 fpCefSharp 的分发版本)。

您可以把所有这些都放在一个专门为 CefSharp 准备的子文件夹中，或者将所有我呢见和文件夹添加到您的应用程序文件夹中。如果您需要添加新的功能，请确保您已经非常清晰关于 AssembyLoadContext 的问题。

性能问题

从技术上讲，CefSharp 和 Chrome 的性能相似。它们基于相同的内核，使用相同的优化技术，相同的消息传递，不同进程的相同结构，相同的渲染引擎 (Blink)，相同的 JavaScript 引擎 (V8)。

现实中，我们看到不同的用户在不同的电脑上的性能差异。我们在 FoxPro 应用中嵌入 catalog 应用，并与 catalog 进行交互，以获得用户的选择。对于一些用户来说，与在 Chrome 浏览器中运行相同应用程序相比，有时 VFP 应用程序中的这些，速度会更慢。我还没有找到原因所在。但是考虑到引擎是相同的，我想它与缓存或 Chrome 设置有关。

在大多数情况下，其性能与用户在 Chrome 浏览器中所体验的几乎完全一致。

特殊功能的可用性

webRTC（视频和音频录制）

众所周知，在桌面应用程序中很难完成的任务之一就是访问网络摄像头和麦克风。虽然这并不是完全做不到的，但是没有一个简单的方法。

几年前我在 Southwest Fox 上讨论过几种方法，那时是在讨论使用传感器。我建议的一种方法是使用支持 webRTC 的网络浏览器，在当时，这是相当前卫的。这种方法所面临的问题是 Internet Explorer 不支持 webRTC 。

而 CefSharp 支持摄像头。嗯，这不完全正确。CefSharp 所依赖的 CEF 支持摄像头。这好像也不完全正确。CEF 所基于的 Blink 引擎包含了支持摄像头和其他媒体设备的代码。您可以从 GitHub(https://github.com/chromium/chromium)克隆 Blink ，大小有 19 GB。

让我们来看几个简单的例子。https://webrtc.github.io/samples 是一个资源库，里面展示了如何在 JavaScript 中使用 webRTC 的示例。对我来说，这些示例也是测试浏览器功能和硬件的一种方法。如果有些东西在我的 WEB 应用中不工作而示例却可以，那我就有事情干了。但是，如果有些东西即使在示例中也失败了，那么问题就出在其他地方。

CefSharp 默认情况下是不启用媒体流的。如果您以默认配置运行，任何访问摄像头的行为都会被拒绝。这和浏览器不同，您也不会看到任何提示。显而易见，作为一个开发者，您需要代表您的客户在遵守当地法律的前提下做出选择。

当您在配置 CefSharp 时，请添加下面的代码:

```
toBridge.InvokeMethod (m.loCefCommandLineArgs, "Add", "enable-media-stream", "1")
```

切记，每个进程只能更改一次设置。如果您已经在 VFP 中使用 CefSharp 进行了操作，您必须先重新启动 VFP ，然后这些更改才会生效。这个问题我已经提了好几次了，因为我没有重启 VFP 而浪费了大量的时间来查找问题的根源。

第一个示例是 https://webrtc.github.io/samples/src/content/getusermedia/gum/ 。这是一个基本的“全部完成”的演示。摄像头和麦克风都已经启用。如果您看不到自己，那说明您的电脑存在硬件问题。

在点击“open camera”之前，请务必阅读关于耳机（headphones）的红色警告。

我们在 2015 年左右开始在 WEB 应用中使用 webRTC。当时，浏览器只支持其部分标准，您必须指定你的应用支持哪些平台上的哪些浏览器。从那之后，浏览器有了显著的改进，如今 webRTC 得到了广泛的支持。几乎所有支持浏览器中视频会议的应用程序的背后，包括在 HopIn 上本主题的演讲，都在使用它。

您是否想过，在没有安装 TearmViewer 等软件的情况下网络应用该如何支持屏幕共享？浏览器支持 webRTC 的媒体源之一就是您的桌面。

您可以通过共享屏幕的质量来判断一个应用程序是否在使用 webRTC 。

webRTC 讲屏幕转换为视频，当您移动窗口时，会导致文字模糊和拖影。当您在短时间内没有任何动作时，质量通常趋于稳定。

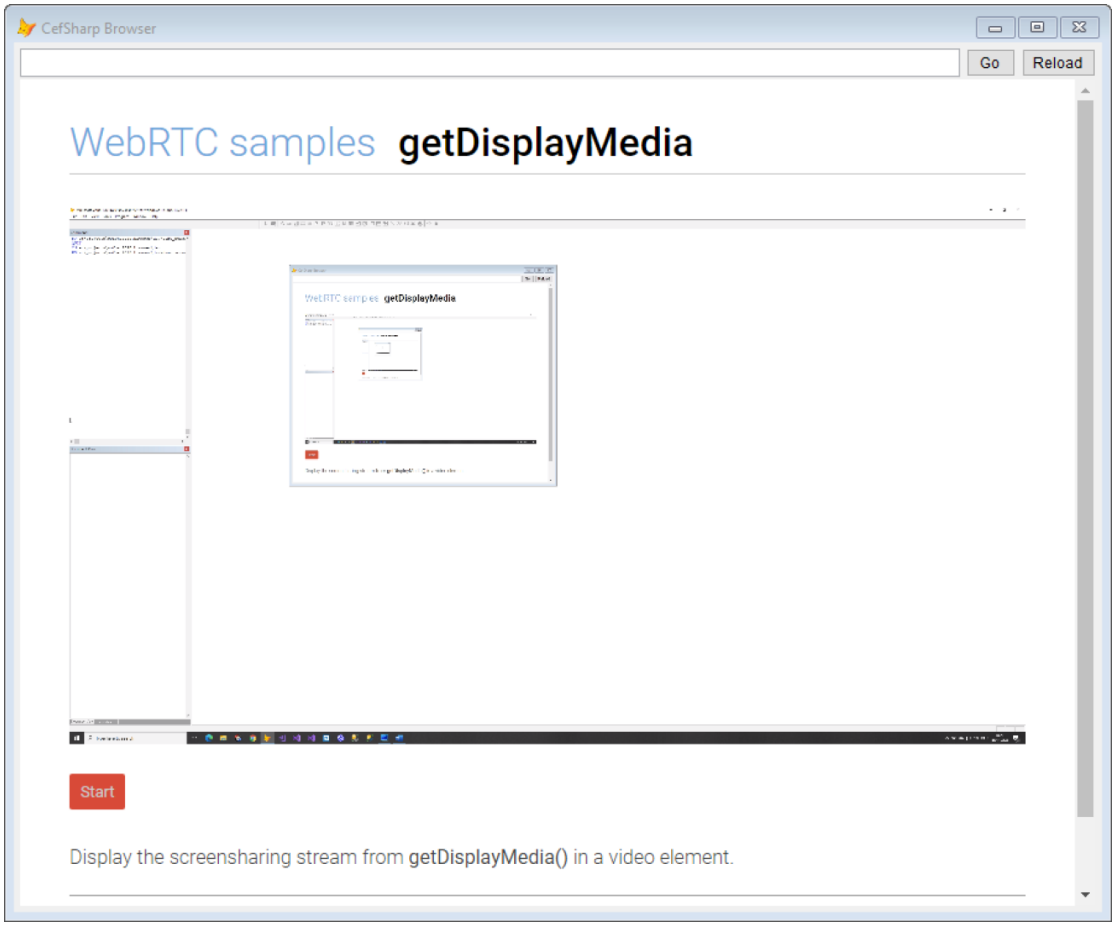
这与用于屏幕共享的协议(例如 RDP、Citrix|CA 或 TearmViewer 或 AnyDesk 等工具)完全不同。这些工具不会捕捉屏幕并将其转换为视频。它们会找出哪些区域已更改，并尝试传输这些区域的损失较小的图像。所以结果很清晰，在许多情况下就好像操作本地机器一样。

当您在浏览器中运行 https://webrtc.github.io/samples/src/content/getusermedia/getdisplaymedia/ 示例并单击开始按钮时，系统会提示您选择屏幕或窗口。在 Chrome 中，您甚至还可以共享 Chrome 标签。

```
toBridge.InvokeMethod (m.loCefCommandLineArgs, "Add", ;
    "--use-fake-ui-for-media-stream", "1")
```

名称有点诡异。它的作用是每当出现屏幕共享对话框时，CefSharp 都会为您选择“share my screen”选项。换句话说，截至目前，您不能共享一个应用程序窗口，也不能以编程方式选择一个特定的窗口。

运行示例时，您会看到浏览器窗口显示了您的屏幕！有一个浏览器窗口显示了您的屏幕！！有一个浏览器窗口显示了您的屏幕！！！我猜您已经兴奋不已。休息一下，在您的屏幕中移动浏览器窗口。



使用 webRTC 时，您需要记住一件事：webRTC 仅与 https 一起使用。您无法从本地磁盘加载页面并访问 webRTC。相反，您可以像之前一样伪造一个 https 链接。您从网页转发到自己远程服务器上的内容也必须加密。

如果您从未实施过应用内支持工具的原因是您不懂如何捕获视频、声音和屏幕，那么，现在，问题解决了。webRTC 非常适合获得用户的反馈。您既可以使用离线解决方案，将视频存储在文件中，并在适当的时间发送给您；或者通过自己的远程服务器来实施在线解决方案，就像其他应用程序和网站支持即时聊天和在线支持。

挡在您前进路上的唯一绊脚石就是一个非常复杂的 JavaScript API 。

定位服务

定位服务是我期待的功能之一。这取决于您的笔记本电脑是否嵌入 GPS 芯片。我的笔记本电脑上是没有，但是很多电脑都有。如果您在 Chrome 或 FireFox 打开谷歌地图，并点击右下角的小图标移动到您当前的位置。如果您看到自己所在位置的地图，则说明您的笔记本电脑具有 GPS 支持，或者谷歌知道您的 WIFI 。

不幸的是，事实证明，地理位置服务从 2018 年就从 CEF 中删除了，因为它们尚未经过测试且无法进行自动测试。由于 CEF 是 CefSharp 的基础，因此您也无法在 CefSharp 中访问它们。

对于在应用程序中进行地理位置的定位，您仍然依赖于我之前演讲的有关在 VFP 中使用传感器的 Windows API 。

查看 PDF 文档

许多地方已经使用 PDF 文件取代了印刷版的文件。发票、报价以及许多商业文件都通过电子邮件将 PDF 附件进行方便的发送。当涉及到生成 PDF 时，VFP 开发人员可以选择不同的工具。有时，您需要在您的应用程序中查看 PDF 文件而不是使用 PDF 默认的阅读器。

有一些 ActiveX 可以显示 PDF 文件。这些控件通常可以为您提供从如何显示到如何与 PDF 进行交互的广泛控制。但是，如果将它们与您的应用程序进行分发，可能是非常的昂贵。

使用 General 字段，您可以使用安装的 COM 兼容的查看器来显示 PDF。不过，这并不能给您带来一致的体验。因为这取决于安装了什么软件，这个方法对有些客户有效，而对

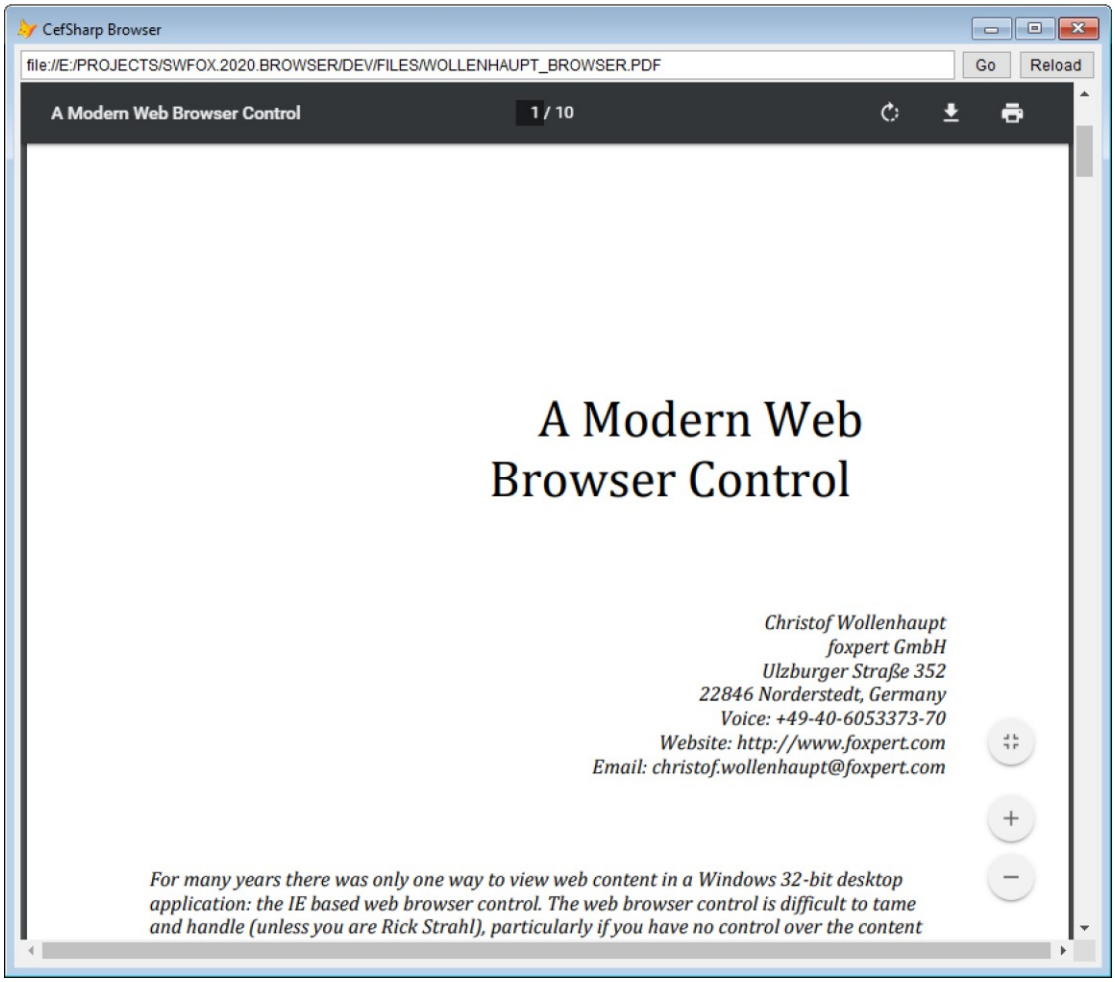
另一些客户无效。

CefSharp 内置了一个 PDF 查看器。如果您导航至一个 PDF 文件，那么在浏览器窗口中您就可以查看它。

如果您使用 Chrome 浏览器，您会觉得很眼熟。浏览器会用 PDF 文件填充所有的区域。如果您把鼠标放在浏览器窗口，会出现一些弹出式控件。

顶部的工具栏可以进行页面跳转、旋转、下载或打印。在右下角，您可以控制它的所放级别。

您无法改变这些控件的行为，也不能添加自己的工具栏按钮。CefSharp 的打印功能与 Chrome 有所不同。您通常是直接进入系统打印对话框而不是谷歌云打印对话框。



浏览器控件的新用途

在现代软件开发中，客户端应用程序的概念正在逐渐消失。当您与一个比 VFP 还年轻

的开发人员交谈时，有两种类型的开发者：前端开发和后端开发。前端开发人员编写浏览器中运行的代码，后端开发人员编写 REST 服务。您需要三个工具，一个浏览器、一个控制台和一个编辑器，控制台和浏览器偶尔可以忽略。

浏览器已经变成运行在操作系统之上的操作系统。它们的用户界面不适用本地控件。它们对新的 Unicode 的支持往往比底层操作系统更完整设置更新。它们支持相当多的硬件，而且，跨平台。

摄像头

我们已经看到 webRTC 如何使用以及如何访问内置的摄像头。如果要实现摄像头监控，那么 WEB 浏览器中的 UI 就很有用。但是，你可以不这样做。无需太多代码就可获取监控内容的截图并将其保存为图片。

将图片合并为一个 PDF 文件

在一个应用程序中，我们支持 WIA 和 TWAIN 扫描仪。现代的扫描仪可以通过其进纸器扫描多页文档，人们不会希望得到多张单独的图片。他们希望得到一个类似于 PDF 那样的。我们采用的一个方法就是生成一个 HTML 文档，该文档将所有这些图片与适当的 CSS 嵌入到一起，然后将每个图片打印在单独的页面上。

使用 CefSharp ，您可以将 HTML 文档转换为 PDF 文件，这样，您就可以查看两种格式的图片。

支持 Unicode 的文本框

VFP 可以在一个表单上处理多个代码页或语言。但这非常的乏味，而且有很多限制。如果要合并真正的 Unicode 输入，则需要使用 ActiveX 控件。您可以使用带有文本框或文本区域的 HTML 页面作为输入控件，而不是使用过时的 VB 控件。

应用程序的自定义

当您允许用户可以自己定制应用程序时，通常使用 FoxPro 脚本。毕竟，宏替换、EVALUATE() 和 EXECSCRIPT() 可以执行任意的 FoxPro 代码（从简单到复杂）。

不过，这并不是唯一的选择。从安全角度来看，这实在是再糟糕不过的事。代码在您的应用程序相同的环境中运行。你无法对齐沙箱化，无法对其进行控制，也没有任何办法进行防护。此代码可以访问 EXE 中嵌入的任何文件，包括 VCX 中的源码。它可以绕过您设置的所有安全边界。它可以访问任意的表、任何文件、任何 SQL 连接，以及您的应用程序有权访问的任何其他资源。

此外，还有一个原因，只有当您想更进一步时才能发现。如果您允许客户编写代码，那么，这也将是您将来需要进行支持的部分。将您的应用程序移动到 WEB 服务器或基于 .NET 或 JavaScript 的另一个平台，您突然发现必须执行用 VPF 编写的脚本。

这里有一个选择。如果您的平台依然是 Intel CPU 上的 Windows，则可以创建一个 COM 服务，或者，您嵌入像 Guineu 这样的库，或者使用 XSharp 编译器，但，哪一个都不是件容易的事。

从客户的角度出发，FoxPro 也不是理想的选择。没有多少 IT 顾问可以帮助他们自定义您的应用程序。客户内部可能就没人知道 VFP。另一方面，JavaScript 如日中天。

那么如何使用 JavaScript 来自定义程序呢?

首先，您可以轻松生成带有一些额外代码的 JavaScript 文件，他们用以设置所需的环境。将其嵌入到 HTML，并在页面加载时调用它们。浏览器未必呈现出来。这只是您代码的执行环境。任何输入参数都可以直接在所生成的代码中进行赋值。

当需要与您的应用程序交互时，您可以创建 JavaScript 封装类，这些类调用 config 对象提供的 REST API。它比简单的调用 EXECSCRIPT() 更为复杂，但是作为回报，您提供了已定义的接口，并限制了自定义代码不能是恶意的。

托管所见即所得的编辑器

在许多应用程序中，用户喜欢格式化的输入。但是 VFP 中的选项非常有限。本机不支持格式化文本输入，微软 RichText 控件以及很长时间没有更新了。即使您提供格式化的输入，但是，大多数客户的需求比您能实现的要多的多，例如拼写检查、自动更正、文本完成甚至建议。

WEB 应用程序已倾向于提供这些功能。难以基于 WEB 编辑器的时代早已经过去。限制，您当然可以直接从剪贴板插入带格式的文本或图像，无需借助某些上传对话框。而且，可以在网络浏览器和本机应用程序间进行拖放操作。

JavaScript 富文本编辑器已经发展了很长一段时间，并提供了比富文本控件更多的功能，看上去也更加的流畅和现代。您可以有很多种选择，有些是免费的，有些是收费的。

Jeferson Mari 在 Github 上有一个各种编辑器的列表：

https://github.com/JefMari/awesome-wysiwyg

如果您使用 Slack，那么您已经在处理 Quill 富文本编辑器的自定义版本了：

https://quilljs.com/

JavaScript 控件

过去，如果您去询问客户对软件的预期以及工作方式，他们所描述的可能指向 Microsoft Office 产品，无论是 Outlook 还是 Excel 。

时代已经变了。如今，您的客户更有可能引用网络产品或应用。可能是一个商业应用，也可能是 Instagram 或 Facebook 之类的。

使用 JavaScript 和正确的库，您可以创建外观绚丽的 WEB 应用程序。人们经常在业务时间利用网络来工作。WEB 应用程序，就意味着“时髦”。

使用 CefSharp ，您可以在 VFP 应用程序中使用这些控件，而不必立即替换整个应用程序。需要很棒的仪表板吗？有！需要绘制带有动画效果的图标吗？有！需要日历控件吗？有！需要 3D 渲染框架、动画库、电子表格、图表、布局引擎、图像编辑组件、数据可视化工具…吗？统统都有！

它们中的大多数仅仅需要一点而服务端的支持，您可以使用 CefSharp 中的 ResourceHandler 轻松应对。

总结

CefSharp 不仅仅是显示网页或使客户可以访问在线应用。使用 CefSharp ，您可以提供其他所有人提供的相同功能，使用和其他人一样的工具集，并拥有与其他人一样的 UI。

您无需替换整个应用程序即可上手。您不必迁移到云中。您可以从某些个表单入手，并仍在您喜欢的计算机语言中工作，同时，还能熟悉 JavaScript 、HTML、CSS。