VCL

TMS TAdvWebBrowser DEVELOPERS GUIDE

Sep 2020

Index

Getting Started

Included in the TMS VCL UI Pack is a WebBrowser that can display web pages, HTML and load files such as PDF files. The WebBrowser also allows executing scripts and catch the result in a callback.

To get started with the WebBrowser, add the AdvWebBrowser unit. The WebBrowser class is called TAdvWebBrowser. TAdvWebBrowser supports Edge Chromium. Please follow the instructions below to correct install Edge Chromium on your Windows operating system.

Installation

TAdvWebBrowser is part of the TMS VCL UI Pack and supports Delphi or C++Builder XE7 and newer versions. Before the TAdvWebBrowser can be used there are a couple of things that need to be done. Below are the steps to take when you want to use the TAdvWebBrowser and enable Edge Chromium on your Windows operating system.

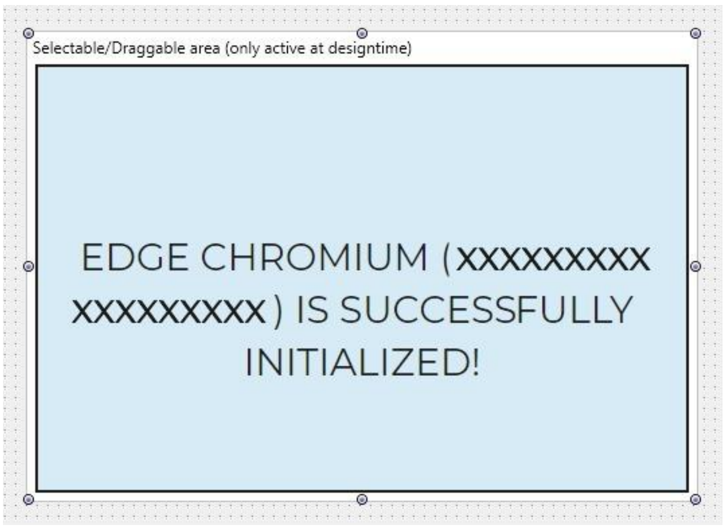
1) Windows 10 with automatic updates enabled has already Edge Chromium included that is used by TAdvWebBrowser.

   If you do not have automatic updates enabled or use an older version of Windows, install Edge Chromium from the following page: https://www.microsoft.com/en-us/edge

   We have tested the installation against v85.0.534.0. Earlier versions are not supported. Newer version updates need to be tested first, as each update might potentially break your application. Please before installing, check the version number and ask us for an update in case you are having troubles getting the browser to run. Microsoft will also push out Edge Chromium through Windows Updates.

2) Make sure the WebView2Loader_x86.dll and WebView2Loader_x64.dll are copied under System32 and SysWow64 folders. The dlls can be found after installation in the source directory under the folder "Edge Support". Please note that these dlls are also necessary when deploying your application!

3) Start the IDE and drop an instance of TAdvWebBrowser on the form. The border around the webbrowser at designtime is for moving/selecting it. The blue box indicating the Edge Chromium is initialized, is interactable and is a live browser instance. You should see the following when the browser is successfully initialized:


Selectable/Draggable area (only active at designtime)
EDGE CHROMIUM (XXXXXXXXX XXXXXXXXX) IS SUCCESSFULLY INITIALIZED!

Navigating to a URL

After creating an instance of the TAdvWebBrowser, navigating to a web page is as simple as using the code below.

```
procedure TForm1.Browse;
begin
  AdvWebBrowser1.URL := ‘https://www.tmssoftware.com’;
end;
```

or

```
procedure TForm1.Browse;
begin
  AdvWebBrowser1.Navigate(‘https://www.tmssoftware.com’);
end;
```

Loading HTML

Loading fully functional HTML/JavaScript can be done with the following code:

```
procedure TForm1.LoadFromHTML;
begin
  AdvWebBrowser1.LoadHTML(‘<b>This is HTML!</b>’);
end;
```

Loading files

Files such as PDF files, images, HTML files and many more can be loaded with the LoadFile method:

```
procedure TForm1.LoadFromFile;
begin
  AdvWebBrowser1.LoadFile(‘MyPDF.pdf’);
end;
```

Capture Screenshot

Capturing a screenshot of the current view of the TAdvWebBrowser is as easy as calling AdvWebBrowser1.CaptureScreenshot. The asynchronous event OnCaptureScreenshot is being called as soon as the screenshot is ready. The OnCaptureScreenShot event has a parameter AScreenShot of the TAdvBitmap type.

Communicating with the application through a JavaScript bridge

Communication with the application is possible through registration of a JavaScript bridge object. The object needs to conform certain number of parameters to function properly. The definition of the bridge object is shown below:

```
  TMyBridgeObject = class(TInterfacedPersistent, IAdvCustomWebBrowserBridge)
  private
    FObjectMessage: string;
    function GetObjectMessage: string;
    procedure SetObjectMessage(const Value: string);
  published
    property ObjectMessage: string read GetObjectMessage write SetObjectMessage;
  end;
```

Notice that the ObjectMessage property is set to published, so internal RTTI can pick up the property and use this to communicate with the application. The IAdvCustomWebBrowserBridge interface is used to make sure the object is picked up in mobile environments, as the communication process is slightly different there. The JavaScript part that is required in HTML is shown below.

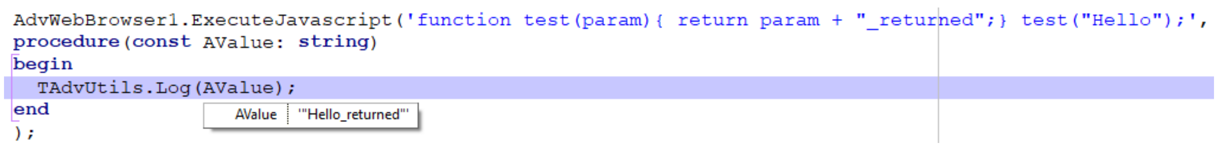
```
procedure TForm1.FormCreate(Sender: TObject);
const
  BridgeName = 'MyBridge';
var
  w: TAdvWebBrowser;
  o: TMyBridgeObject;
  sHTML: string;
begin
  w := TAdvWebBrowser.Create(Self);

  sHTML :=
    '<html>' + #13 +
    '  <head>' + #13 +
    '    <script>' + #13 +
        w.GetBridgeCommunicationLayer(BridgeName) +
    '    </script>' + #13 +
    '  </head>' + #13 +
    '  <body>' + #13 +
    '    <button onclick="send' + BridgeName + 'ObjectMessage("Hello World!");">Click Me!</button>' +
#13 +
    '  </body>' + #13 +
    '</html>';

  w.Parent := Self;

  o := TMyBridgeObject.Create;
  w.AddBridge(BridgeName, o);
  w.LoadHTML(sHTML);
end;
```

First, we need to create a webbrowser instance, create our bridge object and pass it to the webbrowser. The ObjectMessage propery naming is important and needs to remain the same. The HTML code snippet contains the helper function to setup communication between browser and application. As seen in the onclick event of the button, the function is called with a string value as a parameter. Communication between application and browser always happens with a string value.

Executing JavaScript

Executing JavaScript is supported and can also be used in combination with a return value callback. Below is a sample code snippet that shows how to execute JavaScript and get a return value.

```
  AdvWebBrowser1.ExecuteJavascript('function test(param){ return param + "_returned";}
test("Hello");',
  procedure(const AValue: string)
  begin
    TAdvUtils.Log(AValue);
  end
  );
```



```
AdvWebBrowser1.ExecuteJavascript('function test(param){ return param + "_returned";} test("Hello");',
procedure(const AValue: string)
begin
  TAdvUtils.Log(AValue);
end
);
```

Using Events

The TAdvWebBrowser exposes 2 important events: OnBeforeNavigate and OnNavigateComplete. It allows you as a developer to retrieve to which page/URL the webbrowser is navigating to and also allows you to block navigation. Below is a sample that blocks access to a certain page within https://www.tmssoftware.com.

```
procedure TForm1.AdvWebBrowser1BeforeNavigate(Sender: TObject;
  var Params: TAdvCustomWebBrowserBeforeNavigateParams);
begin
  Params.Cancel := Params.URL.Contains('tmsfnccore.asp');
end;
```