TMS FNC WebBrowser
DEVELOPERS GUIDE

September 2020
Copyright © 2020 by tmssoftware.com bvba
Web: http://www.tmssoftware.com
Email: info@tmssoftware.com

Index

Getting Started

Included in the TMS FNC Core 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 FMX.TMSFNCWebBrowser, VCL. TMSFNCWebBrowser, LCLTMSFNCWebBrowser or WEBLib.TMSFNCWebBrowser depending on the chosen framework. The WebBrowser class is called TTMSFNCWebBrowser, and the code is shareable between the four supported frameworks. (In TMS WEB Core, the TTMSFNCWebBrowser is a DIV that can display HTML, navigate functionality is not supported.

Installation

Before the TTMSFNCWebBrowser can be used there are a couple of things that need to be done depending on the chosen platform. Below are the steps to take when you want to use the TTMSFNCWebBrowser for each platform/operating system separately.

iOS/macOS

iOS/macOS is no longer supporting UIWebView/WebView classes and has switched to WKWebView. This was not an issue previously, but Apple has recently created a document that is stating to: “no longer accepting applications that uses the UIWebView/WebView classes starting from April 2020”. (https://developer.apple.com/news/?id=12232019b). The TTMSFNCWebBrowser has changed switched internally to WKWebView to accommodate to these changes. It requires the WebKit framework. Adding frameworks to your iOS/macOS SDK inside IDE is explained at the following page: https://www.tmssoftware.com/site/frameworks.asp

Android

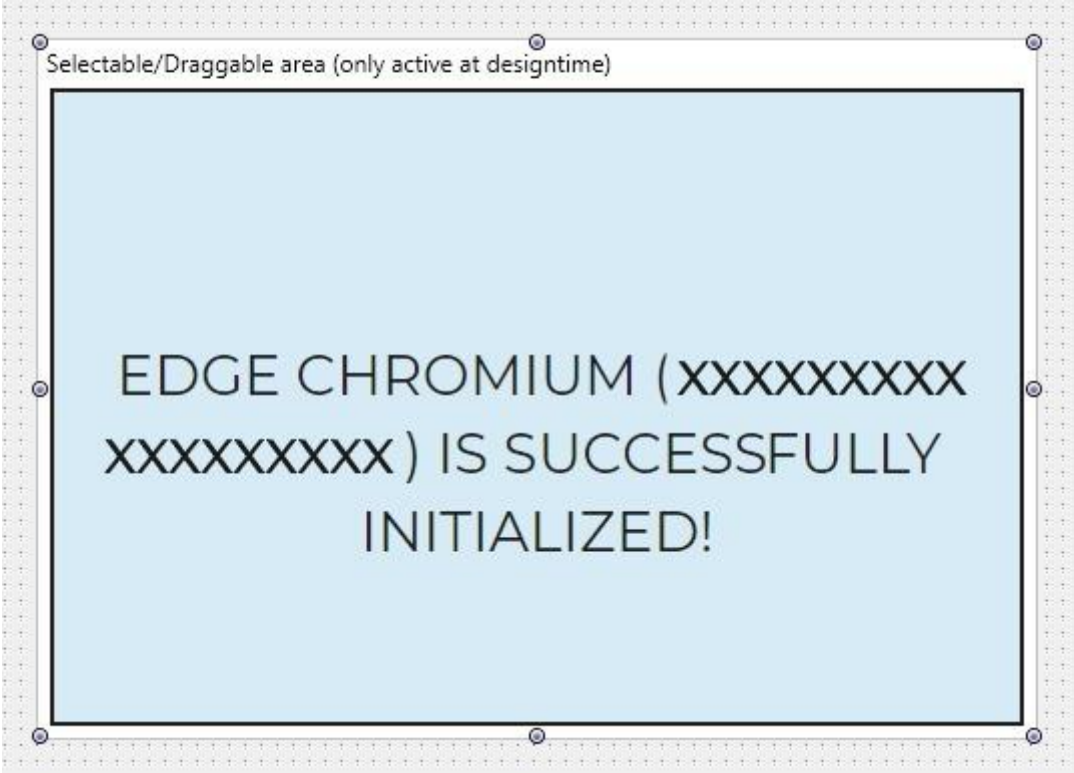
No additional steps needed for basic browser functionality. In case you encounter errors of missing libraries during deployment, please follow instructions at the following page: https://www.tmssoftware.com/site/androidjs.asp

Windows (Edge Chromium)

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 in to the System32 and SysWow64 folders. The dlls can be found after installation in the source directory in 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 TTMSFNCWebBrowser 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 TTMSFNCWebBrowser, navigating to a web page is as simple as using the code below.

```
procedure TForm1.Browse;
begin
  TMSFNCWebBrowser1.URL := ‘https://www.tmssoftware.com’;
end;
```

or

```
procedure TForm1.Browse;
begin
  TMSFNCWebBrowser1.Navigate(‘https://www.tmssoftware.com’);
end;
```

Loading HTML

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

```
procedure TForm1.LoadFromHTML;
begin
  TMSFNCWebBrowser1.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
  TMSFNCWebBrowser1.LoadFile(‘MyPDF.pdf’);
end;
```

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.

```
  TMSFNCWebBrowser1.ExecuteJavascript('function test(param){ return param + "_returned";}
test("Hello");',
  procedure(const AValue: string)
  begin
    TTMSFNCUtils.Log(AValue);
  end
  );
```

```
TMSFNCWebBrowser1.ExecuteJavascript('function test(param){ return param + "_returned";} test("Hello");',
procedure(const AValue: string)
begin
  TTMSFNCUtils.Log(AValue);
end
);
```

AValue '"Hello_returned"'

Capture Screenshot

Capturing a screenshot of the current view of the TTMSFNCWebBrowser is as easy as calling TMSFNCWebBrowser1.CaptureScreenshot. The asynchronous event OnCaptureScreenshot is being called as soon as the screenshot is ready. The OnCaptureScreenShot event has a parameter AScreenShot of the TTMSFNCBitmap 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, ITMSFNCCustomWebBrowserBridge)
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 ITMSFNCCustomWebBrowserBridge 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: TTMSFNCWebBrowser;
  o: TMyBridgeObject;
  sHTML: string;
begin
  w := TTMSFNCWebBrowser.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.

Using Events

The TTMSFNCWebBrowser 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.TMSFNCWebBrowser1BeforeNavigate(Sender: TObject;
  var Params: TTMSFNCCustomWebBrowserBeforeNavigateParams);
begin
  Params.Cancel := Params.URL.Contains('tmsfnccore.asp');
end;
```