TMS TAdvWebBrowser DEVELOPERS GUIDE

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.

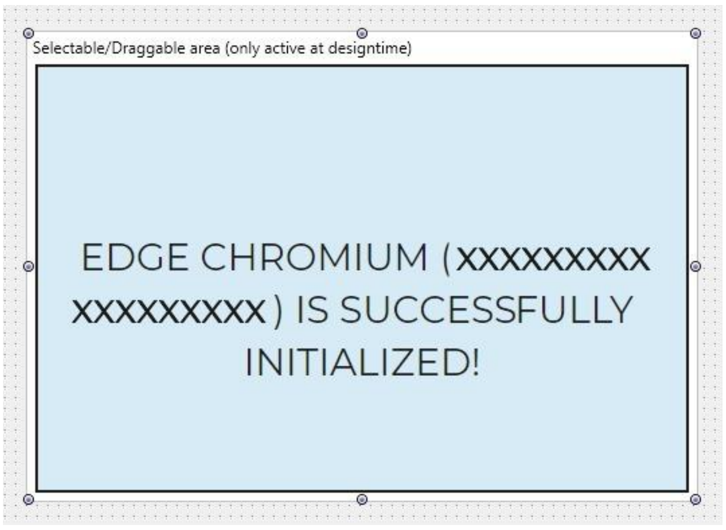
1) Windows 10 with automatic updates enabled has already Edge Chromium included that is used by TAdvWebBrowser.

   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;
```

It is also possible to navigate with a custom method and data with the **NavigateWithData** function.

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.

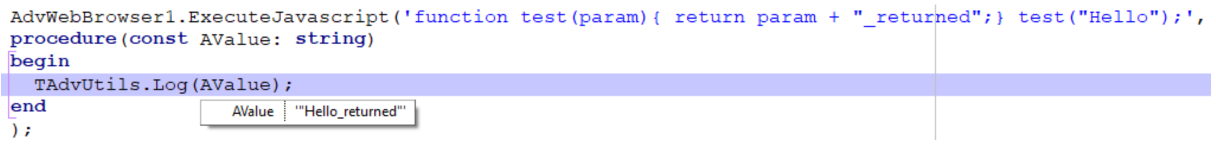
```
procedure TForm1.FormCreate(Sender: TObject);
const
  BridgeName = 'MyBridge';
var
  w: TAdvWebBrowser;
  o: TMyBridgeObject;
  sHTML: string;
begin
  w := TAdvWebBrowser.Create(Self);

  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;
```

Manipulating the Context Menu

It is possible to change the context menu that is shown in the web browser.

- By default it will show the menu that is generated by the browser.

- You can change it to a framework specific menu by assigning a `TPopupMenu` to the EdgeWebBrowser's **PopupMenu** property.

- It is also possible to have this `TPopupMenu` rendered as a native context menu by enabling the **UsePopupMenuAsContextMenu** property in `Settings`. The assigned events or actions when a click happens, will still be triggered.

- You can fully customize the native context menu with the use of the event **OnGetContextMenu**. This event is triggered when the context menu is rendered. It has a target item **TTMSFNCWebBrowserTargetItem** that shows whether it is shown on a *Page, Image, Selected Text, Audio or Video.*

  And you can check if there is a link or selected text attached to it. Based on this you can now choose which items to show in the context menu. In this event there is a list of **TTMSFNCWebBrowserContextMenuItem**. You will notice that it is not possible to get any information, because the system generated items are protected. Only custom items can be changed. To help you with this the **AsSystem** and **AsCustom** functions are available. You can delete system items from the list. But the position can't be changed. **TTMSFNCWebBrowserCustomContextMenuItem** items can be created and inserted in the list.

It is **only** possible to **detect** if an item was clicked with **custom items**.
If you want to have full control over the context menu, we advise to use a popup menu or recreate all of the items.

```
procedure TForm.EdgeWebBrowserGetContextMenu(Sender: TObject; ATarget:
TTMSFNCWebBrowserTargetItem; AContextMenu:
TObjectList<TTMSFNCWebBrowserContextMenuItem>);
var
  I: Integer;
  mi: TTMSFNCWebBrowserCustomContextMenuItem;
begin
  if ATarget.Kind = mtSelectedText then
    SelectionEdit.Text := ATarget.SelectionText;

  //Make sure the customer can't reload the page from the context menu
  while I < AContextMenu.Count do
  begin
    if AContextMenu[I].AsSystem.Name.ToLower <> 'reload' then
      AContextMenu.Delete(I);
    Inc(I);
  end;

  mi := TTMSFNCWebBrowserCustomContextMenuItem.Create;
  mi.Name := 'Go to TMS';

  AContextMenu.Insert(0, mi);
end;

procedure TForm.TMSFNCEdgeWebBrowserCustomContextMenuItemSelected(Sender: TObject;
ASelectedItem: TTMSFNCWebBrowserCustomContextMenuItem);
var
  s: string;
begin
  if ASelectedItem.AsCustom.Name = 'Go to TMS' then
  begin
    TMSFNCEdgeWebBrowser.Navigate('https://www.tmssoftware.com');
  end;
end;
```

DevTools Methods and Events

The Chrome DevTools Protocol provides APIs to instrument, inspect, debug, and profile Chromium-based browsers. The Chrome DevTools Protocol is the foundation for the Microsoft Edge DevTools. Use the Chrome DevTools Protocol for features that aren't implemented in the TMSFNCEdgeWebBrowser.
More information can be found on https://chromedevtools.github.io/devtools-protocol/

- Methods can be called with **CallDevToolsProtocolMethod** and the response to it is retrieved with **OnDevToolsMethodCompleted** as a json string.
- Events should be registered first with **SubscribeDevtools** (the enable method is called by default). After registration the events trigger the **OnDevToolsSubscribedEvent**.
- By default the events **Log.entryAdded** and **Runtime.consoleAPICalled** are already parsed in **OnGetConsoleMessage**.

Properties

Property name	**Description**
Settings	Object of TTMSFNCWebBrowserSettings to configure your web browser in design time.
URL	The address of the current webpage of the browser.
Downloads	List of all the downloads that were starting since initialization.

Settings

Property name	**Description**
EnableAcceleratorKeys	Boolean that indicates whether all accelerator keys that access features specific to a web browser are enabled.
EnableContextMenu	Boolean that indicates whether it is possible to show the context menu.
EnableShowDebugConsole	Boolean that indicates whether it is possible to show the debug console.
UsePopupMenuAsContextMenu	Boolean that indicates whether an assigned popup menu should be shown as the native context menu.

Methods

Method name	**Description**
OpenTaskManager	Opens the Browser Task Manager view as a new window in the foreground.
GetCookies(*AURI: string*)	Gets a list of cookies, if a URI was spacified then it will only show those that match the specific URI.
AddCookie(ACookie: TTMSFNCWebBrowserCookie)	Adds or updates a cookie with the given cookie data; may overwrite cookies with matching name, domain, and path if they exist.
DeleteAllCookies	Deletes all cookies under the same profile.
DeleteCookie(AName, ADomain, APath: string)	Deletes a cookie whose name and domain/path pair match those of the specified cookie.
ShowPrintUI(ACookie: TTMSFNCWebBrowserCookie)	Opens the print dialog to print the current web page.
InitialPrintSettings	Returns TTMSFNCWebBrowserPrintSettings with default values.
Print(*APrintSettings: TTMSFNCWebBrowserPrintSettings*)	Print the current web page asynchronously to the specified printer with the default or provided settings.
PrintToPDFStream(*APrintSettings: TTMSFNCWebBrowserPrintSettings*)	Provides the PDF data of current web page asynchronously for the default or provided settings.
PrintToPDF(AFileName: string; *APrintSettings: TTMSFNCWebBrowserPrintSettings*)	Print the current page to PDF asynchronously with the default or provided settings.
NavigateWithData(AURI: string; AMethod: string; ABody: string; *AHeaders: TStrings*)	Navigates to an URI with a custom method and data as text.

Method name	Description
NavigateWithData(AURI: string; AMethod: string; ABodyStream: TStream; *AHeaders: TStrings*)	Navigates to an URI with a custom method and data as a stream.
CallDevToolsProtocolMethod(AMethodName: string; AParametersAsJSON: string)	Runs an asynchronous DevToolsProtocol method.
SubscribeDevtools(AEventName: string)	Subscribe to an event via the name to receive them in **OnDevToolsSubscribedEvent**.

Events

Event name	Description
OnGetCookies	Returns an array of TTMSFNCWebBrowserCookie after retrieving the cookies from the method GetCookies.
OnPrinted	Returns a boolean to indicate if the method Print was successful.
OnPrintedToPDF	Returns a boolean to indicate if the method PrintToPDF was successful.
OnGetPrintPDFStream	Returns a TMemoryStream after converting the webpage to a PDF with the method PrintToPDFStream.
OnGetContextMenu	Triggered when the context menu is shown. You can check what to handle with TTMSFNCWebBrowserTargetItem and you can manipulate the items with the list of TTMSFNCWebBrowserContextMenuItem
OnGetPopupMenuForContextMenu	Choose the popup menu to be used in the context menu.
OnCustomContextMenuItemSelected	Returns a TTMSFNCWebBrowserCustomContextMenuItem when a custom item was selected in the context menu.

Event name	**Description**
OnDevToolsMethodCompleted	Returns the json response of the **CallDevToolsProtocolMethod**.
OnDevToolsSubscribedEvent	Returns the json message of the registered events with **SubscribeDevtools**.
OnGetConsoleMessage	Triggered when there are new entries in the console.
OnDownloadStarted	Triggered when a new download will be started. You can set the result path and choose if the download should be silent, paused or cancelled.
OnDownloadStateChanged	Triggered when the state of a download was changed.
OnDownloadBytesReceivedChanged	Periodically triggered with an update of the downloads progress.