How to call JavaScript functions in a TWebBrowser from Delphi

Contents

Introduction

On some occasions when using a *TWebBrowser* I've needed to use Delphi to call JavaScript functions contained in the current document.

This is quite easy to do. We'll first examine the techniques then we'll look at a case study that changes the font in an HTML document.

Finally, thanks to contributions from Christian Sciberras, we will look at how to get the return value from a JavaScript function called from Delphi.

Overview of Solution

The window object associated with an HTML document exposes a method – *execScript* – that enables JavaScript to be called. The first parameter of this method takes a string containing the required function call (complete with actual parameters). The method's second parameter specifies the script language being used – in our case `'JavaScript'`.

So, how do we get hold of the window object that exposes the required method? We simply use the *parentWindow* property of the web browser control's document object. We need to ensure that a document object is available by first loading the required document into the web browser and waiting for it to finish loading.

Implementing the Solution

Let us assume we have a *TWebBrowser* control on a form into which an HTML document has been loaded. Assume also that the HTML document defines a JavaScript function named *foo()* that takes a string and an integer parameter. Assuming the document is fully loaded, we can call *foo()* with a method similar to this:

```
uses
  MSHTML;

procedure TForm1.CallFoo(S: string; I: Integer);
  { Calls JavaScript foo() function }
var
  Doc: IHTMLDocument2;       // current HTML document
  HTMLWindow: IHTMLWindow2; // parent window of current HTML document
  JSFn: string;              // stores JavaScipt function call
begin
  // Get reference to current document
  Doc := WebBrowser1.Document as IHTMLDocument2;
```

```
  if not Assigned(Doc) then
    Exit;
  // Get parent window of current document
  HTMLWindow := Doc.parentWindow;
  if not Assigned(HTMLWindow) then
    Exit;
  // Run JavaScript
  try
    JSFn := Format('foo("%s",%d)', [S, I]);  // build function call
    HTMLWindow.execScript(JSFn, 'JavaScript'); // execute function
  except
    // handle exception in case JavaScript fails to run
  end;
end;
```

Listing 1

Let's look at what's going on here. We first check that a document is available and store a reference to in *Doc*. We then attempt to get a reference to the document's parent window object and store it in *HTMLWindow*. Once we have a valid window object we call its *execScript* method, passing the call to *foo()*. The parameters to *foo()* must be literal values – we can't pass variables. String literals must be enclosed by double or single quotes.

Escaping quotes in string literal parameters

When passing string literal parameters to JavaScript functions you need to be careful to *escape* any quote characters contained in the string, otherwise the quote will terminate the string prematurely. Since JavaScript can use either single or double quotes to delimit literal strings you may need to escape either of these types of quote by preceeding it with a backslash.

As an example, suppose we need to pass the string `He didn't say "hello"` to a JavaScript function. If we delimit the string with double quotes we pass the it as `"He didn't say \"hello\""`.

Alternatively we may delimit the string with single quotes and pass it as `'He didn\'t say "hello"'`.

If the JavaScript function contains errors or doesn't exist an exception will be raised. We may wish to handle such exceptions before returning.

Case Study

In this case study we will develop a small application that displays an HTML document in a browser window. The document will contain a Javascript function – *SetFont()* – that can change document's default font. The application will display a combo box containing a list of all installed screen fonts. When the user selects a font from the combo box the font used in the web browser will be changed accordingly. We do this by calling *SetFont()* from Delphi.

Firstly, create an HTML document with any content you choose and include the following JavaScript function in the document's `<head>` section:

```
<script type="text/javascript">
  function SetFont(fontname) {
    document.body.style.fontFamily = fontname;
  }
</script>
```

Listing 2

This is the code that changes the font.

Now create a new delphi application and drop a *TComboBox* and a *TWebBrowser* on the main form. We will load the HTML document when the form is first shown. We will also load the screen fonts into the combo box at the same time. To do this create an *OnShow* event handler for the form with the following code:

```
procedure TForm1.FormShow(Sender: TObject);
  { Setup combo box and load document }
begin
  // Store screen fonts in combo box and disabled it
  ComboBox1.Items.Assign(Screen.Fonts);
  ComboBox1.Enabled := False;
```

```
  // Load the HTML page
  WebBrowser1.Navigate(ExtractFilePath(ParamStr(0)) + 'Test.html');
end;
```

Listing 3

Note that we disabled the combo box to prevent a font from being selected. We do this because to set a font we need to access the browser's *Document* object – and this object is only available when the HTML document is fully loaded. *TWebBrowser* triggers an *OnDocumentComplete* event when the document has finished loading. Therefore we handle that event and enable the combo box:

```
procedure TForm1.WebBrowser1DocumentComplete(Sender: TObject;
  const pDisp: IDispatch; var URL: OleVariant);
  { Document loaded: enable combo box }
begin
  ComboBox1.Enabled := True;
end;
```

Listing 4

We now come to the code that actually calls the JavaScript function. When the user selects a new font the combo box triggers its *OnChange* event. We handle this event by calling the JavaScript *SetFont()* function with the name of the selected font, as follows:

```
procedure TForm1.ComboBox1Change(Sender: TObject);
  { Make browser use selected font }
var
  Doc: IHTMLDocument2;       // current HTML document
  HTMLWindow: IHTMLWindow2; // parent window of current HTML document
  JSFn: string;              // stores JavaScipt function call
begin
  // Get reference to current document
  Doc := WebBrowser1.Document as IHTMLDocument2;
  if not Assigned(Doc) then
    Exit;
  // Get parent window of current document
  HTMLWindow := Doc.parentWindow;
  if not Assigned(HTMLWindow) then
    Exit;
  // Run JavaScript
  try
    JSFn := 'SetFont(''' + ComboBox1.Text + ''')';
    HTMLWindow.execScript(JSFn, 'JavaScript');
  except
    // handle exception in case JavaScript fails to run
    ShowMessage('Error running JavaScript');
  end;
end;
```

Listing 5

This method is very similar to that described in the previous section. *SetFont()* is called with the name of the font selected in the combo box. If an exception is raised when the JavaScript is called we display a message.

We have now developed all the code. All that remains is to add the following uses clause to the implementation section of the unit to enable the program to compile:

```
uses
  SysUtils, MSHTML, Dialogs;
```

Listing 6

Getting the Return Value

All this is very well, but how can we grab the return value of a JavaScript function called from Delphi? The short answer is that we can't, because the *execScript()* function doesn't ever return a useful value. Here's what the Microsoft SDK Help says about it (my emphasis):

> *Syntax*

```
HRESULT execScript(
  BSTR code,
  BSTR language,
  VARIANT *pvarRet
);
```

Parameters

code

[in] BSTR that specifies the code to be executed.

language

[in] BSTR that specifies the language in which the code is executed. The language defaults to Microsoft JScript.

pvarRet

[out, retval] Address of a VARIANT of type VT_EMPTY. ***This method always returns VT_EMPTY****.*

Return Value

Returns S_OK if successful, or an error value otherwise.

Huh?

What's the point of a function result that's always empty?

Search me!

Because Delphi uses the **safecall** convention for the definition of *execScript* in the `MSHTML` unit, the return value from *execScript* is that of the *pvarRet* parameter (flagged [retval]). (Delphi uses the documented return value to generate an exception if it is not *S_OK*). Notice that the help entry for *pvarRet* says it always returns *VT_EMPTY*. This means that *execScript* on Delphi always returns and empty variant, not the JavaScript function return value that we may have expected.

However, Christian Sciberras has come up with a clever work-around, providing you can modify the document's HTML and the JavaScript source code.

Taking our original JavaScript *foo()* function, let us assume it returns a value we want to capture.

First of all modify your HTML document's body so that it contains a hidden input field with an *id* attribute of 'result', i.e.:

```
<input type='hidden' id='result' value='' />
```

Listing 7

Assume that *foo()* was originally coded like this:

```
function foo(str, int) {
  var result;
  /* do something with str and int and store return value in result */
  return result;
}
```

Listing 8

We modify it to write its result into the hidden input field's *value* attribute like so:

```
function foo(str, int) {
  var result;
  /* do something with str and int and store return value in result */
  document.getElementById('result').value = result;
  return result;
}
```

Listing 9

Notice we've retained the return statement so that any other code that calls *foo()* still works. It remains to grab the value of the hidden input field from Delphi. *Listing 10* shows a generic function written by Christian that can get a named attribute from an HTML tag with a specified id.

```
function GetElementIdValue(WebBrowser: TWebBrowser;
  TagName, TagId, TagAttrib: string):string;
var
  Document: IHTMLDocument2;
  Body: IHTMLElement2;
  Tags: IHTMLElementCollection;
  Tag: IHTMLElement;
  I: Integer;
begin
  Result:='';
  if not Supports(WebBrowser.Document, IHTMLDocument2, Document) then
    raise Exception.Create('Invalid HTML document');
  if not Supports(Document.body, IHTMLElement2, Body) then
    raise Exception.Create('Can''t find <body> element');
  Tags := Body.getElementsByTagName(UpperCase(TagName));
  for I := 0 to Pred(Tags.length) do begin
    Tag:=Tags.item(I, EmptyParam) as IHTMLElement;
    if Tag.id=TagId then Result := Tag.getAttribute(TagAttrib, 0);
  end;
end;
```

Listing 10

The function first gets a reference to the <body> tag in the loaded HTML document. It then gets hold of a collection of all the contained tags named *TagName* and searches the collection looking for the tag with the required id (*TagId*). When the correct tag is found the function returns the value of its *TagAtrrib* attribute. If the tag or attribute is not found the empty string is returned.

Finally we just need to adapt the code from *Listing 1* that calls *foo()* to gets its return value. *Listing 11* shows the revised code.

```
uses
  MSHTML;

function TForm1.CallFoo(S: string; I: Integer): string;
  { Calls JavaScript foo() function and returns its return value }
var
  Doc: IHTMLDocument2;       // current HTML document
  HTMLWindow: IHTMLWindow2; // parent window of current HTML document
  JSFn: string;              // stores JavaScipt function call
begin
  Result := '';
  // Get reference to current document
  Doc := WebBrowser1.Document as IHTMLDocument2;
  if not Assigned(Doc) then
    Exit;
  // Get parent window of current document
  HTMLWindow := Doc.parentWindow;
  if not Assigned(HTMLWindow) then
    Exit;
  // Run JavaScript
  try
    JSFn := Format('foo("%s",%d)', [S, I]);    // build function call
    HTMLWindow.execScript(JSFn, 'JavaScript'); // execute function
    // get result
    Result := GetElementIdValue(WebBrowser1, 'input', 'result', 'value')
  except
    // handle exception in case JavaScript fails to run
  end;
end;
```

Listing 11

There are only three changes to the code:

1. *CallFoo* is changed from a procedure to a function that returns a string value.
2. A default return value of the empty string is set in case of error.
3. The return value of the function is set by calling the *GetElementIdValue* function from *Listing 10*.

Demo code

Demo code to accompany this article is available for download (see below). The demo includes the following:

- An implementation of the case study.
- A demo by Christian Sciberras that illustrates getting the return value a JavaScript function call. This demo lets you enter code that will be evaluated and displayed in a "console" window.

The source code was tested using Delphi 7 Professional.

This source code is merely a proof of concept and is intended only to illustrate this article. It is not designed for use in its current form in finished applications. The code is provided on an "AS IS" basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. The demo is open-source. See `license.txt` included in the download for license details.

Download the demo code
