Google Summer of Code 2017

Final Report

wxWebView JavaScript Integration

Jose Lorenzo

Organization: wxWidgets

INDEX

I. Introduction

Google Summer of Code is a Google project which main aim is to work in a open source project in summer. In my personal case I chose wxWebView JavaScript Integration, a project from wxWidgets organization. The main idea in this project is improve RunScript method to return every data type when JavaScript code has a result and it was reached on GTK, OSX and MSW. More information about this project idea could be found on [1] and [2].

1. Quick getting started

If you want to have a quick check of the project, you can check the following links (not pointing to references to have a quick view):

- Proposal: https://docs.google.com/document/d/1grSE69lUlE-3B-yglORXxohZgxUdvkOOru9cu3GSefI/edit?usp=sharing
- Proof of concept:
  - Explanation: https://github.com/joseeloren/wxWidgets/wiki/Proof-of-Concept:-Mock-RunScript-function-with-Return-Value
  - Code: https://github.com/joseeloren/Proof_of_Concept_WebKit2
- Pull request: https://github.com/wxWidgets/wxWidgets/pull/538
- Discussion thread: https://groups.google.com/forum/#!topic/wx-dev/2nPSfft_2mE%5B1-25%5D

II. Report Development

1. Schedule followed

Initially, there was a schedule to follow, that could be found on [3]. I followed it in most cases but sometimes it is difficult to do, because of some issues that happens on the way to reach the goal. So, I will detail what was done over the weeks until the goal was reached.

1. **First month.** During the first month the main aim of the project was discuss the API, study documentation, see possibilities and implement RunScript returning values on GTK. It was really challenging to get it working, because initially I get it working, in synchronous way. However, I didn't do it like the organization specs to do it, following some style guidelines. For instance, I used a void pointer, that it is not allowed by the that guide. Also, I implemented it asynchronously, but at the end, this was removed. More details about why this was removed will be on this document. The first API was returning a wxString and use a user_data pointer to pass some

information to the method. About the research done, I will write some more concrete words on other section, and it will describe why the project was done like it is.

2. **Second month.** With the API almost refined and some basic returning on GTK, was time to try with other platforms. On the first week I got OSX returning some values, but the native method to do it has bugs. So, I had to something done on OSX, so it is possible to reach the goal and I moved to MSW. MSW platform was really challenging because I didn't have experience with it before and with data types on this platform. Also, it was really weird because there was a method called execScript that doesn't work and also, it was deprecated. With some tests and a research, I found how to do it with Invoke method, calling internal JavaScript function eval. Also, some developers from the mailing list helped me to improve the code and no have memory leaks.
3. **Third month.** On the last month, it was a mixture of platforms. The main aim of this month is having everything ready and a common API on every platform. Also, that all platforms return the same data types and with the same shape. Finally, the final API was returning a boolean that if is true is that javascript code runs properly and false otherwise. The returning value will be on a wxString pointer. So, the final API looks like bool RunScript(const wxString& javascript, wxString* output). On this month there was a discussing on the mailing list of having or not having objects, null and undefined as returning types. This is because OSX doesn't support it natively and I had to make a workaround to get it working, mainly related to add a level more of escape quotes on the code. Finally, the idea was accepted and it is implemented like that on every platform (because this gave the opportunity to surround code by a JavaScript try-catch to get more information when the code fails). More technical aspects of this will be found on this document.

2. Information about platforms supported, methods implemented and a short summary of main features.

This information could be found also on the pull request done to wxWidgets master branch [4]. However, I will write it here to have everything on this report. More technical information could be found on RunScript wxWiki entry [5]. Also, there you can find images of the samples running. Most important information about this entry will be rewritten here to had it on a single document, organized.

RunScript method now returns strings, floating point numbers, booleans, objects (in JSON, as string), null and undefined. Also, I tested the code with Arrays and Dates, as special objects. Also, it is enabled to modify DOM with javascript.

Platforms support

All platforms support this features (GTK3, OSX and MSW). However, there are some limits that I will explain here, but they are also found on wxWiki and a complete summary on Doxygen docs.

On GTK1, I also modify the API, but it always returns true unless you give it an output variable. In that case, it will return false and no javascript will be run because GTK1 doesn't support WebKit2 (returning is impossible).

Methods implemented

Mainly, RunScript definition was changed. Before the project, it was void RunScript(const wxString& javascript) and now it is bool RunScript(const wxString& javascript, wxString* output = NULL).

Also, there are some internal methods and classes that help to get RunScript working. Reading this and the implementation could help to understand what is the workflow and clarify some words here.

On GTK (found mainly on src/gtk/webview_webkit2.cpp):

- *wxWebKitJavascriptResult Class.* It just has an explicit constructor, a destructor and an overlap operator *(). This class is for automating WebKitJavascriptResult freeing when scoping in finished. It is a RAII class and it is on include/wx/gtk/private/webkit.h.
- *wxJSStringRef Class.* It has a constructor, a destructor and a method called ToWxString. It is a RAII class for JSStringRef. ToWxString basically returns JSStringRef attribute as wxString. This class is included on include/wx/gtk/private/webkit.h.
- *RunScriptInternal Method (bool RunScriptInternal(const wxString& javascript, wxString* output)).*On GTK, there is this method. Basically, it is a helper to not repeat code. It calls webkit_web_view_run_javascript and waits until it gets the result. Then, JSResultToString process the result to get it as wxString and return it on output pointer.
- *wxgtk_run_javascript_cb Function*. It is for pass through the result. It is a callback connected to webkit_web_view_run_javascript.

- *JSResultToString (bool JSResultToString(GObject *object, GAsyncResult *result, wxString* output)) Function.* This function gets the result when javascript has finished. Then, it checks if it is well-formed and basically, returns the result as String in case of a non-object and as JSON if it is an object. Also, it checks if an error occurs and notifies the user (the developer). If everything goes perfectly, it returns the JS result on output pointer.
- *RunScript Method.* Obviously, it has the same API we said at the beginning. It calls JSScriptWrapper methods to get some Javascript to execute. This will be explained in a section for it, but mainly JSScriptWrapper is a class to get JS code in some way that reports errors and helps some platforms to return every type we defined previously (strings, integers, objects as JSON, etc.). It calls initially RunScriptInternal with the main javascript code with error handling, evaluating this code and putting it on a js variable. Then, this variable is transformed to a C++ output and finally, js variable is cleaned up.

On MSW (found mainly on src/msw/webview_ie.cpp):

- *MSWSetModernEmulationLevel Method (bool MSWSetModernEmulationLevel(bool modernLevel = true)).* This method is used to change the MSW registry entry related to which emulation level we want to run our program. It is designed to unset this registry entry (modernLevel = false) or set it to IE8 emulation level (modernLevel = true). I decided to use IE8 level because it is the last IE version that WinXP support. It is a deprecated operating system but really used worldwide and I think we need to give support to it. I will be a great idea to adapt this level to every Windows version but it could change the behaviour of RunScript and it is not a good way to do a cross-platform method, IMHO.
- *RunScriptInternal Method (bool RunScriptInternal(wxVariant varJavascript, wxAutomationObject* scriptAO, wxVariant* varResult)).* It is a method to help to not repeat code. Basically, it Invokes eval javascript function with the javascript code passed by varJavascript and returns the result on varResult. It returns false in case of fail and true in a successful case.
- *RunScript Method.* Obviously, it has the same API we said at the beginning. On this method, we get the HTML document, then the document script, and finally it is like GTK. The script is evaluated and the result is assigned to a javascript variable, then we check everything is okay, return the result and clean up the variable. As well as GTK, it is highly recommended goes to where JSScriptWrapper class is explained. Notice that on MSW it is compulsory to have a script tag on DOM (I didn't add it programmatically because there are some DOM examples without body or head, even without html tag, so it is not possible to get where put script tag).

On OSX(found mainly on src/osx/webview_webkit.mm):

- *RunScript method.* Obviously, it has the same API we said at the beginning. This is the most simple platform (about implementation). Firstly, I check if wxWebView object is created. Then is almost the same as GTK and MSW, related to wxJSScriptWrapper class. The script is evaluated and the result is assigned to a javascript variable, then we check everything is okay, return the result and clean up the variable. One important thing here is the native WebKit method used here, stringByEvaluatingJavaScriptFromString, is deprecated and Apple recommends to move to evaluateJavaScript:completionHandler [7]. Also, this method has some limits [8]:
    1. JavaScript allocations greater than 10MB are not allowed.
    2. JavaScript that takes longer than 10 seconds to execute is not allowed.

On "every platform" (found on include/wx/private/jsscriptwrapper.h):

- *wxJSScriptWrapper Class.* This class is mainly for help some platforms to support returning every type of variable. It has three methods:
    1. *GetWrapperCode method.* This method returns a Javascript code that does a few things. Firstly, evals the js code and put the result into a variable. Secondly, in case of error, with a Javascript try-catch, we can know what happens if the code fails. For doing this, it is needed to add quotes to call the js code with eval js function, so we need to escape internally the quotes because we add one more level because of this. Also, there is a trick to check if there is an error on eval or not and it returns true when eval finished. Then, on every platform, I have to check if it true otherwise there was an error and I have to report it to the user.

The main reason of this method (and actually, the class) is that on every platform, it is not possible to know if there is an error without it. On GTK and MSW, it is possible to know if there is an error, but not what kind of error is. On OSX, there is a bug (see [6]) that always returns a String (empty or not) in case js code runs properly or not.

2. *GetOutputCode method.* This returns, mainly, the variable with the appropriate content. However, it is a little bit different on every platform.

On OSX, nulls, objects and undefines are not supported natively, using UIWebView. So, I developed a simple JS code to get this working. If it is an object, it returns it a JSON (in includes nulls). If it is undefined, return 'undefined' as String. If it not this cases, it just returns the variable (because it is supported by the backend).

On GTK, it just returns the variable, WebKit2 is well implemented and accepts every type of returning values.

On MSW, it a little bit more tricky. When you call a wxVariant method MakeString() it does not return some types as wxString (booleans, nulls and undefines). So I decided to check if it is null or not an object, return the result as String (using js code). If it is not this case (it is an object but not null), it returns it as a string using JSON.stringify. By default, JSON objects don't exist on wxWebView on MSW. How we manage this it is explained in the appropriate section and also in Doxygen docs. But, in case it is defined, JSON.stringify is called. In case of not, a custom stringify (__wx$stringifyJSON) is called.

3. *GetCleanUpCode method*. This just returns a javascript code like every GetXXX() of this class. It assigns undefined to the variable created before (the variable that has the result of evaluating js code).

One more important thing about this class, the variable that has the output name. This variable changes of every call of RunScript, to prevent that it will be overlapped by another javascript result. So, it changes the name like this __wx$0, __wx$1.... __wx$counter. The counter is an integer attribute on the common class header (include/wx/webview.h) that is initialized by zero and increment by one on every RunScript call.

Other methods implemented: tests and samples.

I created this section to write about some other things implemented but not related to RunScript itself: tests and samples.

On tests, on tests/controls/webtest.cpp, I encounter some difficulties at the beginning, that I solved when I develop the new RunScript on every platform. Mainly, they occurred on GTK3 with WebKit2. One was that ENSURE_LOADED doesn't catch the event when it triggers. ENSURE_LOADED uses WX_ASSERT_EVENT_OCCURS to check if the website is loaded, to do some tests. So, I had to split the creation of wxWebView object using New and Create. In between of them, I create the event counter, so it could catch the event.

Also, by default, it loads about:blank page, so it does not need LoadURL("about:blank") anymore. On GTK, it catches that two events were triggered and it is logical, one about:blank was loaded by default and other by an implicit LoadURL.

Another thing that I encountered is that LoadURL takes too much to load an URL (more than the other platforms). I mention this because by default WX_ASSERT_EVENT_OCCURS has a time out of 100ms, so I had to refactor the code (on tests/testprec.h) that implies it and do a WX_ASSERT_EVENT_OCCURS_IN to put a custom time. I set it to 1 second, to have enough time to load a web page and not to fail by timeout.

About tests themselves, I removed HistoryEnable and HistoryClear from GTK3 because they are not implemented on WebKit2. Also, I added tests on RunScript to check the method is running properly: tests returning a string, integer, double, object, Date, array, modifying DOM, boolean, null, undefined and a failing js code. Moreover, I added tests to check the new level of quotes added because of wxJSScriptWrapper class and some special case related to having a backslash at the end of a string, and what happens with escaped quotes. On MSW, because there are two possibilities, doing JSON natively changing emulation level or using my custom stringify, I decided to put tests checking it also works when the registry value is changed.

Also, I tested if there are any memory leaks, mainly on GTK. This is because on OSX there isn't any pointer that I have to free and on MSW it was checked by more experienced people from the mailing list (Maarten Bent and PB). On OSX and MSW memory leaks checking was done by simple inspection and on GTK by a profiler, Valgrind. Valgrind report is available on [15].

Talking about the sample, on samples/webview/webview.cpp, I decided to change RunScript menu item to a submenu including all the possible returning values of the API (including on MSW the possibility to check with and without emulation level) and adding a menu item that when you click a blank space appears to add custom javascript code. So, the user could try with other different codes. I changed default web page on webview sample because it doesn't work with DOM sample (it returns false but the DOM is changed; wxAutomationObject doesn't get eval function). It will be related to having a script tag on the DOM and having the DOCTYPE like Internet Explorer expects. Some images of the example could be found on wxWiki [5].

3. Workflow

On every platform there is a common flow followed, as it was explained before. To illustrate this, I did a generic workflow diagram:

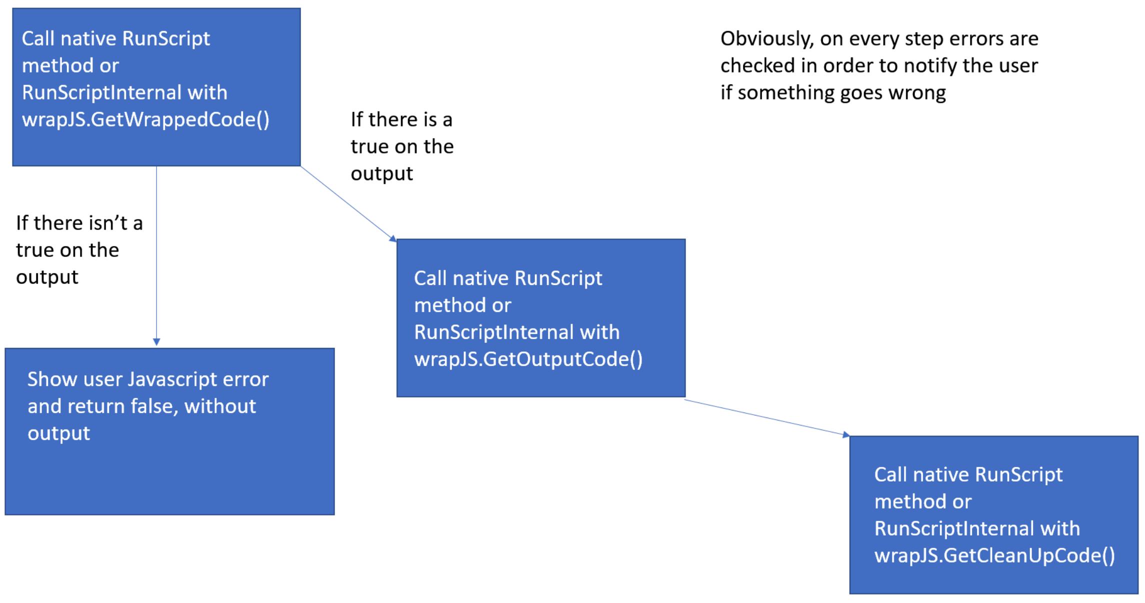
Call native RunScript method or RunScriptInternal with wrapJS.GetWrappedCode()
If there is a true on the output
If there isn't a true on the output
Show user Javascript error and return false, without output
Call native RunScript method or RunScriptInternal with wrapJS.GetOutputCode()
Call native RunScript method or RunScriptInternal with wrapJS.GetCleanUpCode()
Obviously, on every step errors are checked in order to notify the user if something goes wrong

Figure 1. Generic Workflow Diagram

4. Known issues.

Most issues are solved and I did some work to get some things working. Some issues I found and I solved are:

1. Returning objects, null and undefined on OSX. As I said before, this is not supported natively by OSX native method. So, I had to make a workaround using javascript code (found on wxJSScriptWrapper class GetOutputCode method). It is based on check internally on JavaScript if there are some of this types and returning them as string.
2. Returning objects on MSW. One challenge thing was doing this on MSW. MSW Invoke supports every data type but objects were returned on a wxVariant as IDispatch. Finally, as it was explained on the previous section, it is checked internally if JSON.stringify exists and in that case, the object is returned as JSON. If not, a custom stringify is used. Dealing with IDispach is not required anymore, because it is done internally by Javascript. There were a way to do it with the wxAutomationObject and checking if it was a void* type, but I remove this way to do it because it fails on some webpages (wxAutomationObject doesn't found eval when DOM is changed; it actually changes DOM but returns an error).

Some issues not solved (because of the backend):

1. Returning a Date object as String with this format on MSW: "yyyy-mm-ddThh:mm:ssZ". This is not supported by Internet Explorer versions below 9. Further information could be found on [9].

5. Future work

There are some work that could be done on RunScript on the future, that it wasn't done because there isn't enough time and I believe it is not know the time to do it (I will explain in the following words why I think this).

1. Synchronous and asynchronous method. On of the ideas that came up when this was develop is develop it synchronous and/or asynchronous. Nowadays, it is synchronous because there are one main limit: MSW backport. Internet Explorer API, as I researched doesn't have a method or a way to do a native asynchronous call to JavaScript. I had an idea to do it with wxThread, but someone told me not to do it like this. So, I did it synchronously.

Some of the advantages of doing synchronous and asynchronous is that the interface doesn't get stuck. As you can imagine, if the main loop waits until JavaScript code finish, the interface stucks. However, this is was done on GTK, a asynchronous method but it stucks the interface too. I didn't research more about this because the MSW limit.

On OSX, there is a way to do it asynchronously, that is using the new backport, WKWebView, instead of UIWebView, that has the retuning Javascript method deprecated (see [7]). It will be interesting, in the future, move to this backport. I didn't take it because I didn't have time to do it (because I had to develop a new wxWebView class for OSX; also I tried just invoking a WKWebView object and calling the method, see [10] but it works worst than UIWebView) and WKWebView evaluateJavaScript method is supported from macOS 10.10+ version. An idea was develop a wxWebVierw class with WKWebView and use this one or the old one depending on OSX version. However, we still having the same problem: older version still having the issue of synchronous version. Maybe, in the future, it will be nice to do it, when versions below 10.10 will be deprecated.

2. Because of the first future work, I will add this one. Why I didn't change backport on MSW to do an asynchronous method? There were ideas to port to Edge or compile WebKit on Windows but on the mailing list there was the idea of doing this with Internet Explorer API and I think that, on this moment, is the best way to do it. It is the same as OSX problem, there are computer that runs Windows XP nowadays, so it is important to have backward compatibility. Microsoft Edge is only available on Windows 10 and it not a good idea implement a class to use Edge, invest a lot of time with it now to have a certain percent of users worldwide that uses this browser. Also, I didn't found anything related to call C++ events on Edge and there is something that exists on WebKit (see [11]).

Other ideas were about moving to Chrome core. There is a Steve Lamerton implementation of this on Windows found on [12]. Another related example of something like this is found on Chromium Embedded JavaScript integration on [13].

6. Compilation step-by-step instructions

This information could be found on the wxWiki that has technical aspects of the project, in [5].

7. Sample C++ with Javascript and demo app usage

This information could be found on the wxWiki that has technical aspects of the project, in [5].

8. Even more information about the project

If you are interesting on even more information about the project, there is a discussion about it on the mailing list. It could be found on [14].

9. Acknowledgements

On this project, I received some support and help from different people from wxWidgets organization. These people help me on some challenges that I had to do and I didn't have enough experience or knowledge to resolve them properly.

- *Mariano Reingart.* One of my mentors. He was with me from the first moment, when I sent the proposal and I did the proof of concept. Also, anything we had to talk about the project or doubts I had he tried to solve them as soon as possible. Also, he had some interesting ideas added to the project, like adding a boolean true at the end of the try block to check that the eval function is running properly. Contact: https://github.com/reingart
- *Steve Lamerton.* My other mentor. He also was an important person on the project. He has more experience than Mariano on some fields related to GTK and wxWidgets itself. Steve helped me a lot with the project and with doubts I had, and with some programming related problems. Contact: https://github.com/sjlamerton
- *Vadim Zeitlin.* He is part of the core team. He helped me in how to drive the project to a wxWidgets like project. He resolved me many doubts about wxWidgets and internal structure. Also, he had some interesting ideas about how to manage the one more level of quotes on JavaScript and how to deal with different RunScript calls (using __wx$counter). Contact: https://github.com/vadz
- *PB.* He contributed on MSW part mainly. He also design a code using wxAutomationObject that I didn't know that it exists. In addition, he also helped finding memory leaks on MSW. Part of that code was used on Contact: https://github.com/pbfordev
- *Maarten Bent.* As PB, he contributed on MSW part (I didn't have enough experience to deal with MSW backport properly). He also contribute on a previous code to get a MSW RunScript working. Moreover, he contribute in not having leaks on MSW and the registry key modification in order to get it working on Internet Explorer compatibility mode. Contact: https://github.com/MaartenBent
- I also appreciate comments and words from other people from the mailing list. *Catalin Raceanu, Stefano Mtangoo, Igor Korot, Scott Talbert, John Roberts* and other people from outside/inside the mailing list helped me on this road to get the project finished. They gave me some little keys or some works that help me to continue on this project.

In summary, I have to say **thank you** to all people that helped me to reach the goal and finish the project on time and with non-significant issues. I am glad to work with this amazing people.

III. Conclusions and final remarks

Talking about the final product, I am really happy with the result, it is what I expected to do at the beginning of the summer. It returns every data type on every platform. However, it will be great if I could do calling C++ methods from Javascript, but there aren't enough time to research, develop, discuss and so on.

About the implementation timeline I think I followed it more or less like it was defined. I just changed, as a requirement, the implementation between platforms (it should have been on platform per month but I had a simple returning data types at the end of July). However, I think that it was more than enough time to do a powerful product,

There were some difficulties during the development, as usual. Some of them were related to my inexperience on this specific field and the style guide that I had to follow. Also, I think that community opinion is necessary to not develop unliked code. I mean, they wanted some characteristics and code style, and develop in some way, not like I develop as usual. But I strongly believe I improved the way I develop code and how to solve problems.

About recommendations I don't have any. I think the workflow as followed as it is expected and the organization has previous experience with Google Summer of Code. I really appreciate how the community is and how helpful it is when I need to know something that I don't.

As final conclusion, I enjoyed developing this project. It was really challenging, in my humble opinion and I leant a lot about wxWidgets, open source and coding.

IV. References

[1]. wxWidgets projects ideas for GSoC. https://www.wxwidgets.org/gsoc/projects/
[2]. GSoC page with project description.
https://summerofcode.withgoogle.com/projects/#6085667476996096
[3]. Initial schedule. https://github.com/joseeloren/wxWidgets/milestones?state=closed
[4] RunScript Pull Request. https://github.com/wxWidgets/wxWidgets/pull/538
[5]. RunScript wxWiki. https://wiki.wxwidgets.org/RunScript
[6]. OSX native method (deprecated) bug
https://stackoverflow.com/questions/37108379/how-does-stringbyevaluatingjavascriptfromstring-return-nil
[7]. OSX native method definition
https://developer.apple.com/documentation/uikit/uiwebview/1617963-stringbyevaluatingjavascriptfrom?language=objc

[8] OSX native method (deprecated) limits
https://stackoverflow.com/questions/7388743/stringbyevaluatingjavascriptfromstring-doesnt-always-seem-to-work
[9] Date.prototype.toISOString by Mozilla:
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Date/toISOString
[10] WKWebView evaluting JavaScript implementation example.
https://stackoverflow.com/questions/26778955/wkwebview-evaluate-javascript-return-value
[11] Calling a C++ callback when a DOM item is clicked, for example.
https://webkitgtk.org/reference/webkitdomgtk/2.3.2/WebKitDOMEventTarget.html
[12] wxWebViewChromium by Steve Lamerton.
https://github.com/sjlamerton/wxWebViewChromium
[13] Chromium Embedded JavaScript integration by CEF
https://bitbucket.org/chromiumembedded/cef/wiki/JavaScriptIntegration.md
[14] Mailing List WebView JavaScript Integration discussion
https://groups.google.com/forum/#!topic/wx-dev/2nPSfft_2mE%5B176-200%5D
[15] Valgrind report.
https://drive.google.com/file/d/0BxbyJZWK1lF-ZWdFckx0S0J5dXc/view?usp=sharing