



HB_WebView

A Harbour wrapper for the cross-platform Webview C/C++ library
(<https://github.com/webview/webview>)

Copyright 2025 by Dr. Claudio Soto (from Uruguay).
mail: <srvet@adinet.com.uy>
blog: <http://srvet.blogspot.com>

Webview library is a tiny cross-platform webview library for C/C++ to build modern cross-platform GUIs. The goal of the Webview library is to create a common HTML5 UI abstraction layer for the most widely used platforms. It supports two-way JavaScript bindings (to call JavaScript from C/C++ and to call C/C++ from JavaScript).

Now with HB_WebView you can easily bind Harbour functions with Javascript, that is, you can call any Harbour function from an HTML page.

Download the source code and demos of the HB_WebView from:
<https://www.hmgforum.com/viewtopic.php?t=7659>

https://drive.google.com/open?id=1gCRK__9fylXj29D28DuAnMJXqbcgej7O&usp=drive_fs

Main functions

1) Creates a new webview instance.

w = hb_webview_create(debug, window)

@param debug If true enable developer tools if supported by the backend.

@param window Optional native window handle, i.e. @c GtkWidget pointer, @c NSWindow pointer (Cocoa) or @c HWND (Win32). If non-null, the webview widget is embedded into the given window, and the caller is expected to assume responsibility for the window as well as application lifecycle. If the window handle is null, a new window is created and both the window and application lifecycle are managed by the webview instance.

2) Destroys a webview instance and closes the native window.

hb_webview_destroy(w)

@param w The webview instance.

3) Runs the main loop until it's terminated.

hb_webview_run(w)

@param w The webview instance.

4) Stops the main loop.

It is safe to call this function from another other background thread.

hb_webview_terminate(w)

@param w The webview instance.

5) Returns the native handle of the window associated with the webview instance.

The handle can be a @c GtkWidget pointer (GTK), @c NSWindow pointer (Cocoa) or @c HWND (Win32).

window = hb_webview_get_window(w)

@param w The webview instance.

@return The handle of the native window.

6) Get a native handle of choice.

handle = hb_webview_get_native_handle(w, kind)

@param w The webview instance.
@param kind The kind of handle to retrieve.
@return The native handle or @c NULL.

The @param kind can be one of the following constants:

WEBVIEW_NATIVE_HANDLE_KIND_UI_WINDOW

Return the Top-level window. @c GtkWindow pointer (GTK), @c NSWindow pointer (Cocoa) or @c HWND (Win32).

WEBVIEW_NATIVE_HANDLE_KIND_UI_WIDGET

Return the Browser widget. @c GtkWidget pointer (GTK), @c NSView pointer (Cocoa) or @c HWND (Win32).

WEBVIEW_NATIVE_HANDLE_KIND_BROWSER_CONTROLLER

Return the Browser controller. @c WebKitWebView pointer (WebKitGTK), @c WKWebView pointer (Cocoa/WebKit) or @c ICoreWebView2Controller pointer (Win32/WebView2).

7) Updates the title of the native window.

hb_webview_set_title(w, title)

@param w The webview instance.
@param title The new title.

8) Updates the size of the native window.

Remarks: Using WEBVIEW_HINT_MAX for setting the maximum window size is not supported with GTK 4 because X11-specific functions such as gtk_window_set_geometry_hints were removed. This option has no effect when using GTK 4.

hb_webview_set_size(w, width, height, hints)

@param w The webview instance.
@param width New width.
@param height New height.
@param hints Size hints.

The @param hints can be one of the following constants:

WEBVIEW_HINT_NONE

Width and height are default size.

WEBVIEW_HINT_MIN

Width and height are minimum bounds.

WEBVIEW_HINT_MAX

Width and height are maximum bounds.

WEBVIEW_HINT_FIXED

Window size can not be changed by a user.

9) Navigates webview to the given URL.

URL may be a properly encoded data URI.

Example:

```
hb_webview_navigate(w, "https://github.com/webview/webview")
hb_webview_navigate(w, "data:text/html,%3Ch1%3EHello%3C%2Fh1%3E")
hb_webview_navigate(w, "data:text/html;base64,PGgxPkhlbGxvPC9oMT4=")
```

hb_webview_navigate(w, url)

@param w The webview instance.

@param url URL.

10) Load HTML content into the webview.

Example:

```
hb_webview_set_html( w, "<h1>Hello</h1>" );
```

hb_webview_set_html(w, html)

@param w The webview instance.

@param html HTML content.

11) Injects JavaScript code to be executed immediately upon loading a page.

The code will be executed before window.onload.

hb_webview_init(w, js)

@param w The webview instance.

@param js JavaScript content.

12) Evaluates arbitrary JavaScript code.

Use bindings if you need to communicate the result of the evaluation.

hb_webview_eval(w, js)

@param w The webview instance.

@param js JavaScript content.

13) Binds a Harbour function to a new global JavaScript function.

Internally, JS glue code is injected to create the JS function by the given name. The callback function is passed a request identifier, a request string and a user-provided argument. The request string is a JSON array of the arguments passed to the JS function.

```
context = hb_webview_bind2( w, funcName, callType,  
no_translate_param_toANSI, no_translate_ret_toUTF8 )
```

@param w The webview instance.

@param funcName Name of the JavaScript function.

@param callType Type of way in which Harbour functions are called.

@param no_translate_param_toANSI By default is false. If set to true, parameters are always passed to the function in UTF8. This is useful when the called Harbour function only works with UTF8 arguments. If set to false, parameters are converted from UTF8 to the specified ANSI code page. If Harbour's current code page is UTF8, no translation is performed.

@param no_translate_ret_toUTF8 By default is false. If set to true, the value returned by the Harbour function called is not translated to UTF8. It is useful when the returned value is in ANSI and should remain that way without any translation.

@return The context (memory space) in which the Harbour function is bind.

The @param callType can be one of the following constants:

HB_CALL_FUNC_INDIRECT

Allows you to bind a single Harbour function and use it to call any other Harbour function. The first parameter to the bind function is the name of the Harbour function to be called, enclosed in quotation marks, followed by its parameters, separated by commas.

e.g. `hb_webview_bind2(w, "callHB", HB_CALL_FUNC_INDIRECT)`
call in JS as `callHB( "hb_memoread", "text.txt" );`
results in Harbour as `&'hb_memoread("text.txt")'`

HB_CALL_FUNC_DIRECT

Allows you to bind a Harbour function and call it in the usual way as it is called in Harbour.

e.g. `hb_webview_bind2(w, "hb_memoread", HB_CALL_FUNC_DIRECT)`
call in JS as `hb_memoread( "text.txt" );`
results in Harbour as `&'hb_memoread("text.txt")'`

HB_CALL_FUNC_MACRO

Allows you to bind a single Harbour function and use it to call any other Harbour function. The bind function takes a single parameter enclosed in quotes and consists of a macro written in the usual Harbour format.

e.g. `hb_webview_bind2(w, "callMACRO", HB_CALL_FUNC_MACRO)`
call in JS as `callMACRO( 'hb_memoread("text.txt")' );`
results in Harbour as `&'hb_memoread("text.txt")'`

Remember that JavaScript is case sensitive: If you bind the HB_ALERT() function as "hb_Alert" in the Harbour environment, you must call it as hb_Alert() in JavaScript environment.

On the other hand, Harbour binding functions are passed as promises to JavaScript. Therefore, they are executed in the JavaScript environment as asynchronous functions.

For more details on promises in JavaScript, see:

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise

For promises to execute synchronously, you must chain the calls in chained .then() statements or call them with the await statement inside an asynchronous function.

For more details see:

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise/then

<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/await>

14) Sets the codepage for translating characters between UTF8 and ANSI

hb_webview_translate_ansi_codepage(ansi_codepage)

@param ansi_codepage Is the Harbour ANSI codepage if you work with ANSI data (e.g. "ESWIN", "cp1252", etc.). This is useful when the Harbour codepage is set to UTF8 and you are working with ANSI data, as this way the characters are correctly translated between UTF8 and ANSI by HB_WebView. It is not necessary to set a codepage when Harbour is working in ANSI because HB_WebView defaults to Harbour's current codepage to translate characters between UTF8 and ANSI.

@return Returns the previously set codepage.

15) Removes a binding created with hb_webview_bind and hb_webview_bind2 functions.

hb_webview_unbind(w, funcName)

@param w The webview instance.

@param funcName Name of the binding.

16) Frees the memory allocated to the context.

hb_webview_destroy_context(context)

@param context The context instance.

17) Gets the webview C/C++ library version information.

hb_webview_version()

18) Gets the HB_WebView library version information.

hb_webview_wrapper_version()

Undocumented functions for use in background

```
hb_webview_dispatch( w, ptrFunc, context )
hb_webview_bind( w, name, ptrFunc, context )
hb_webview_return( w, id, status, result )
context = hb_webview_create_context( w, funcName, callType, ptrCargo )
hb_webview_set_callHB_func( cHB_callback_function_name )
hb_webview_HbBridgeJS( par_cJsPromiseId, par_cJsonParameter,
par_ptrContext, par_cFuncName, par_nCallType, no_translate_param_toANSI,
no_translate_ret_toUTF8 )
```

Navigate Demo

```
#include "hb_webview.ch"

FUNCTION Main
LOCAL w, allowDevTools := .F.

    w = hb_webview_create( allowDevTools, NIL )

    hb_webview_set_size(w, 800, 480, WEBVIEW_HINT_NONE)

    hb_webview_set_title(w, "Harbour WebView - Navigate Demo")

    hb_webview_navigate(w, "https://harbour.github.io/doc/")

    hb_webview_run(w)

    hb_webview_terminate( w )

RETURN NIL
```

Bind Demo

```
#include "hb_webview.ch"

REQUEST hb_alert, hb_memoread, hb_memowrit

FUNCTION Main
LOCAL w, html, allowDevTools := .T.
LOCAL context1, context2
```

```

html = ''
html += '<div>'
html += '  <p>Press CTRL+SHIFT+I to open the DevTools window</p>'
html += '  <button onclick="dispText()"> hb_memoread( "demo1.prg" )
</button>'
html += '  <pre id="idResult"></pre>'
html += '</div>'
html += '<script>'
html += '  async function dispText(){'
html += '    document.getElementById("idResult").textContent = await
hb_memoread("demo1.prg");'
html += '    await hb_alert("HB_MemoRead(\"demo1.prg\") done!");'
html += '  }'
html += '</script>'

w = hb_webview_create( allowDevTools, NIL )

  hb_webview_set_size(w, 480, 320, WEBVIEW_HINT_NONE)

  hb_webview_set_title(w, "Harbour WebView - Bind Demo")

  context1 = hb_webview_bind2(w, "hb_memoread", HB_CALL_FUNC_DIRECT)
  context2 = hb_webview_bind2(w, "hb_alert",    HB_CALL_FUNC_DIRECT)

  hb_webview_set_html(w, html)

hb_webview_run(w)

hb_webview_destroy_context( context1 )
hb_webview_destroy_context( context2 )

hb_webview_terminate( w )

RETURN NIL

```