W WEBVIEW2

AUTOIT UDF - QUICK START

WebView2
in ten minutes

From an empty script to a browser you can drive: four short programs, every line commented, nothing to install and nothing to download.

AutoIt 3.3.16+ | No installer | No network | 4 programs | 76 functions

```
#include <GUIConstantsEx.au3>
#include "WebView2.au3"

Local $hGui  = GUICreate("Window", 900, 620)
GUISetState(@SW_SHOW, $hGui)

Local $hView = _WV2_Create($hGui, 0, 0, -1, -1)
_WV2_NavigateToString($hView, "<h1>Hello</h1>")

While GUIGetMsg() <> $GUI_EVENT_CLOSE
WEnd

GUISetState(@SW_HIDE, $hGui)
_WV2_Destroy($hView)
GUIDelete($hGui)
```

One include. That is the whole setup.

THE WHOLE DEPENDENCY LIST

VERSION	RUNTIME	LICENCE
0.6.0	AutoIt 3.3.16+	MIT

WHAT YOU NEED

Windows 10 or 11
WebView2 runtime included

▼

AutoIt 3.3.16+
x86 and x64, tested on x86

▼

WebView2.au3
next to your script

BY THE NUMBERS

public functions	76
complete programs	4
installers to run	0
network calls	0
files to register	0

Every program in this booklet was run before it was printed: local pages only, no network, no leftover file.

THE FOUR PROGRAMS

01 **Your first window** — hello from the engine

02 **Events and messages** — the page and the host talk

03 **Your own content** — JavaScript answers

04 **Fitting the window** — a toolbar that follows

COLOPHON

Typeset with PDFX 1.4.0, the author's other UDF.

PDFX - forum topic 213930

Numeric1 - forum profile 118605

Version 0.6.0 - October 2026

HOW TO USE THIS BOOKLET

Start here

Six chapters. The first one is already a working program; the next five add one idea at a time.

01 Your first window 03

02 Events and messages 05

03 Your own content 09

04 Fitting the window 12

05 Recipes 15

06 When it goes wrong 19

Every entry is clickable; every heading is also a bookmark.

BEFORE YOU TYPE

01 Windows 10 or 11, with the WebView2 runtime - it is already there.

02 AutoIt 3.3.16 or later - the samples were run on AutoIt3.exe (x86).

03 WebView2.au3 next to your script - nothing to register.

THE SHAPE OF IT

Four complete programs. The snippets between them are pieces of those programs, not separate files. This 0.6 edition carries the three lots that took the UDF from forty functions to seventy-six: chapter 5 shows what they made possible.

75 PUBLIC FUNCTIONS one include	**4** RUNNABLE PROGRAMS commented
0 FILES TO REGISTER nothing to install	**6** CHAPTERS in reading order

READING PATHS

First contact	01
A real application	01 - 02 - 04
Look something up	05 - 06

What this is not

Not the API reference. The 76 functions carry their own #FUNCTION# headers, and the Developer Guide explains the machinery under them.

THREE HABITS that avoid most beginner problems

1. **Keep a band free**
   The browser is a child window and covers controls under it.
2. **Drain the queue**
   Call _WV2_GetEvent on every pass of the loop.
3. **Destroy the view first**
   Hide the window, _WV2_Destroy, then GUIDelete.

01

FROM ZERO TO A PAGE

Your first window

One include, one window, one page. The program is complete: type it, run it, and you have a browser.

A WebView2 view is a child window inside a normal AutoIt GUI. There is no ActiveX control and no file to register: the UDF builds the browser through the runtime Windows already carries, so the whole installation is one step - put WebView2.au3 next to your script.

What you need

- **Windows 10 or 11.** The WebView2 runtime comes with Microsoft Edge and is present on any maintained machine. If it is ever missing, _WV2_Create returns 0 with @error = $WV2_ERR_RUNTIME (3); nothing is downloaded.
- **AutoIt 3.3.16 or later.** Both AutoIt3.exe and AutoIt3_x64.exe work. The programs in this booklet were checked on 3.3.16.1, x86.
- **WebView2.au3.** One file, next to your script. The example and the self test in the package are optional.

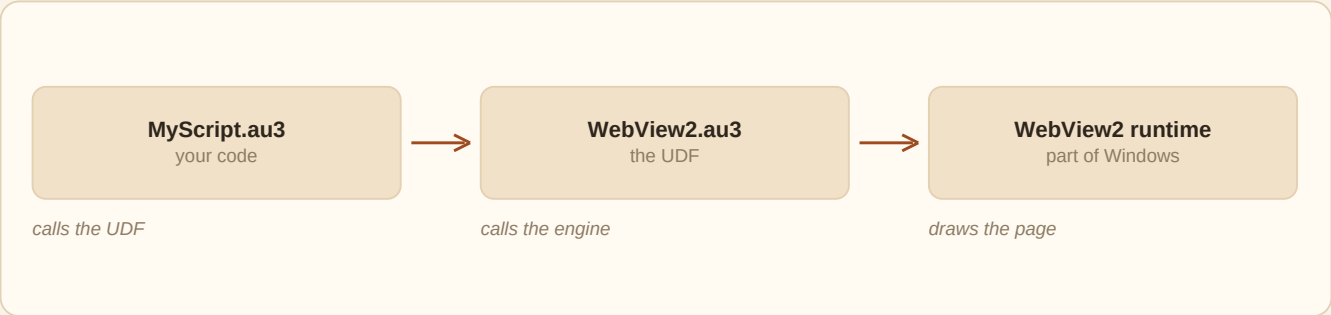


Figure 1. *Only the left box is yours. The UDF holds the COM plumbing; the engine is the part of Windows that renders the page.*

The program

Six numbered comments divide it. Nothing is hidden: this is the whole file, comments included.

```
; ============================================================
;  My first WebView2 window
;  AutoIt 3.3.16+  -  no network, no extra file
; ============================================================
#include <GUIConstantsEx.au3>
#include "WebView2.au3"

Opt("MustDeclareVars", 1)

; 1. An ordinary AutoIt window. Nothing special yet.
Global $hGui = GUICreate("My first WebView2", 900, 620)
GUISetState(@SW_SHOW, $hGui)

; 2. Put the browser inside it. 0, 0, -1, -1 means "from the
;    top left corner, take all the rest of the client area".
Global $hView = _WV2_Create($hGui, 0, 0, -1, -1)
If $hView = 0 Then
    ; Always test the handle: a window without a browser is useless.
    MsgBox(16, "WebView2", "The view could not be created." & @CRLF & @CRLF & _
            "@error = " & @error & "   @extended = " & @extended & @CRLF & _
```

```
            _WV2_GetLastErrorMessage())
    Exit 1
EndIf

; 3. Keep the browser out of sight until its page is really there. The first
;    start of the engine takes seconds; an empty frame is what a user calls a
;    freeze, and this line makes sure they never see one.
_WV2_SetVisible($hView, False)

; 4. Give it a page. The HTML travels with the script, so this
;    works offline and leaves no file behind.
Global $sHtml = _
        "<html><body style='font-family:Segoe UI;background:#FAF4E8;color:#2E251C'>" & _
        "<div style='padding:48px'>" & _
        "<h1 style='color:#9E4B1F'>Hello from the Edge engine</h1>" & _
        "<p>This page is drawn by the same engine as Microsoft Edge,</p>" & _
        "<p>inside an AutoIt window, with one include and no installer.</p>" & _
        "</div></body></html>"
_WV2_NavigateToString($hView, $sHtml)

; 5. Wait for the page. 1 = the navigation finished, 0 = it
;    never did; @extended = 0 when the page loaded correctly,
;    otherwise the web error code. Messages keep flowing.
Global $iDone = _WV2_WaitNavigation($hView)
If $iDone = 0 Or @extended <> 0 Then
    MsgBox(16, "WebView2", "The page did not load. @extended = " & @extended & @CRLF & _
            _WV2_GetLastErrorMessage())
EndIf

; 6. Now it can be seen. Revealing a view that already holds its page costs a
;    few milliseconds - measured four - and the window appears with content.
_WV2_SetVisible($hView, True)

; 7. The message loop. WebView2 needs it as much as the window:
;    without it nothing is painted and the window freezes.
While 1
    If GUIGetMsg() = $GUI_EVENT_CLOSE Then ExitLoop
WEnd

; 8. Hide the window, then close the browser. The browser takes a few
;    hundred milliseconds to leave; with the window already hidden the
;    user never sees a page that has disappeared.
GUISetState(@SW_HIDE, $hGui)
_WV2_Destroy($hView)
GUIDelete($hGui)
```

What just happened

CALL	WHAT IT DOES
_WV2_Create	Returns a handle, 1 to 16. Every other call takes it first.
-1, -1	Takes the rest of the client area from $iLeft, $iTop.
_WV2_WaitNavigation	Waits and pumps messages; @extended is 0 or the web error.
_WV2_Destroy	Releases the engine; hide the window before it, on the same thread.

NEXT CHAPTER 02 Events and messages

02

TWO DIRECTIONS, ONE QUEUE

Events and messages

The page talks, AutoIt listens. AutoIt talks, the page listens. Neither side blocks.

WebView2 never calls your script directly. It queues events, and your GUI loop reads that queue. The design keeps a slow handler from stalling the engine, but it also means the loop has to drain the queue on every pass.

The bridge, both ways

In the page, the bridge is one object: chrome.webview. A script calls postMessage to send to AutoIt, and listens to the message event to receive.

```
// page -> AutoIt
chrome.webview.postMessage("hello from the page");

// AutoIt -> page
chrome.webview.addEventListener("message", function (e) {
    document.getElementById("out").textContent = e.data;
})
```

On the AutoIt side, both directions are single calls. A plain string is easier on both ends; JSON is there when a structure has to cross.

```
_WV2_EnableEvent($hView, $WV2_EVENT_WEB_MESSAGE)   ; before loading
_WV2_PostWebMessageAsString($hView, "ping")        ; AutoIt -> page
_WV2_PostWebMessage($hView, "{""x"": 1}")           ; JSON text
```

Only NAVIGATION_COMPLETED is enabled by _WV2_Create. Every other event stays off until _WV2_EnableEvent, and enabling twice is harmless.

The event queue

One call takes the oldest event and fills two variables. The table says what they hold.

CONSTANT	$SDATA	$IVALUE	ON BY DEFAULT
$WV2_EVENT_NAVIGATION_STARTING	the URI	1 if the user started it	no
$WV2_EVENT_NAVIGATION_COMPLETED	the URI	1 on success, else the web error	yes
$WV2_EVENT_WEB_MESSAGE	the message	1 plain string, 0 JSON	no
$WV2_EVENT_TITLE_CHANGED	the new title	nothing	no
$WV2_EVENT_HISTORY_CHANGED	nothing	nothing	no
$WV2_EVENT_NEW_WINDOW	the URI	nothing	no
$WV2_EVENT_PROCESS_FAILED	nothing	$WV2_PROCFAIL_* 0 to 9	no
$WV2_EVENT_DOWNLOAD_STARTING	the URI	1 refused, 0 accepted	no

NINE MORE KINDS

The 0.4 and 0.5 lots added $WV2_EVENT_RESOURCE_REQUESTED, _FIND, _DOWNLOAD_STATE, _DOWNLOAD_PROGRESS, _SCRIPT_DIALOG, _ACCELERATOR, _CONTEXT_MENU, _FAVICON_CHANGED and _DEVTOOLS. They reach the same queue and the same _WV2_GetEvent; chapter 5 uses most of them, and the Developer Guide lists what each one carries.

QUEUE SIZE

Each view keeps 64 events. When the queue overflows the oldest is dropped, and _WV2_GetEvent reports how many were lost in @extended. Draining it on every pass is cheaper than sizing it.

```
While 1
    Switch GUIGetMsg()
        Case $GUI_EVENT_CLOSE
            ExitLoop
        Case $idPing
            _WV2_PostWebMessageAsString($hView, "ping")
    EndSwitch

    Local $sData = "", $iValue = 0
    While 1
        Local $iEvent = _WV2_GetEvent($hView, $sData, $iValue)
        If $iEvent <= 0 Then ExitLoop   ; 0 = empty, -1 = bad handle
        ; ... use $iEvent, $sData and $iValue here ...
    WEnd
WEnd
```

_WV2_WaitEvent is the opposite: it pumps until one event of the type you name arrives, leaving the others in the queue. It is handy right after a navigation.

The program

```
; ============================================================
;  Events and messages: page -> AutoIt, AutoIt -> page
; ============================================================
#include <GUIConstantsEx.au3>
#include "WebView2.au3"

Opt("MustDeclareVars", 1)

; The window: a browser on top, 40 pixels for our own controls.
Global $hGui = GUICreate("Events and messages", 900, 620)
Global $idPing = GUICtrlCreateButton("Ping the page", 12, 588, 150, 24)
Global $idInfo = GUICtrlCreateLabel("Waiting for the page...", 176, 593, 700, 20)
GUISetState(@SW_SHOW, $hGui)

Global $hView = _WV2_Create($hGui, 0, 0, -1, 580)
If $hView = 0 Then
    MsgBox(16, "WebView2", "The view could not be created: " & _WV2_GetLastErrorMessage())
    Exit 1
EndIf

; The engine takes seconds for its first page. Keep it out of sight until one is
; really there, so the user never reads that time as a freeze.
_WV2_SetVisible($hView, False)

; Ask for the events we care about, before loading the page.
```

```
; NAVIGATION_COMPLETED is already on; every other event is off.
_WV2_EnableEvent($hView, $WV2_EVENT_TITLE_CHANGED)
_WV2_EnableEvent($hView, $WV2_EVENT_WEB_MESSAGE)

; The page: one button that posts, one listener that receives.
Global $sHtml = _
        "<html><head><title>Two directions</title></head>" & _
        "<body style='font-family:Segoe UI;background:#FAF4E8;color:#2E251C'>" & _
        "<div style='padding:40px'>" & _
        "<h1 style='color:#9E4B1F'>Two directions</h1>" & _
        "<button style='font-size:16px;padding:8px 14px' " & _
        "onclick=""chrome.webview.postMessage('hello from the page')"">" & _
        "Send a message to AutoIt</button>" & _
        "<p id='out' style='margin-top:28px;font-size:18px'>" & _
        "No message from AutoIt yet.</p>" & _
        "<script>" & _
        "chrome.webview.addEventListener('message', function (e) {" & _
        "  document.getElementById('out').textContent = 'AutoIt says: ' + e.data;" & _
        "});" & _
        "</script></div></body></html>"
_WV2_NavigateToString($hView, $sHtml)
_WV2_WaitNavigation($hView)
; Four milliseconds, and the window appears with its page in it.
_WV2_SetVisible($hView, True)

; The loop: GUI messages first, then the event queue.
While 1
    Switch GUIGetMsg()
        Case $GUI_EVENT_CLOSE
            ExitLoop
        Case $idPing
            ; AutoIt -> page: the text arrives in event.data.
            _WV2_PostWebMessageAsString($hView, "ping at " & @HOUR & ":" & @MIN)
    EndSwitch
    _DrainEvents()
WEnd

; Hide first, then close the browser: the few hundred milliseconds it takes to
; leave happen behind a window the user has already stopped looking at.
GUISetState(@SW_HIDE, $hGui)
_WV2_Destroy($hView)
GUIDelete($hGui)

; Page -> AutoIt: take everything that arrived since the last
; pass. 0 = queue empty, -1 = invalid handle.
Func _DrainEvents()
    Local $sData = "", $iValue = 0
    While 1
        Local $iEvent = _WV2_GetEvent($hView, $sData, $iValue)
        If $iEvent <= 0 Then ExitLoop
        Switch $iEvent
            Case $WV2_EVENT_TITLE_CHANGED
                WinSetTitle($hGui, "", $sData & " - WebView2")
            Case $WV2_EVENT_WEB_MESSAGE
                ; $iValue = 1 for a plain string, 0 for JSON.
                GUICtrlSetData($idInfo, "Page says: " & $sData)
        EndSwitch
    WEnd
EndFunc
```

Click Send a message to AutoIt and the label updates; click Ping the page and the page text changes. The window title follows the page title, because TITLE_CHANGED is enabled.

THE BROWSER COVERS YOUR CONTROLS

A view is a real child window, drawn above the GUI. The 40 pixels kept at the bottom of this program are not decoration: any AutoIt control placed under the view would be invisible and unclickable. Chapter 4 builds a toolbar in that band.

NEXT CHAPTER 03 Your own content

03

HTML FROM A STRING

Your own content

No server and no temporary file: the page travels inside the script, and JavaScript answers questions.

_WV2_NavigateToString displays HTML you already hold. The UDF hands it to the engine as a data: URI, so scripts, styles and the message bridge all work. A file on disk works the same way, through a file:/// URL. One consequence matters: _WV2_GetSource returns the data: URI, not your original text, so keep the HTML in a variable or a file.

Ask the page a question

_WV2_ExecuteScript runs JavaScript in the page and returns the value of the last expression as JSON text. The table below is the whole encoding for the types a beginner meets.

EXPRESSION	$SRESULT	WHY
2 + 2	4	a number is bare
document.title	"My page"	a string keeps its quotes
({ answer: 42 })	{"answer":42}	an object becomes JSON
throw new Error()	null	a throw is not a failure

Read $sResult as text. A number can be compared directly; a string arrives quoted; null means either a throw or no value. When a script can fail, return something you can test - Number(...) and String(...) make the type explicit, as the program below does.

One script runs at a time per view, and the call pumps Windows messages while it waits. The timeout defaults to $WV2_EXEC_TIMEOUT, five seconds.

Before the page runs

_WV2_AddScriptToExecuteOnDocumentCreated installs JavaScript that runs at the start of every document loaded afterwards, before the page's own code. That is where a helper object belongs.

```
Local $sId = ""
_WV2_AddScriptToExecuteOnDocumentCreated($hView, _
        "window.host = { version: '0.6', ping: function () { return 'pong'; } };", $sId)

; later, when it is no longer wanted:
_WV2_RemoveScriptToExecuteOnDocumentCreated($hView, $sId)
```

Pages already loaded keep the script; removal only affects the next documents. The program below changes the two numbers in the page and presses Read the result: the label shows what Number(...) returned, not a quoted string.

The program

```
; ============================================================
;  Your own content: HTML from a string, answers from JavaScript
; ============================================================
#include <GUIConstantsEx.au3>
#include "WebView2.au3"
Opt("MustDeclareVars", 1)

Global $hGui = GUICreate("Own content", 900, 620)
Global $idRead = GUICtrlCreateButton("Read the result", 12, 588, 150, 24)
Global $idInfo = GUICtrlCreateLabel("Change the numbers, then read.", 176, 593, 700, 20)
GUISetState(@SW_SHOW, $hGui)
Global $hView = _WV2_Create($hGui, 0, 0, -1, 580)
If $hView = 0 Then
    MsgBox(16, "WebView2", "The view could not be created: " & _WV2_GetLastErrorMessage())
    Exit 1
EndIf
; The engine takes seconds for its first page; an empty frame is what a user
; calls a freeze, so the view stays out of sight until it holds one.
_WV2_SetVisible($hView, False)

; The page is a small form. JavaScript does the work; AutoIt
; only asks for the answer when its button is clicked.
Global $sHtml = _
        "<html><body style='font-family:Segoe UI;background:#FAF4E8;color:#2E251C'>" & _
        "<div style='padding:40px'>" & _
        "<h1 style='color:#9E4B1F'>Do the maths in the page</h1>" & _
        "<input id='a' type='number' value='20' style='font-size:18px;width:90px'>" & _
        " + <input id='b' type='number' value='22' style='font-size:18px;width:90px'>" & _
        " = <b id='res' style='font-size:22px'>42</b>" & _
        "<p>What the page shows, AutoIt can read back.</p>" & _
        "<script>" & _
        "function add() {" & _
        "  var a = Number(document.getElementById('a').value);" & _
        "  var b = Number(document.getElementById('b').value);" & _
        "  document.getElementById('res').textContent = a + b;" & _
        "}" & _
        "document.getElementById('a').oninput = add;" & _
        "document.getElementById('b').oninput = add;" & _
        "</script></div></body></html>"
_WV2_NavigateToString($hView, $sHtml)
_WV2_WaitNavigation($hView)
_WV2_SetVisible($hView, True)

While 1
    Switch GUIGetMsg()
        Case $GUI_EVENT_CLOSE
            ExitLoop
        Case $idRead
            _ReadResult()
    EndSwitch
WEnd

; The window goes before the browser: a close that happens behind a hidden
; window is a close that never looks like a freeze.
GUISetState(@SW_HIDE, $hGui)
_WV2_Destroy($hView)
GUIDelete($hGui)

; ExecuteScript runs JavaScript and brings back the value of
; the last expression as JSON text. Number(...) makes it a
; bare number: 42, without quotes.
```

```
Func _ReadResult()
    Local $sJson = ""
    If _WV2_ExecuteScript($hView, _
            "Number(document.getElementById('res').textContent)", $sJson) Then
        GUICtrlSetData($idInfo, "The page computed " & $sJson)
    Else
        GUICtrlSetData($idInfo, "Failed: " & _WV2_GetLastErrorMessage())
    EndIf
EndFunc
```

NEXT CHAPTER 04 Fitting the window

04

BOUNDS, RESIZE, ZOOM

Fitting the window

The browser is a child window with its own rectangle. Place it once, then keep it in step.

_WV2_Create places the view at a fixed rectangle, or fills the rest of the client area when width and height are -1. The view never follows the window by itself: you decide when it moves. Four calls cover every layout.

SITUATION	CALL
A fixed rectangle	_WV2_Create($hGui, 0, 40, 900, 600)
The rest of the area	_WV2_Create($hGui, 0, 40, -1, -1)
Follow a toolbar	_WV2_SetBounds($hView, 0, 40, $w, $h - 40)
Cover everything	_WV2_FitToParent($hView)

Follow the window

AutoIt tells your loop when the client area changed. Get the new size, then hand the view the rectangle below the toolbar. Keep the toolbar height in one constant, so the two numbers cannot drift apart.

```
Case $GUI_EVENT_RESIZED, $GUI_EVENT_MAXIMIZE, $GUI_EVENT_RESTORE
    _Layout()

Func _Layout()
    Local $aSize = WinGetClientSize($g_hGui)
    _WV2_SetBounds($g_hView, 0, $TOOLBAR, $aSize[0], $aSize[1] - $TOOLBAR)
EndFunc
```

Zoom, hide, show

```
_WV2_SetZoom($hView, 1.25)        ; 0.2 to 10; _WV2_GetZoom reads it back
_WV2_SetVisible($hView, False)    ; hide the browser, keep the window
_WV2_FitToParent($hView)          ; fill the whole client area
```

A hidden view keeps running, but its snapshot is empty or black: pixels exist only once the engine has painted. Capture after the page has loaded and the view is visible.

The program

```
; ============================================================
;  A browser with a toolbar that follows the window
;  Back, Forward, Reload, an address field and a status line.
; ============================================================
#include <GUIConstantsEx.au3>
#include "WebView2.au3"

Opt("MustDeclareVars", 1)

; The toolbar band is never covered by the browser.
Global Const $TOOLBAR = 40

Global $g_hGui = 0, $g_hView = 0
Global $g_idBack = 0, $g_idForward = 0, $g_idReload = 0
Global $g_idUrl = 0, $g_idGo = 0, $g_idStatus = 0

$g_hGui = GUICreate("Mini browser", 1000, 700)
$g_idBack = GUICtrlCreateButton("Back", 8, 8, 64, 24)
$g_idForward = GUICtrlCreateButton("Forward", 80, 8, 64, 24)
$g_idReload = GUICtrlCreateButton("Reload", 152, 8, 64, 24)
$g_idUrl = GUICtrlCreateInput("", 232, 8, 560, 24)
$g_idGo = GUICtrlCreateButton("Go", 800, 8, 48, 24)
$g_idStatus = GUICtrlCreateLabel("Ready", 860, 12, 130, 20)
GUISetState(@SW_SHOW, $g_hGui)

; The view starts below the toolbar, on the first client size.
Global $aClient = WinGetClientSize($g_hGui)
$g_hView = _WV2_Create($g_hGui, 0, $TOOLBAR, $aClient[0], $aClient[1] - $TOOLBAR)
If $g_hView = 0 Then
    MsgBox(16, "WebView2", "The view could not be created: " & _WV2_GetLastErrorMessage())
    Exit 1
EndIf

; Left alone, a target="_blank" link makes the runtime open a window of its own, outside
; this toolbar and outside its history. Enabling the event cancels that window and queues
; the address instead, so _DrainEvents can keep the page in this view.
_WV2_EnableEvent($g_hView, $WV2_EVENT_NEW_WINDOW)

; A local page with an anchor, so Back has somewhere to go. The view stays out of
; sight until it holds that page: a browser window with nothing in it looks broken,
; and the first start of the engine takes seconds.
_WV2_SetVisible($g_hView, False)
Global $sHtml = _
        "<html><body style='font-family:Segoe UI;background:#FAF4E8;color:#2E251C'>" & _
        "<div style='padding:40px'>" & _
        "<h1 style='color:#9E4B1F'>Mini browser</h1>" & _
        "<p>Type any address in the field above, then press Go.</p>" & _
        "<p><a href='#details'>Jump to the details</a></p>" & _
        "<div style='height:900px'></div>" & _
        "<h2 id='details'>The details</h2>" & _
        "<p>Back returns to the top of this page.</p>" & _
        "</div></body></html>"
_WV2_NavigateToString($g_hView, $sHtml)
_WV2_WaitNavigation($g_hView)
_WV2_SetVisible($g_hView, True)

While 1
    Switch GUIGetMsg()
        Case $GUI_EVENT_CLOSE
            ExitLoop
        Case $GUI_EVENT_RESIZED, $GUI_EVENT_MAXIMIZE, $GUI_EVENT_RESTORE
```

```
            _Layout()
        Case $g_idBack
            _WV2_GoBack($g_hView)
        Case $g_idForward
            _WV2_GoForward($g_hView)
        Case $g_idReload
            _WV2_Reload($g_hView)
        Case $g_idGo
            _Go()
    EndSwitch
    _DrainEvents()
WEnd

; Hide the window before releasing the browser, so the hundreds of milliseconds
; the browser needs to leave are not spent staring at an empty frame.
GUISetState(@SW_HIDE, $g_hGui)
_WV2_Destroy($g_hView)
GUIDelete($g_hGui)

; Fits the browser to the new client area after a resize.
Func _Layout()
    Local $aSize = WinGetClientSize($g_hGui)
    If Not IsArray($aSize) Or $aSize[0] <= 0 Then Return
    _WV2_SetBounds($g_hView, 0, $TOOLBAR, $aSize[0], $aSize[1] - $TOOLBAR)
EndFunc

; Reads the address field and asks the view to load it.
Func _Go()
    Local $sUrl = StringStripWS(GUICtrlRead($g_idUrl), 3)
    If $sUrl <> "" Then _WV2_Navigate($g_hView, $sUrl)
EndFunc

; Keeps the status line in step with the navigations, and the pages the site wanted to
; open elsewhere inside this view.
Func _DrainEvents()
    Local $sData = "", $iValue = 0
    While 1
        Local $iEvent = _WV2_GetEvent($g_hView, $sData, $iValue)
        If $iEvent <= 0 Then ExitLoop
        If $iEvent = $WV2_EVENT_NEW_WINDOW And $sData <> "" Then
            GUICtrlSetData($g_idUrl, $sData)
            _WV2_Navigate($g_hView, $sData)
            ContinueLoop
        EndIf
        If $iEvent = $WV2_EVENT_NAVIGATION_COMPLETED Then
            If $iValue = 1 Then
                GUICtrlSetData($g_idStatus, "Loaded")
            Else
                GUICtrlSetData($g_idStatus, "Error " & $iValue)
            EndIf
        EndIf
    WEnd
EndFunc
```

Back and Forward go straight to the engine, Reload asks for the same document, Go reads the address field. The status line follows NAVIGATION_COMPLETED. NEW_WINDOW is enabled for one reason: without it a target=_blank link opens a window of its own, outside this toolbar and outside its history, and _DrainEvents reopens the address here instead.

NEXT CHAPTER 05 Recipes

05

NINE THINGS YOU WILL WANT

Recipes

Snapshot, PDF, downloads, search, cookies, your own host name, dialogs, the icon of the page and the settings: each one is a short call with one trap worth knowing.

Recipes 1, 2, 3 and 9 were already in the 0.3 booklet - 2 and 3 have grown since. The five others came with the last two lots. They are all one or two calls, and each one carries a detail that costs an afternoon when you meet it cold.

1. Save what the page shows

```
Local $sFile = @ScriptDir & "\page.png"
If _WV2_CapturePreview($hView, $sFile) Then
    MsgBox(64, "Snapshot", "Saved " & $sFile)
EndIf
_WV2_CapturePreview($hView, @ScriptDir & "\page.jpg", $WV2_IMAGE_JPEG)
```

The image is built in memory by the engine and written by the UDF; no temporary file is involved. A view that is hidden or has never painted gives an empty or black image, which is what Chromium reports. Capture after _WV2_WaitNavigation.

2. Print the page to PDF, with a say

```
Local $sPdf = @ScriptDir & "\page.pdf"
If _WV2_PrintToPdf($hView, $sPdf) Then
    MsgBox(64, "PDF", "Saved " & $sPdf)
Else
    MsgBox(16, "PDF", _WV2_GetLastErrorMessage())
EndIf
```

Defaults: portrait, the page's own paper size, no header or footer, backgrounds left out. The call pumps messages while the engine prints. A missing folder is $WV2_ERR_FILE; a refusal from the runtime is $WV2_ERR_CALL, with the HRESULT in @extended.

```
Local $hPrint = 0
_WV2_PrintSettingsCreate($hView, $hPrint)
_WV2_PrintSettingsSet($hPrint, "Orientation", $WV2_PRINT_LANDSCAPE)
_WV2_PrintSettingsSet($hPrint, "ShouldPrintBackgrounds", True)
_WV2_PrintSettingsSet($hPrint, "MarginTop", 0.4)          ; inches
_WV2_PrintToPdfEx($hView, @ScriptDir & "\wide.pdf", $hPrint)
_WV2_PrintSettingsDestroy($hPrint)

Local $bPdf = ""
_WV2_PrintToPdfStream($hView, $bPdf)   ; the PDF in memory, not on disk
_WV2_ShowPrintUI($hView, $WV2_PRINT_UI_SYSTEM)   ; the dialog of Windows
```

Eleven names fill a settings object: Orientation, ScaleFactor, PageWidth, PageHeight, the four margins, the three ShouldPrint flags, HeaderTitle and FooterUri. An unknown name is $WV2_ERR_ARG. Eight objects may be live at once, and each belongs to the view that made it - destroy it before destroying the view.

3. Choose where downloads land, and steer them

```
Local $sFolder = @ScriptDir & "\downloads"
If Not FileExists($sFolder) Then DirCreate($sFolder)
_WV2_SetDownloadFolder($hView, $sFolder)      ; the folder must exist
_WV2_SetDownloadFilter($hView, "\.exe$")      ; matching URIs are refused
```

Both calls enable $WV2_EVENT_DOWNLOAD_STARTING by themselves. Every download then reaches the queue: _WV2_GetEvent gives the URI, with $iValue = 1 when the filter refused it. A refused download never reaches the disk; the engine is told before a byte is written. The file keeps the name chosen by the server - only the folder changes.

```
Local $iN = _WV2_GetDownloadCount($hView)
Local $aDl[1]
For $i = 1 To $iN
    If _WV2_GetDownload($hView, $i, $aDl) Then
        ; $aDl[0] uri  [1] path  [2] state  [3] received  [4] total
        ;     [5] end time  [6] can resume  [7] interrupt reason
    EndIf
Next

_WV2_DownloadPause($hView, 1)
_WV2_DownloadResume($hView, 1)
_WV2_DownloadCancel($hView, 1)
```

Each view remembers its last eight downloads. State and progress also arrive in the queue as $WV2_EVENT_DOWNLOAD_STATE and _PROGRESS. Trap worth knowing: a download made from a blob URL stays in the running state while the page that started it lives, and its total is unknown until the end - watch the file, not the state.

4. Search inside the page

```
_WV2_EnableEvent($hView, $WV2_EVENT_FIND)
_WV2_FindStart($hView, "webview", True, False)   ; case, whole word
_WV2_FindNext($hView)
_WV2_FindPrevious($hView)
Local $iMatches = _WV2_FindCount($hView)
Local $iActive  = _WV2_FindActive($hView)        ; -1 when none
_WV2_FindStop($hView)
```

The engine highlights every match and scrolls to the active one; the default find bar of the runtime is turned off, so the page is the only thing the user sees. Two traps: the search object is built on the first call, which can take longer than the five-second default timeout on a cold view - pass a bigger $iTimeoutMs the first time; and a runtime older than 1.0.3240 has no search at all, which answers $WV2_ERR_CALL with E_NOINTERFACE in @extended.

5. Read, write and clear cookies

```
Local $aCookies[1][1]
Local $iN = _WV2_GetCookies($hView, "https://app.local/", $aCookies)
For $i = 0 To $iN - 1
    ; [0] name  [1] value  [2] domain  [3] path  [4] expires
    ; [5] httpOnly  [6] secure  [7] session  [8] sameSite
Next

_WV2_SetCookie($hView, "theme", "dark", "app.local")
_WV2_DeleteCookie($hView, "theme", "https://app.local/")
_WV2_ClearCookies($hView)
```

The jar belongs to the user data folder, not to the view: two views on the same folder share it, and a page already open keeps what it read until it is reloaded. Three traps. A cookie needs a real origin - file:// and NavigateToString cannot store one. SameSite None is silently dropped unless $bSecure is True. And an expiry further out than 400 days is clamped by the engine.

```
_WV2_ClearBrowsingData($hView, $WV2_DATA_ALL)   ; history, downloads, cookies, cache
_WV2_Reload($hView)
```

_WV2_ClearCookies only takes cookies. The four kinds of _WV2_ClearBrowsingData are $WV2_DATA_BROWSING_HISTORY, _DOWNLOADS, _COOKIES and _CACHE, or $WV2_DATA_ALL. localStorage is not one of them: clear it from the page.

6. Give your own content a real host name

```
_WV2_SetVirtualHostNameToFolderMapping($hView, "app.local", @ScriptDir & "\site")
_WV2_Navigate($hView, "https://app.local/index.html")
```

The browser answers from the folder: no web server, no port, nothing extracted to a temporary directory. The page is then a true web origin, so storage, fetch, workers and cookies behave as they do online - which is what makes recipe 5 work with your own files. Nothing is cached, so a file edited on disk is served as it is on the next request. The mapping applies to the next navigation; call it before _WV2_Navigate.

7. Answer alert, confirm and prompt

```
_WV2_SetDialogHandler($hView, "_MyDialog")   ; before the navigation

Func _MyDialog($iHandle, $iKind, $sMessage, $sDefault, $sUri)
    ; $iKind: $WV2_DIALOG_ALERT, _CONFIRM, _PROMPT or _BEFORE_UNLOAD
    Return MsgBox(YesNoCancel, $sUri, $sMessage) = 6   ; 6 = Yes
EndFunc
```

Return True to accept the dialog, a string to answer a prompt with that text, False to dismiss it. The page stands still until your function returns, so a MsgBox is fine and a second call into the same view is refused with $WV2_ERR_BUSY. Install the handler before navigating - the setting that lets it run is read when the document loads - and every dialog also reaches the queue as $WV2_EVENT_SCRIPT_DIALOG.

8. Show the icon of the page

```
Local $sUri = ""
_WV2_GetFaviconUri($hView, $sUri)
Local $bIcon = ""
If _WV2_GetFavicon($hView, $bIcon) Then
    FileWrite(@ScriptDir & "\icon.png", $bIcon)
EndIf
```

The bytes are PNG or JPEG whatever the site published - the runtime converts them - and nothing touches the disk on the way. A page that declares no icon is not a failure: the runtime keeps answering with the last icon the view knew, so read _WV2_GetFaviconUri first when the absence matters. The icon arrives a moment after the navigation ends; $WV2_EVENT_FAVICON_CHANGED says when.

9. Turn settings off, DevTools on

```
_WV2_SettingWrite($hView, "AreDevToolsEnabled", True)
_WV2_SettingWrite($hView, "AreDefaultContextMenusEnabled", False)
Local $bJs = _WV2_SettingRead($hView, "IsScriptEnabled")   ; 1 or 0
```

Nine names are accepted, in any case and with or without spaces: IsScriptEnabled, IsWebMessageEnabled, AreDefaultScriptDialogsEnabled, IsStatusBarEnabled, AreDevToolsEnabled, AreDefaultContextMenusEnabled, AreHostObjectsAllowed, IsZoomControlEnabled, IsBuiltInErrorPageEnabled. A name outside the list is $WV2_ERR_ARG.

With DevTools enabled, F12 opens the inspector inside the view. And when the inspector is not enough, the protocol behind it is one call away: _WV2_CallDevToolsProtocolMethod($hView, "Page.captureScreenshot", '{"format":"png"}', $sResult) brings the answer as JSON text, and _WV2_SetDevToolsProtocolEvent names one event to receive as $WV2_EVENT_DEVTOOLS.

> **NEW IN THESE RECIPES**
> Recipe 6 came with the 0.4 lot; recipes 4, 5, 7 and 8 with 0.5. The guide that ships beside this booklet explains what each one costs the wrapper: a handler object in the same heap, and a vtable rank counted from Microsoft's header.

NEXT CHAPTER 06 When it goes wrong

06

ERRORS, TRAPS, LOOKUP

When it goes wrong

Every failure is a return value. The UDF never shows a dialog and never writes to the console.

The UDF reports; your script decides. A failed call returns 0 or -1, sets @error to a $WV2_ERR_* code, and often puts the raw HRESULT in @extended. _WV2_GetLastErrorMessage() turns the last failure into a sentence such as Method call failed in ICoreWebView2::Navigate (HRESULT 0x80070057).

Thirteen codes

CONSTANT	VALUE	MEANING
$WV2_ERR_NONE	0	no error
$WV2_ERR_HANDLE	1	invalid or destroyed handle
$WV2_ERR_ARG	2	bad argument for this call
$WV2_ERR_RUNTIME	3	WebView2 runtime not found on this machine
$WV2_ERR_COM	4	COM initialisation or interface wrapper failed
$WV2_ERR_ENV	5	environment creation failed
$WV2_ERR_CONTROLLER	6	controller creation failed
$WV2_ERR_NOT_READY	7	instance not usable yet
$WV2_ERR_CALL	8	a WebView2 method returned a failure HRESULT
$WV2_ERR_TIMEOUT	9	the operation did not finish in time
$WV2_ERR_FULL	10	sixteen live views already
$WV2_ERR_FILE	11	a file given to the UDF could not be written
$WV2_ERR_BUSY	12	a handler of this view is running; the call would re-enter it

THE ONE YOU CANNOT FIX IN CODE

$WV2_ERR_RUNTIME means the machine has no WebView2 runtime. The UDF never installs or downloads anything; the README says where the runtime comes from and how to check for it.

Seven traps

- **Blocking the loop.** Events are delivered by the message loop. A long Sleep or a modal dialog stops the engine just as it stops your window: use _WV2_WaitEvent, _WV2_WaitNavigation or _WV2_ExecuteScript, which pump while they wait.
- **Controls under the browser.** A view is a child window and covers AutoIt controls that overlap it. Reserve a band (chapter 4).
- **Creating on one thread, destroying on another.** The apartment is initialised by _WV2_Create in the calling thread. Keep the whole life of the view there.
- **Expecting a string from ExecuteScript.** The result is JSON text: a string keeps its quotes, a number is bare, a

throw gives null. _WV2_ExecuteScriptEx answers with the exception text instead.

- **Reading -1 as a size.** In _WV2_Create, -1 means the rest of the client area, not the size of the parent window and not a default.
- **Calling back into the same view.** While a resource, dialog, key or menu handler of a view is running, a call that would pump messages into that same view is refused with $WV2_ERR_BUSY. Answer at once, set a flag, and do the rest from the main loop.
- **Judging a page by its first search.** The search object is built on the first _WV2_FindStart of a view, and that one call can outlast the five-second default. Give the first search a longer timeout.

Cheat sheet

All seventy-six functions, in the order you are likely to need them. Full signatures, parameters and remarks are in the #FUNCTION# header of each one in WebView2.au3.

Window and lifecycle

FUNCTION	WHAT IT DOES
_WV2_Create	Creates a view in a GUI; returns a handle from 1 to 16
_WV2_Destroy	Releases the engine objects; hide the window first
_WV2_DestroyAll	Releases every live view at once, from the message loop
_WV2_IsReady	1 when the view can be used
_WV2_GetRuntimePath	Full path of the runtime DLL, empty when it is missing
_WV2_GetBrowserVersion	Version of the engine behind this view
_WV2_GetLastErrorMessage	Text of the last failure, empty after a success

Navigation

FUNCTION	WHAT IT DOES
_WV2_Navigate	Loads a URL; the navigation itself is asynchronous
_WV2_NavigateToString	Loads HTML carried by the script
_WV2_WaitNavigation	Pumps until the navigation ends; @extended is the web error
_WV2_IsNavigationComplete	Polls the same state without waiting
_WV2_Reload	Reloads the current document
_WV2_GoBack	Moves one entry back in history
_WV2_GoForward	Moves one entry forward
_WV2_Stop	Stops the current navigation
_WV2_SetNavigationFilter	PCRE; matching URLs are refused

History, bounds, zoom

FUNCTION	WHAT IT DOES
_WV2_CanGoBack	1 if Back would move
_WV2_CanGoForward	1 if Forward would move
_WV2_SetBounds	Places the view in the parent client area
_WV2_GetBounds	Reads its rectangle, by reference
_WV2_FitToParent	Covers the whole client area
_WV2_SetVisible	Shows or hides the browser
_WV2_SetZoom	Zoom factor, 0.2 to 10
_WV2_GetZoom	Current zoom factor

Page, scripts, events

FUNCTION	WHAT IT DOES
_WV2_GetTitle	Document title, by reference
_WV2_GetSource	Current URL, by reference
_WV2_ExecuteScript	Runs JavaScript; returns the value as JSON text
_WV2_AddScriptToExecuteOnDocumentCreated	Installs a script for the next documents
_WV2_RemoveScriptToExecuteOnDocumentCreated	Removes it by the id it returned
_WV2_EnableEvent	Subscribes to one event type
_WV2_GetEvent	Takes the oldest queued event; 0 when empty
_WV2_WaitEvent	Waits for one event type, pumping messages
_WV2_PostWebMessage	Sends JSON text to the page
_WV2_PostWebMessageAsString	Sends a plain string to the page

Files and settings

FUNCTION	WHAT IT DOES
_WV2_CapturePreview	Writes the visible page as PNG or JPEG
_WV2_PrintToPdf	Writes the page as PDF
_WV2_SetDownloadFolder	Folder where downloads land; it must exist
_WV2_SetDownloadFilter	PCRE; matching downloads are refused
_WV2_ClearCookies	Empties the cookie jar of the profile
_WV2_SettingRead	Reads one of the nine page settings
_WV2_SettingWrite	Writes one of the nine page settings

Your own content - new in 0.4

FUNCTION	WHAT IT DOES
_WV2_SetResourceHandler	Names the function that answers the requests of the page
_WV2_AddResourceFilter	Adds a URI pattern and a $WV2_RES_* kind
_WV2_RemoveResourceFilter	Takes one pattern back
_WV2_SetVirtualHostNameToFolderMapping	Serves a folder under https://yourhost/
_WV2_ClearVirtualHostNameToFolderMapping	Removes one host name

Search in page - new in 0.5

FUNCTION	WHAT IT DOES
_WV2_FindStart	Searches; case, whole word and highlight are arguments
_WV2_FindNext	Steps to the next match
_WV2_FindPrevious	Steps back one
_WV2_FindStop	Ends the search and clears the highlights
_WV2_FindCount	How many matches there are
_WV2_FindActive	Which one is current, -1 when none

Printing, with a say

FUNCTION	WHAT IT DOES
_WV2_PrintSettingsCreate	Builds a settings object for this view
_WV2_PrintSettingsSet	Writes one of the eleven names
_WV2_PrintSettingsGet	Reads one of them back
_WV2_PrintSettingsDestroy	Releases the object
_WV2_PrintToPdfEx	Prints with those settings
_WV2_PrintToPdfStream	Brings the PDF back in memory
_WV2_ShowPrintUI	Opens the preview of the browser or the dialog of Windows

Downloads and cookies

FUNCTION	WHAT IT DOES
_WV2_GetDownloadCount	How many downloads the view remembers
_WV2_GetDownload	URI, path, state, bytes, resume and interrupt reason
_WV2_DownloadPause	Pauses one, by index
_WV2_DownloadResume	Starts a paused one again
_WV2_DownloadCancel	Stops one for good

FUNCTION	WHAT IT DOES
_WV2_GetCookies	The jar as seen from one URI
_WV2_SetCookie	Writes name, value, domain, path, expiry, SameSite
_WV2_DeleteCookie	Takes every cookie of that name for that URI

Handlers, icon, protocol

FUNCTION	WHAT IT DOES
_WV2_SetDialogHandler	Answers alert, confirm, prompt and beforeunload
_WV2_SetAcceleratorHandler	Sees Ctrl+P and friends before the browser does
_WV2_SetContextMenuHandler	Takes the menu the page asks for
_WV2_GetFaviconUri	The address the page declares as its icon
_WV2_GetFavicon	The icon as PNG or JPEG bytes, in memory
_WV2_ClearBrowsingData	History, downloads, cookies and cache, in any set
_WV2_ExecuteScriptEx	Result, whether it worked, and the exception text
_WV2_CallDevToolsProtocolMethod	Any DevTools method; the answer as JSON
_WV2_SetDevToolsProtocolEvent	Puts one DevTools event into the queue

The names you will type

CONSTANT	WHAT IT IS FOR
$WV2_EVENT_*	the seventeen kinds _WV2_GetEvent answers, from _NAVIGATION_STARTING to _DEVTOOLS
$WV2_ERR_*	the thirteen @error codes, 0 to 12; @extended carries the raw HRESULT
$WV2_RES_*	one name per kind of request a resource filter matches, and $WV2_RES_ALL
$WV2_DL_IN_PROGRESS, _COMPLETED, _INTERRUPTED	the state of a download
$WV2_DIALOG_ALERT, _CONFIRM, _PROMPT, _BEFORE_UNLOAD	the kind a script dialog arrives as
$WV2_KEYEVENT_DOWN, _UP, _SYS_DOWN, _SYS_UP	why an accelerator was seen
$WV2_DATA_COOKIES, _CACHE, _BROWSING_HISTORY, _DOWNLOADS, _ALL	what _WV2_ClearBrowsingData may take
$WV2_PRINT_PORTRAIT, _LANDSCAPE	the Orientation of a print settings object
$WV2_PRINT_UI_BROWSER, _SYSTEM	which preview _WV2_ShowPrintUI opens
$WV2_CREATE_TIMEOUT, _NAV_TIMEOUT	how long create and navigation wait
$WV2_EXEC_TIMEOUT, _PRINT_TIMEOUT	how long a script and a print wait

THE REST OF THE PACKAGE

You are ready

Everything past this point is a variation on the four programs: the same handle, the same message loop, and the same rule that a failure is a return value, never a dialog.

IN THE PACKAGE

README.md	installation, the API in one page, the limits
WebView2_Developer_Guide.pdf	the COM machinery under this API
WebView2_Example_GUI.au3	a complete browser, with a self test
WebView2.au3	every function carries a #FUNCTION# header

Questions, bug reports and updates belong in the forum thread for this UDF. The README repeats the API on one page; the Developer Guide explains what runs underneath it.

_WV2_GetLastErrorMessage() is the first thing to print when a call fails: it names the method and carries the HRESULT that @extended shows as a number.

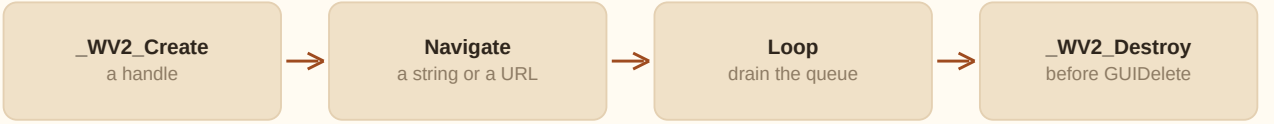


Figure 2. *Every program in this booklet is this sequence. What changes between them is what happens in the middle: a message, a script, a download, a page printed to PDF.*

END OF THE QUICK START WebView2 UDF 0.6.0 - Numeric1 - MIT

Everything here was measured on Windows 10 with AutoIt 3.3.16.1 (x86); the four programs are next to the UDF in the zip and run as printed. Beside them, the example runs its own self-test: 297 checks in 37 sections, without network access.