Development Kit User Manual (V4.0)

Make your own software with our development kit and the CHROMATEQ DMX Interfaces

Windows (x64) and macOS (Apple Silicon and Intel) · November 2026

Overview

Our development kit is a dynamic library that lets your own software drive the CHROMATEQ USB-DMX interfaces: open the interfaces, send and receive DMX universes, set the direction of each DMX port and the DMX timings, read the firmware version.

On Windows, the library is called `DevKit.dll` (64-bit DLL, Windows 10 or later). On macOS, it is called `libDevKit.dylib` (universal binary for Apple Silicon and Intel, macOS 13 or later). Both come with a test application, **DevKitTest**, which drives every function of the library without writing any code.

The interface is a set of plain C functions: it can be used from C and C++ (Visual Studio, MinGW, Clang, Xcode, Qt Creator), and from any language able to call a C library: C#, Delphi, Python, Swift...

What's new in version 4

- **One API for all the interfaces**: the interfaces of the previous generation (protocol V1) and of the current generation (protocol V2) are driven the same way. The library detects the protocol.
- **Handles**: an open interface is identified by a handle, which stays valid until the interface is closed, whatever the other interfaces opened or closed.
- **DMX ports**: the direction (off, input, output) of each port is set individually, and the library tells which directions each port supports.
- **DMX timings in physical units**: microseconds, milliseconds and kbps.
- **Plain C**: pointers instead of C++ references, `int` instead of `bool`, cdecl calling convention. Every function can be called from any thread, and your program does not need to be a Qt application.
- macOS: universal binaries, signed and notarized.

Version 4 is not source compatible with version 3: see *Migrating from version 3* at the end of this manual.

Contents of the kit

Windows

Path	Description
`include/devkit.h`	The interface of the library. Every function is documented in this header.
`lib/DevKit.dll`	The library (64-bit).
`lib/DevKit.lib`	Import library for Visual Studio (MSVC, Clang-cl).
`lib/libDevKit.dll.a`	Import library for MinGW / GCC.
`lib/DevKit.def`	Exported functions, to create an import library for another toolchain.
`lib/libusb-1.0.dll`	libusb 1.0.29, used by the library to talk to the interfaces (LGPL 2.1).
`lib/Qt6*.dll`, `lib/lib*.dll`	Qt 6 libraries used by DevKit.dll (LGPL 3), and the MinGW runtime.
`DevKitTest/DevKitTest.exe`	Test application, ready to run.
`Driver/`	WinUSB driver of the interfaces, signed.
`README.md`	Quick start.

macOS

Path	Description
`include/devkit.h`	The interface of the library.
`lib/libDevKit.dylib`	The library (universal; install name `@rpath/libDevKit.dylib`).
`lib/libusb-1.0.0.dylib`	libusb 1.0.29, used by the library to talk to the interfaces (LGPL 2.1).
`lib/Qt*.framework`	Qt 6 frameworks used by the library (LGPL 3).
`DevKitTest.app`	Test application, signed and notarized.
`README.md`	Quick start.

Installation

Windows: driver

The interfaces are accessed through WinUSB. The driver must be installed once on each computer, with administrator rights: run `Driver/usb dmx drivers.exe`, or `Driver/Files/installDriver.bat` (`pnputil /add-driver usb-dmx-drivers.inf /install`). If CHROMATEQ software is already installed on the computer, the driver is already there.

Then plug in the interface and start `DevKitTest/DevKitTest.exe` to check that it opens.

macOS

No driver is needed. Copy `DevKitTest.app` anywhere, plug in the interface and start the application to check that it opens.

Concepts

Interfaces and handles

`cqdevkitOpenUsbDevice()` opens the next interface that is not open yet and returns its **handle** (0 or more), or -1 when no other interface can be opened. Call it again to open several interfaces. Every other function takes this handle; it stays valid until `cqdevkitCloseDevice()`, `cqdevkitCloseAllDevices()` or `cqdevkitDestroy()`.

Interfaces of the previous generation (protocol V1) and of the current generation (protocol V2) can not be open at the same time: the protocol is set by the first interface opened.

DMX ports

An interface has `cqdevkitGetNumberOfPorts()` DMX ports, numbered from 0 (port A). Each port is off, an input (`DEVKIT_PORT_IN`) or an output (`DEVKIT_PORT_OUT`). After opening, every port that can be an output is an output.

`cqdevkitGetPortCapabilities()` tells which directions a port supports. On the current generation, every port can be an input or an output. On the previous generation, only the last DMX connector can be an input, and an input requires all the other ports to be outputs. When the licence of such an interface covers fewer universes than its connectors, its DMX input appears as an extra, input-only port after the output ports.

Constant	Value	Meaning
`DEVKIT_PORT_OFF`	0	Port not used.
`DEVKIT_PORT_IN`	1	DMX input: read the frames with `cqdevkitReadDmxUniverse()`.
`DEVKIT_PORT_OUT`	2	DMX output: send the frames with `cqdevkitWriteDmxUniverse()`.
`DEVKIT_PORT_CAN_OUT`	0x01	Capability: the port can be an output.
`DEVKIT_PORT_CAN_IN`	0x02	Capability: the port can be an input.
`DEVKIT_DMX_CHANNELS`	512	Size of a DMX buffer, in bytes.

DMX frames

A DMX universe is a buffer of `DEVKIT_DMX_CHANNELS` (512) bytes, one byte per channel (channel 1 at index 0). Send the output frames regularly, for example every 25 to 40 ms, from a timer or a dedicated thread. Read the input ports just as regularly: `cqdevkitReadDmxUniverse()` returns the last frame received since the previous call.

Return values, threads and blocking calls

- Functions returning a status return **1 on success, 0 on failure**.
- Every function can be called from any thread: the calls are serialized by the library.
- `cqdevkitOpenUsbDevice()` and the configuration functions (port directions, DMX timings, firmware version, key) talk to the interface and **block for up to a few seconds**. Do not call them from your user interface thread, or from the thread that sends the DMX frames.
- Your program does not need to be a Qt application, even though the library uses Qt internally.

Function reference

All the functions are declared in `devkit.h`, inside `extern "C"`, with the cdecl calling convention (`DEVKIT_CALL`).

Library

`void cqdevkitCreate(void);`

Initializes the library. Call it once, before any other function.

`void cqdevkitDestroy(void);`

Closes the interfaces still open and releases the library. All the handles become invalid.

Interfaces

`int cqdevkitGetNumberOfUsbDevices(void);`

Number of CHROMATEQ USB interfaces plugged into the computer, open or not.

`int cqdevkitOpenUsbDevice(const unsigned char* ucKey, int size);`

Opens the next USB interface that is not open yet, with all its ports as outputs (when they can be).

Parameters

- `ucKey, size`: public key the interface must hold, up to 256 bytes (letters and numbers), or `NULL` / 0 for no check. Only the interfaces of the previous generation (protocol V1) hold a key: it is ignored by the current generation.

Returns the handle of the interface (0 or more), or -1 if no other interface could be opened.

- Blocks for up to a few seconds.

`void cqdevkitCloseDevice(int hDevice);`

Closes an interface. Its handle becomes invalid.

`void cqdevkitCloseAllDevices(void);`

Closes all the open interfaces.

`int cqdevkitIsDeviceConnected(int hDevice);`

1 if the interface is open and connected, 0 otherwise (unplugged, closed, or invalid handle). An interface that was unplugged is not reopened automatically: close it and open it again.

Ports

`int cqdevkitGetNumberOfPorts(int hDevice);`

Number of DMX ports of the interface, 0 on error.

`int cqdevkitGetPortCapabilities(int hDevice, int iPort);`

Directions supported by a port: `DEVKIT_PORT_CAN_OUT` and/or `DEVKIT_PORT_CAN_IN` (off is always possible). 0 on error.

`int cqdevkitGetPortDirection(int hDevice, int iPort);`

Current direction of a port: `DEVKIT_PORT_OFF`, `DEVKIT_PORT_IN` or `DEVKIT_PORT_OUT`, -1 on error.

`int cqdevkitSetPortsDirection(int hDevice, const int* nDirections, int nPorts);`

Sets the direction of the ports 0 to nPorts - 1 and writes the configuration to the interface. The other ports keep their direction.

Parameters

- `nDirections`: array of nPorts `DEVKIT_PORT_*` values.

Returns 1 on success. 0, and the current configuration is kept, if a port does not support its direction or if the interface does not support the combination (previous generation: an input requires all the other ports as outputs).

- Blocks for a few hundred milliseconds.

`int cqdevkitSetPortDirection(int hDevice, int iPort, int nDirection);`

Same as `cqdevkitSetPortsDirection()`, for a single port.

Configuration

`int cqdevkitReadFirmwareVersion(int hDevice, unsigned char* ucMajorVersion, unsigned char* ucAnnualVersion, unsigned char* ucMinorVersionH, unsigned char* ucMinorVersionL);`

Reads the firmware version of the interface, displayed as Major.Annual.MinorH.MinorL.

`int cqdevkitReadKey(int hDevice, unsigned char* ucKey, int size);`

Reads the public key of the interface (up to 256 bytes) into ucKey (size bytes at most).

Returns 1 on success, 0 on failure or with an interface of the current generation (no key).

`int cqdevkitWriteKey(int hDevice, const unsigned char* ucKey, int size);`

Writes the public key of the interface (up to 256 bytes, letters and numbers, completed with zeros). An empty key removes the key.

Returns 1 on success, 0 on failure or with an interface of the current generation (no key).

`int cqdevkitReadDmxConfig(int hDevice, unsigned short* usBreak, unsigned short* usMaB, unsigned short* usPeriod, unsigned short* usBitRate);`

Reads the DMX timings of the outputs.

Parameters

- `usBreak`: duration of the break, in microseconds.
- `usMaB`: duration of the mark after break, in microseconds.
- `usPeriod`: period between two DMX frames, in milliseconds.
- `usBitRate`: bit rate, in kbps: 255, 250 (DMX standard) or 245.

`int cqdevkitWriteDmxConfig(int hDevice, unsigned short usBreak, unsigned short usMaB, unsigned short usPeriod, unsigned short usBitRate);`

Writes the DMX timings of the outputs (same units as `cqdevkitReadDmxConfig()`).

- The interfaces of the previous generation take the nearest values they support: read the configuration back to know the values in use.

DMX

`int cqdevkitReadDmxUniverse(int hDevice, int iPort, unsigned char* ucDmxArray);`

Copies into ucDmxArray (512 bytes) the last DMX frame received on an input port.

Returns 1 if a new frame was received since the previous call, 0 otherwise (no new frame, port not an input, invalid handle). 0 is not an error: call it again at the next period.

`int cqdevkitWriteDmxUniverse(int hDevice, int iPort, const unsigned char* ucDmxArray);`

Sends a DMX frame (512 bytes) on an output port.

Returns 1 on success, 0 on failure (port not an output, interface unplugged, invalid handle).

Using

C / C++

```c
#include <string.h>
#include "devkit.h"

void Example(void)
{
    unsigned char ucDmxOut[DEVKIT_DMX_CHANNELS];
    unsigned char ucDmxIn[DEVKIT_DMX_CHANNELS];

    cqdevkitCreate(); // always call cqdevkitCreate() before the other functions

    int hDevice = cqdevkitOpenUsbDevice(NULL, 0); // -1 if no interface could be opened
    if (hDevice >= 0)
    {
        // Optional: port B as an input, if the interface supports it
        if (cqdevkitGetNumberOfPorts(hDevice) > 1 &&
            (cqdevkitGetPortCapabilities(hDevice, 1) & DEVKIT_PORT_CAN_IN))
            cqdevkitSetPortDirection(hDevice, 1, DEVKIT_PORT_IN);

        memset(ucDmxOut, 255, sizeof(ucDmxOut)); // all the channels at full

        for (int i = 0; i < 1000; i++)
        {
            cqdevkitWriteDmxUniverse(hDevice, 0, ucDmxOut); // port A

            if (cqdevkitReadDmxUniverse(hDevice, 1, ucDmxIn)) // port B: new frame received
            {
                // ... use ucDmxIn
            }

            Sleep(25); // Windows (usleep(25000) on macOS): one frame every 25 ms
        }

        cqdevkitCloseDevice(hDevice);
    }

    cqdevkitDestroy();
}
```

C#

No import library is needed. The process must be 64-bit (on Windows, *Platform target: x64*, or *Any CPU* without *Prefer 32-bit*).

```
using System.Runtime.InteropServices;

static class DevKit
{
    const string Dll = "DevKit.dll"; // "libDevKit.dylib" on macOS
    const CallingConvention Cdecl = CallingConvention.Cdecl;

    [DllImport(Dll, CallingConvention = Cdecl)]
    public static extern void cqdevkitCreate();

    [DllImport(Dll, CallingConvention = Cdecl)]
    public static extern void cqdevkitDestroy();

    [DllImport(Dll, CallingConvention = Cdecl)]
    public static extern int cqdevkitOpenUsbDevice(byte[] key, int size);

    [DllImport(Dll, CallingConvention = Cdecl)]
    public static extern void cqdevkitCloseDevice(int hDevice);

    [DllImport(Dll, CallingConvention = Cdecl)]
    public static extern int cqdevkitWriteDmxUniverse(int hDevice, int iPort, byte[] dmx);

    [DllImport(Dll, CallingConvention = Cdecl)]
    public static extern int cqdevkitReadFirmwareVersion(int hDevice,
        out byte major, out byte annual, out byte minorH, out byte minorL);
}

// DevKit.cqdevkitCreate();
// int hDevice = DevKit.cqdevkitOpenUsbDevice(null, 0);
// byte[] dmx = new byte[512]; dmx[0] = 255;
// if (hDevice >= 0) DevKit.cqdevkitWriteDmxUniverse(hDevice, 0, dmx);
```

Python (ctypes)

```
import ctypes, os, sys, time

if sys.platform == "win32":
    os.add_dll_directory(r"C:\path\to\DevKit\lib")   # folder of DevKit.dll and its DLLs
    dk = ctypes.CDLL("DevKit.dll")
else:
    dk = ctypes.CDLL("/path/to/DevKit/lib/libDevKit.dylib")

dk.cqdevkitCreate()
hDevice = dk.cqdevkitOpenUsbDevice(None, 0)
if hDevice >= 0:
    dmx = (ctypes.c_ubyte * 512)()
    dmx[0] = 255
    for i in range(400):                              # 10 s
        dk.cqdevkitWriteDmxUniverse(hDevice, 0, dmx)
        time.sleep(0.025)
    dk.cqdevkitCloseDevice(hDevice)
dk.cqdevkitDestroy()
```

Building and shipping your application

Windows

- Compile with `include` in your include paths and link with the import library of your toolchain: `cl my_app.c /I include lib\DevKit.lib` (Visual Studio), `gcc my_app.c -Iinclude -Llib -lDevKit` (MinGW).
- Your program must be 64-bit.
- Ship next to your executable every DLL of the `lib` folder: `DevKit.dll`, `libusb-1.0.dll`, the `Qt6*.dll` and the MinGW runtime (`libgcc_s_seh-1.dll`, `libstdc++-6.dll`, `libwinpthread-1.dll`).
- Install the driver on the computers of your users (see Installation), for example from your installer.

macOS

- Compile with `include` in your header search paths and link with `lib/libDevKit.dylib`: `clang my_app.c -Iinclude -Llib -lDevKit -Wl,-rpath,@executable_path/../Frameworks`.
- Copy into the `Contents/Frameworks` folder of your application bundle: `libDevKit.dylib`, `libusb-1.0.0.dylib` and every `Qt*.framework` of the `lib` folder. The library finds libusb and Qt next to itself, your executable finds the library through its rpath.
- Sign your application with your own Developer ID (hardened runtime), inside-out, then notarize it. No entitlement is needed to access the interfaces (outside of the App Sandbox).

> If your application is itself a Qt 6 application (Qt 6.11 or later; on Windows, built with MinGW 64-bit), use its own Qt libraries instead of the ones of the kit: do not ship two copies of Qt.

Migrating from version 3

Version 3	Version 4
`cqdevkitOpenUsbDevice(iDevice, ucConfig, ucKey, size)` then `cqdevkitInitDevices()`	`hDevice = cqdevkitOpenUsbDevice(ucKey, size)`: opens and initializes the next interface, returns its handle. Then `cqdevkitSetPortsDirection()` if needed.
Device index (`iDevice`)	Handle returned by `cqdevkitOpenUsbDevice()`.
`cqdevkitWriteInOutConfig(iDevice, Dmx_*)`	`cqdevkitSetPortsDirection()` / `cqdevkitSetPortDirection()` with `DEVKIT_PORT_OFF / IN / OUT`, and `cqdevkitGetPortCapabilities()`.
`cqdevkitGetNumberOfUniverses()`	`cqdevkitGetNumberOfPorts()`.
`bool` return values	`int`: 1 on success, 0 on failure.
References (`unsigned char&`, `unsigned short&`)	Pointers: `cqdevkitReadFirmwareVersion(h, &major, ...)`, `cqdevkitReadDmxConfig(h, &brk, ...)`.
DMX timings as codes (0 to 255)	Physical values: break and MaB in microseconds, period in milliseconds, bit rate in kbps.
`cqdevkitReadDmxUniverse()` always true	Returns 1 only when a new frame was received on the input port.
Key on every interface	Key on the interfaces of the previous generation only.

Licenses

libusb is distributed under the GNU LGPL 2.1, and the Qt libraries under the GNU LGPL 3. They are separate dynamic libraries: you can replace them with compatible versions. When you ship them with your application, include their license and give your users access to their source code (https://github.com/libusb/libusb, https://download.qt.io/archive/qt/).

CONTACTS

TECHNICAL SUPPORT:
http://www.chromateq.com
support (at) chromateq.com
You can visit our web site and use the online form for technical support.
Chromateq will reply shortly.

CRASH REPORT:
http://www.chromateq.com
You can visit our web site and use the online form for a crash report.
Chromateq will reply shortly and could supply an update file quickly.

GENERAL INFORMATION AND PURCHASE:
http://www.chromateq.com
info (at) chromateq.com
Tel: +33 (0)467 525 531
Fax: +33 (0)467 525 531

CHROMATEQ, Lighting and Visual Control Solutions.
The development kit is copyright (C) 2010-2026 CHROMATEQ SARL.
All rights reserved. All trademarks registered.