# maxon Motion Library - C++ Example

**ESCON2 in Profile Velocity Mode - Documentation**

Version 1.0.0 - 2026-09-21

This document collects the README and documentation files of the example archive. The same texts are in the archive as Markdown files next to the code.

# Contents

**Motion Helper**
- Quickstart (run the PVM example)
- Contents of the archive

**Getting Started**
- Prerequisites
- Using the Motion Helper in your own project

**Examples**
- Presets and output location
- Using Visual Studio 2022 or 2026
- Using Visual Studio Code
- Using the command line

**Example - Profile Velocity Mode**
- Prerequisites
- What the example does
- The device state machine in one minute
- When something goes wrong
- Adapting the example

maxon

# Motion Helper

**A C++ example for the maxon Motion Library: connect to an ESCON2 and run a motor in Profile Velocity Mode.**

The Motion Helper is two small C++17 classes, `MotionDeviceBase` and `MotionDevicePVM`, that wrap the C functions of the maxon *Motion Library* (`VCS2_*`) into a device object with a clear sequence: connect, select the operation mode, set the parameters, enable the power stage, command a velocity. The example program in `examples/profile_velocity_mode/` shows this sequence end to end and is meant to be copied as the starting point for your own program.

**Scope of this version:** ESCON2 over USB, Profile Velocity Mode (PVM), Windows, Visual Studio 2022 or 2026. Other operation modes and interfaces are not included; `MotionObjects.h` explains how to add objects for them.

- **Getting Started** - installing the Motion Library and Visual Studio, and using the Motion Helper in your own CMake project.
- **Examples** - building the examples with Visual Studio, Visual Studio Code or the command line.
- **Example: Profile Velocity Mode** - what the example does, the device state machine in one minute, and what to do when something goes wrong.

## Quickstart (run the PVM example)

> **Warning**
>
> Running the example turns the motor shaft. Read the safety notes in the example's README first.

1. Unpack the ZIP archive from the maxon Support Portal, e.g. to `C:\dev\motion_helper\`.
2. Open a **Developer PowerShell for VS 2022** (or 2026) from the Start menu. It puts `cmake`, `ninja` and the compiler `cl.exe` on the PATH; a plain PowerShell does not work with the presets.
3. Configure, build and run:

```
cd C:\dev\motion_helper
cmake --preset x64-release
cmake --build build\x64-release
build\x64-release\examples\profile_velocity_mode\example_pvm.exe
```

Requires an ESCON2 connected over USB, the Motion Library installed (see Getting Started), and motor and sensor parameters already configured with Motion Studio. Motion Studio must be closed while the example runs, because it holds the USB port. The build copies `MotionCmd64.dll` next to the `.exe`; the

executable does not run without it.

If the configure step ever fails with `The CMAKE_C_COMPILER: cl.exe is not a full path`, you are not in a Developer PowerShell. Delete the `build\` folder before retrying: the failed run leaves a cache behind that later runs would reuse.

## Contents of the archive

| Path | What it is |
|---|---|
| `source/` | The Motion Helper: `MotionDeviceBase`, `MotionDevicePVM`, the object dictionary, the error type |
| `examples/profile_velocity_mode/` | The PVM example: `main.cpp`, its `CMakeLists.txt`, its README |
| `motion_helper.cmake` | CMake module that finds the Motion Library and defines the targets `maxon_MotionLibrary` and `maxon_MotionHelper` |
| `CMakeLists.txt`, `CMakePresets.json` | Top-level project that builds all examples; the four presets |
| `documentation/` | Getting Started and the build instructions |

The same documentation is included as PDF in the archive.

© 2026 maxon. All rights reserved.

# Getting Started

## Prerequisites

Two things must be installed before the Motion Helper can be built:

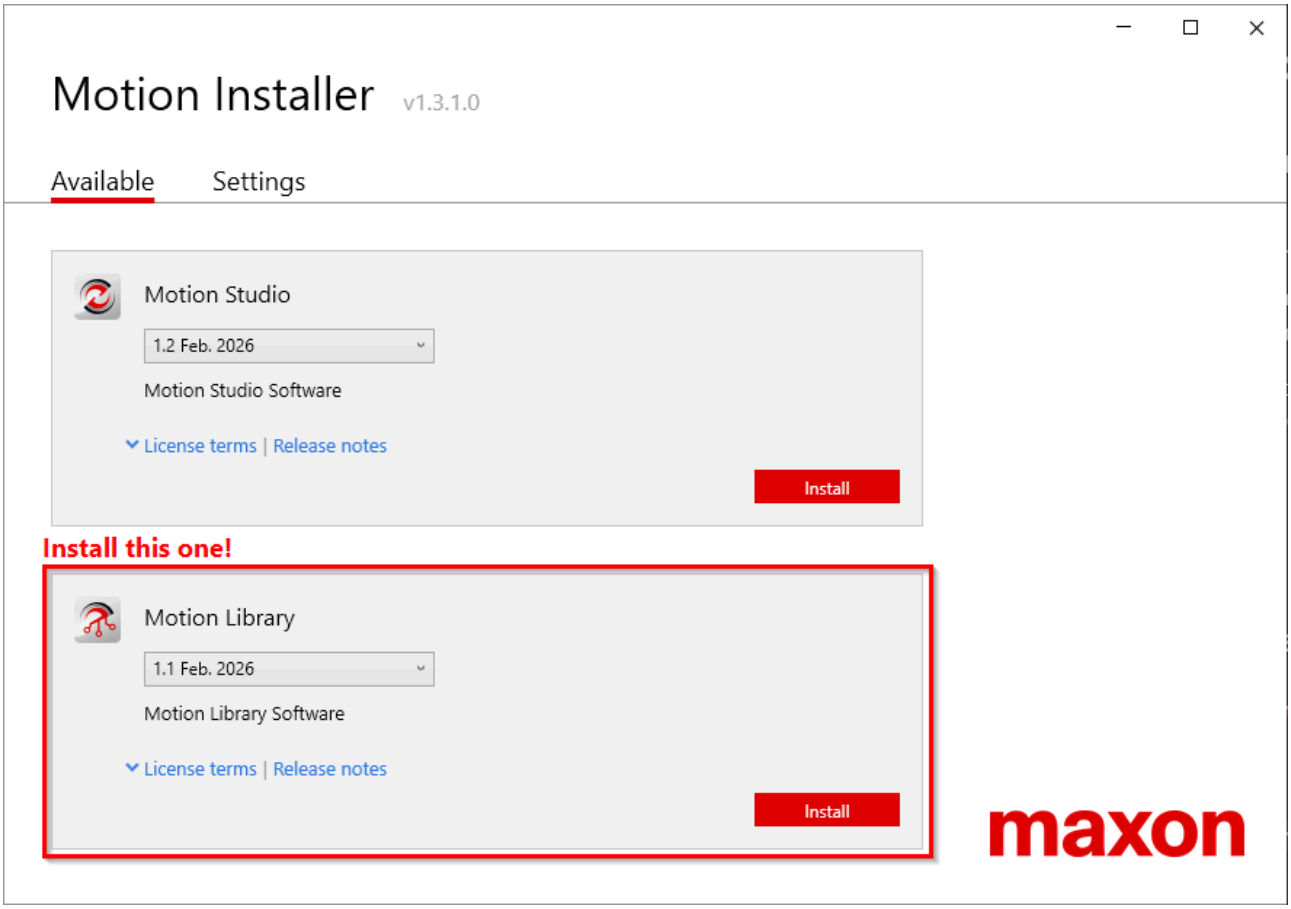
- the Motion Library - the communication library from maxon
- Visual Studio 2022 or 2026 - the C++ build tools

### Installing the Motion Library

The Motion Library (DLL, import library and header) is only distributed with the *Motion Studio* installer.

1. Download the Motion Studio installer from the maxon product page, unzip it and start it: https://www.maxongroup.com/en/drives-and-systems/controls/current-and-speed-controllers#software-included-242572

2. Select the package *Motion Library* for installation.




3. Click "Install" and wait until it has finished.

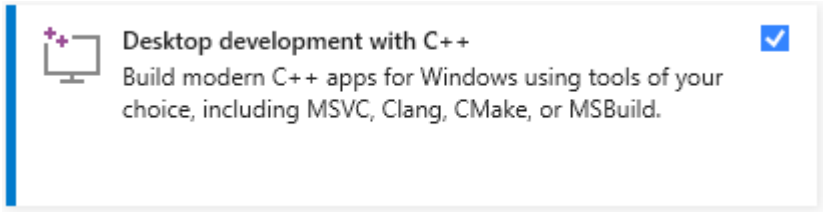
By default the library is installed to `C:\Program Files (x86)\maxon\Motor and Motion Control\Software\Motion Library`, and the CMake module finds it there without further configuration.

> **Note**
>
> If you install to a different location, note the path of the `Motion Library` folder (the one that contains `Windows Motion Library`, not `Windows Motion Library` itself). You will pass it to CMake as `MOTION_LIBRARY_PATH`, see below.

### Installing Visual Studio

Visual Studio 2022 or 2026 with the workload **Desktop development with C++**. Both are supported; any edition works, including Community.

1. Download Visual Studio: https://visualstudio.microsoft.com/downloads/
2. In the installer, select the workload "**Desktop development with C++**". It brings the compiler, CMake and Ninja.




3. Wait until it has finished.

## Using the Motion Helper in your own project

The Motion Helper is designed for CMake projects. The quickest way to your own program is to copy the example folder `examples/profile_velocity_mode/` and edit `main.cpp`. The steps below describe the same thing for an existing project.

1. Unpack the archive into a folder of your project, e.g. `third_party/motion_helper/`:

```
C:\dev\my_motion\
    ├── CMakeLists.txt
    ├── main.cpp
    └── third_party\
        └── motion_helper\
            ├── motion_helper.cmake
            ├── source\
            └── ...
```

2. In your `CMakeLists.txt`, include `motion_helper.cmake`, link the target `maxon_MotionHelper` and let the module copy the Motion Library DLL next to your executable. A complete minimal file:

```
cmake_minimum_required(VERSION 3.21)

project(my_motion LANGUAGES CXX)

set(CMAKE_CXX_STANDARD 17)
set(CMAKE_CXX_STANDARD_REQUIRED ON)

add_executable(my_motion main.cpp)

# Finds the Motion Library and defines the targets maxon_MotionLibrary and
maxon_MotionHelper.
include("${CMAKE_CURRENT_LIST_DIR}/third_party/motion_helper/motion_helper.cmake")

# Linking the helper brings the Motion Library (header and import library) along.
target_link_libraries(my_motion PRIVATE maxon_MotionHelper)

# The Motion Library DLL is not on the PATH; this copies it next to the executable.
maxon_copy_motion_library_dll(my_motion)
```

The `include()` may come before or after `add_executable()`; the two lines that name your target come after it.

3. In `main.cpp`, include `MotionDevicePVM.h` and follow the sequence of the example.

### Motion Library in a non-default location

If the Motion Library is not installed in its default location, tell CMake where it is - either with a CMake variable before the `include()`:

```
set(MOTION_LIBRARY_PATH "D:/tools/maxon/Software/Motion Library")
include(".../motion_helper.cmake")
```

or with an environment variable `MOTION_LIBRARY_PATH`, or on the command line with `-DMOTION_LIBRARY_PATH=...`. The path points at the `Motion Library` folder, **not** at `Windows Motion Library`. A path that is set but wrong stops the configure step with an error.

### Building from the command line

The CMake presets pin the compiler to `cl.exe`, so they need a **Developer PowerShell for VS** (Start menu), not a plain shell. The `x86` presets need the *x86 Native Tools Command Prompt*. See Examples.

# Examples

The examples are in the `examples` folder:

- Profile Velocity Mode - connect to an ESCON2, enable it and control the velocity from the keyboard.

Each example folder is a complete CMake project of its own, and the top-level `CMakeLists.txt` builds all of them at once. The sections below build the top-level project; the result is the same.

> **Important**
>
> Install the prerequisites first, and read the safety notes in the example's README before running anything - the examples turn the motor shaft.

## Presets and output location

`CMakePresets.json` defines four configure presets: `x64-debug`, `x64-release`, `x86-debug`, `x86-release`. The preset decides the build folder and which Motion Library (64- or 32-bit) is used. The built example is

```
build\<preset>\examples\profile_velocity_mode\example_pvm.exe
```

with the Motion Library DLL copied next to it.

## Using Visual Studio 2022 or 2026

Visual Studio opens CMake projects directly, without a solution file.

1. Start Visual Studio and choose **Open a local folder** (or *File → Open → Folder...*).
2. Select the **root folder** of the archive - the one that contains `CMakeLists.txt` and `CMakePresets.json`. Visual Studio reads the presets and configures the first one; the output window shows the CMake messages.
3. Select the configuration in the presets dropdown of the toolbar, e.g. *x64 Release*.
4. *Build → Build All* (`Ctrl+Shift+B`). The executable is in `build\<preset>\...` as described above; *Debug → Start Without Debugging* runs it in a console window.

If Visual Studio reports "Rebuild failed" without any error text after you switched presets or deleted a build folder by hand, the preset has no configured cache yet: run *Project → Configure Cache* once, then build.

## Using Visual Studio Code

1. Install the extension CMake Tools.
2. Start VS Code **from a Developer PowerShell for VS** ( `code .` in the archive's root folder). The presets pin the compiler to `cl.exe`, which is only on the PATH in that shell.
3. Press `F1` and run - `CMake: Select Configure Preset` to choose the preset, - `CMake: Build` to build, - `CMake: Run Without Debugging` to run.
4. To debug, create `.vscode/launch.json` with this configuration; it starts the example that CMake Tools has selected as launch target:

```
{
  "version": "0.2.0",
  "configurations": [
    {
      "name": "Example (external terminal)",
      "type": "cppvsdbg",
      "request": "launch",
      "program": "${command:cmake.launchTargetPath}",
      "cwd": "${command:cmake.launchTargetDirectory}",
      "console": "externalTerminal"
    }
  ]
}
```

## Using the command line

The presets need the compiler on the PATH. Open a **Developer PowerShell for VS 2022** (or 2026) from the Start menu for the `x64` presets, or an **x86 Native Tools Command Prompt for VS** for the `x86` presets, then:

```
cd C:\dev\motion_helper
cmake --preset x64-release
cmake --build build\x64-release
build\x64-release\examples\profile_velocity_mode\example_pvm.exe
```

If the configure step fails with `The CMAKE_C_COMPILER: cl.exe is not a full path`, the shell is not a Developer shell. Delete the `build\` folder before retrying; the failed run leaves a cache that later runs would reuse. If it fails with "The compiler on PATH targets 64-bit, but this preset expects 32-bit", the preset and the shell do not match.

# Example - Profile Velocity Mode

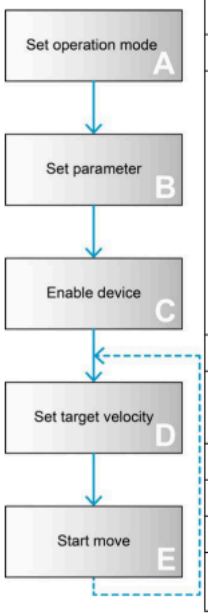
> **Warning**
>
> Running this example turns the motor shaft. Before the first run: secure the motor mechanically, remove or secure anything mounted on the shaft, and keep the power supply within reach. The target velocity is 0 at start; press `+` once (100 rpm) and watch the shaft before going further. `q` switches the power stage off - the shaft then coasts to a stop, it is not braked. If the program is interrupted in another way (closing the window, `Ctrl+C`), the drive keeps the last state it was commanded; switch the supply off if in doubt.

## Prerequisites

- An ESCON2 connected over USB, with logic and motor supply switched on.
- Motor and sensor configured and the controller tuned with **Motion Studio**, and Motion Studio **closed** afterwards - it holds the USB port.
- The Motion Library installed and the example built, see Getting Started and Examples.

## What the example does

`main.cpp` follows the *Profile Velocity Mode* sequence of the *ESCON2 Application Notes* (table *Device programming - Profile Velocity Mode*, downloads on the product page):




| # | Object name | Object | Description [unit] |
|---|---|---|---|
| A | Modes of operation | 0x6060-00 | 0x03 = (Profile Velocity Mode) |
| | | | |
| B | Max profile velocity | 0x607F-00 | Motor- and application-specific [rpm]; depending on motor, mechanics, application |
| | Profile acceleration | 0x6083-00 | Application-specific [rpm/s]; default: 10'000 depending on electrical and mechanical limits |
| | Profile deceleration | 0x6084-00 | Application-specific [rpm/s]; default: 10'000 depending on electrical and mechanical limits |
| | Quick stop deceleration | 0x6085-00 | Application-specific [rpm/s]; default: 10'000 |
| | Motion profile type | 0x6086-00 | Application-specific [0] |
| | | | |
| C | Controlword (Shutdown) | 0x6040-00 | 0x0006 |
| | Controlword (Switch on & Enable) | 0x6040-00 | 0x000F |
| | | | |
| D | Target velocity | 0x60FF-00 | Desired velocity [rpm] |
| | | | |
| E | Controlword | 0x6040-00 | 0x000F |

Table 2-6 Device programming | Profile Velocity Mode (Start)

| Step | Call in `main.cpp` | What happens on the device |
|---|---|---|
| Connect | `escon2.Connect("ESCON2", "MAXON SERIAL V2", "USB", "USB0")` | Opens the USB connection |
| Recover | `TryRecoverFromFault()` | If the device is in 'Fault', requests a fault reset |
| A | `SetModeOfOperation(OperationMode::PROFILE_VELOCITY_MODE)` | Selects PVM and verifies it |
| B | `PVM_SetParameters(4000, 5000, 5000, 10000)` | Velocity limit and the three ramps |
| C | `EnablePower()` | Walks the state machine to 'Operation Enabled'; the motor is now energized |
| D, E | `PVM_ApplyVelocity(velocity)` | Writes the target velocity and starts the move |
| Loop | keyboard | `+` / `-` change the velocity, `s` toggles a quick stop, `q` quits |
| Exit | `DisablePower()`, `Disconnect()` | Power stage off, connection closed |

The console shows the averaged actual velocity, the target velocity and whether a quick stop is active:

```
[LOG] ESCON2 connected!
[LOG] Device state: Operation Enabled
[LOG] ------------------------ Terminal Interface ------------------------

(+) to increase by 100rpm | (-) to decrease by 100rpm
(q) to quit               | (s) toggle quick stop

averaged:    500rpm / target:    500rpm
```

### Parameters used by the example

| Parameter | Object | Value in the example | Unit | Note |
|---|---|---|---|---|
| Max profile velocity | 0x607F | 4000 (`MAX_VELOCITY`) | rpm | Must not exceed *Max motor speed* configured in Motion Studio, otherwise `PVM_SetParameters` throws |
| Profile acceleration | 0x6083 | 5000 | rpm/s | Ramp when speeding up |
| Profile deceleration | 0x6084 | 5000 | rpm/s | Ramp when slowing down |
| Quick stop deceleration | 0x6085 | 10000 | rpm/s | Ramp used by the quick stop |
| Target velocity | 0x60FF | 0, changed in steps of ±100 | rpm | The sign is the direction of rotation |

Lower `MAX_VELOCITY` in `main.cpp` for your first runs.

## The device state machine in one minute

The ESCON2 follows the CiA 402 device state machine. Only in **Operation Enabled** does it accept a target velocity; everything else in the example exists to get there and back. The states the example passes through, and the controlword commands between them:

```
Switch On Disabled ──«Shutdown»──► Ready To Switch On ──«Enable operation»──► Operation
Enabled
        ▲                                                                          │        ▲
        │                                                              «Quick stop» │        │
«Enable operation»
        │                                                                          ▼        │
        │                                                                      Quick Stop
Active
        │
        ├── «Disable voltage» ── from every state above: power stage off, shaft coasts
        │
        └── «Fault reset» ──────  Fault  (entered from any state when the device detects an
error)
```

| Method | Command(s) it sends | Resulting state |
|---|---|---|
| `EnablePower()` | «Disable voltage», «Shutdown», «Enable operation» | Operation Enabled (power stage on) |
| `PVM_ApplyVelocity(v)` | «Enable operation» with a new target | stays in Operation Enabled |
| `PVM_ActivateQuickStop()` | «Quick stop» | Quick Stop Active - decelerates on the quick stop ramp, power stage stays on |
| `PVM_ReleaseQuickStop()` | «Enable operation» | Operation Enabled |
| `DisablePower()` | «Disable voltage» | Switch On Disabled - power stage off, from any state, shaft coasts |
| `ClearFault()` | «Fault reset» | Switch On Disabled, if the cause is gone |
| `GetDeviceState()` | reads the statusword | the current state; `DeviceStateName()` gives its name |

Every method that changes the state waits until the device reports the new state and throws if it does not - the message names the state that was expected and the one the device is in. The device enters **Fault** on its own when it detects an error (supply, overcurrent, sensor, ...); the example checks for it in every loop pass and stops.

## When something goes wrong

| Message | Cause | What to do |
| --- | --- | --- |
| `Connect: connection to controller failed (error code: 0x20000004: Opening port error)` | No device on `USB0`, or a different port | Replace the `Connect()` call in `main.cpp` with `escon2.ConnectUsingGUI()` once; the dialog lists the values valid on this PC. Then put them into `Connect()`. |
| `Connect: device handle is 0 - device already in use?` | Another program holds the port | Close Motion Studio or the other instance of the example |
| `ESCON2 entered state 'Fault'` | The device detected an error | Read the error in Motion Studio, fix its cause, restart the example - it clears the fault on startup |
| `Timeout while switching to 'Ready To Switch On' - device is in 'Switch On Disabled'` | Usually no motor supply, or the enable input is not set | Check the supply and the digital inputs configured in Motion Studio |
| `PVM_SetParameters ... (error code: 0x06090030: ...)` | A value is out of range | Lower `MAX_VELOCITY` below *Max motor speed* |
| `MotionCmd64.dll was not found` at start | The DLL is not next to the `.exe` | Run the executable from the build folder, or copy the DLL next to it |

Error messages from the Motion Library carry the abort code and its description. The codes are listed in *Motion Library Specification.pdf* in the `Manuals` folder of the Motion Library installation.

## Adapting the example

- **Connection**: the four strings passed to `Connect()` at the top of `main()`. `ConnectUsingGUI()` shows the valid values.
- **Velocity limit and ramps**: `MAX_VELOCITY` and the four values passed to `PVM_SetParameters()`.
- **Other objects**: `ReadObject()` / `WriteObject()` with the constants in `source/MotionObjects.h`; declare further objects there.
- **Another operation mode**: derive a class from `MotionDeviceBase` like `MotionDevicePVM`, add the mode to `OperationMode` and its objects to `MotionObjects.h`, and register the new `.cpp` and `.h` in `motion_helper.cmake`.