ELVEFLOW an ELVESYS brand

Elveflow Quick Start

SDK SOFTWARE DEVELOPMENT KIT

DOCUMENT REF: UGSDK-Cpp-230517

C++ GUIDE

Support: customer@elveflow.com

Symbols used in this document

Important information. Disregarding this information could increase the risk of damage to the equipment, or the risk of personal injuries.

Helpful information. This information will facilitate the use of the instrument and/or contribute to its optimal performance.

Additional information available on the internet or from your Elveflow representative.

READ THIS MANUAL CAREFULLY BEFORE USING THE SOFTWARE

This manual must be read by every person who is or will be responsible for using the Elveflow software development kit (SDK).

Due to the continual development of the products, the content of this manual may not correspond to the new software. Therefore, we retain the right to make adjustments without prior notification.

Important SDK safety notices:

1. The SDK gives the user complete control over Elveflow products. Beware of pressure limits for containers, chips and other parts of your setup. They might be damaged if the pressure applied is too high.

2. Use a computer with enough power to avoid software freezing.

If these conditions are not RESPECTED, the user is exposed to dangerous situations and the instrument can undergo permanent damage. Elvesys and its partners cannot be held responsible for any damage related to the misuse of the instruments.

Table of contents

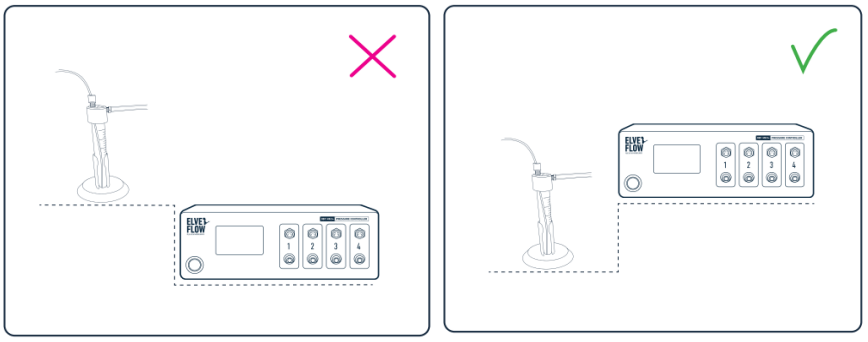
Getting started

Elveflow proposes a standard development kit for LabVIEW, C++, Python and MATLAB

The following sections will guide you through the steps to add a new instrument or sensor, explore its basic and advanced features and use it with other instruments to automate your experiment in C++.

Before starting

To prevent backflow in the pressure regulator, always place liquid reservoirs under the instrument



Important remarks

For all programming languages:

- **If MUX Distribution/Distributor/Recirculation/Injection or BFS are used,** FTDI drivers are required (http://www.ftdichip.com/Drivers/D2XX.htm). You can find these drivers in the same folder the ESI is installed. Default location would be C:\Program Files (x86)\Elvesys\driver (look for driver_MUX_distAndBFS.exe).
- **Do not simultaneously use the ESI software and the SDK**, some conflict would occur.
- **For X64 libraries running on an AMD computer,** the impossibility of communicating with instruments has been occasionally reported. Contact us if it is the case. The only fix found so far is to set the environment variable **MKL_DEBUG_CPU_TYPE** to **4.**

If you would like our expert advice on a particular section of your code, please be sure to give us some details about your issue.

Such as

- the SDK function you're using: e.g. OB1_init.vi
- the version of the SDK are you using: e.g. V3.08.00
- your Windows version: Windows 7 32-bit, or Windows 10 64-bit ?
- the environment you're using e.g. e.g. LabVIEW dev 2018.
- the elveflow device(s) you will use this code with (e.g. OB14000452042983 with a MUX-D-42989).
- etc... (add any relevant information you may have).

This will help minimize the turnaround time for understanding what your issue is and what the general fix would look like.

The SDK documentation

This document does not explain the list of functions available with the DLL, as well as the logic on how to connect and interact with Elveflow devices and sensors. That information is available in the “User Guide DLL SDK” and we strongly recommend referring to it during the reading of this document.

Specifics of C++ SDK programming

- **Not all compilers work with the DLL. Visual studio works.**
- An example has been written for every instrument, to show how to use every function of the SDK. These examples are included in the SDK folder (...\DLL64\Example_DLL64_Visual_Cpp\ElveflowDLL\OB1.cpp for example).
- Please remember to add the directory that contains the DLL library in Visual studio or another compiler.
- Please remember to include the “Elveflow64.h” located in the DLL library to the source code you are developing. It contains all the constants’ definition, aliases and functions.

Example using visual C++:

Some complete examples are compiled and embedded within the SDK. Each example has a source code that allows to use all the available SDK functions.

For x32 or x64 operating systems:

- ...\DLL32\Example_DLL32_Visual_Cpp\ElveflowDLL\Debug

- ...\DLL32\Example_DLL32_Visual_Cpp\ElveflowDLL\Release

For x64 operating systems only:

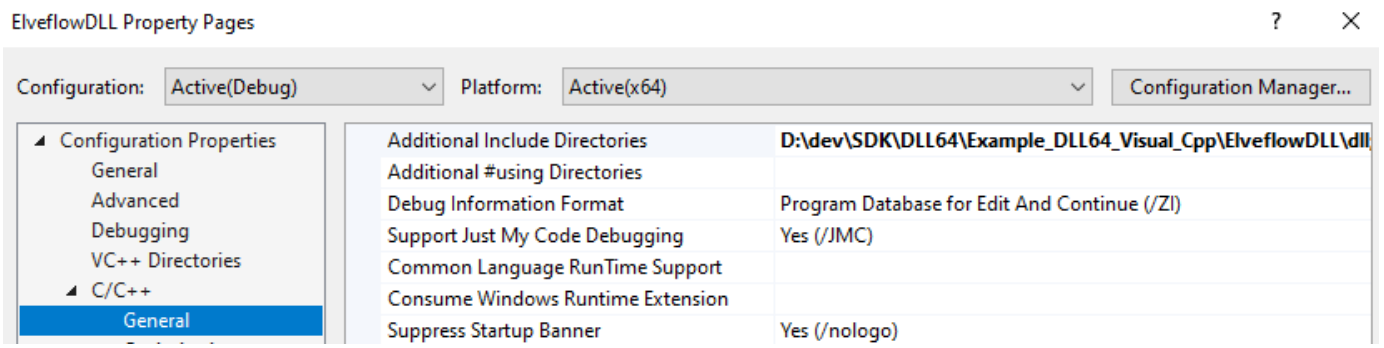
- ...\DLL64\Example_DLL64_Visual_Cpp\ElveflowDLL\x64\Debug

- ...\DLL64\Example_DLL64_Visual_Cpp\ElveflowDLL\x64\Release

Those examples will not work properly for your specific device (because the device name and configuration are hard coded within the code). However, because each example has a source code that allows to use all the available SDK functions, testing these executables will allow you to see if the DLL is properly working.

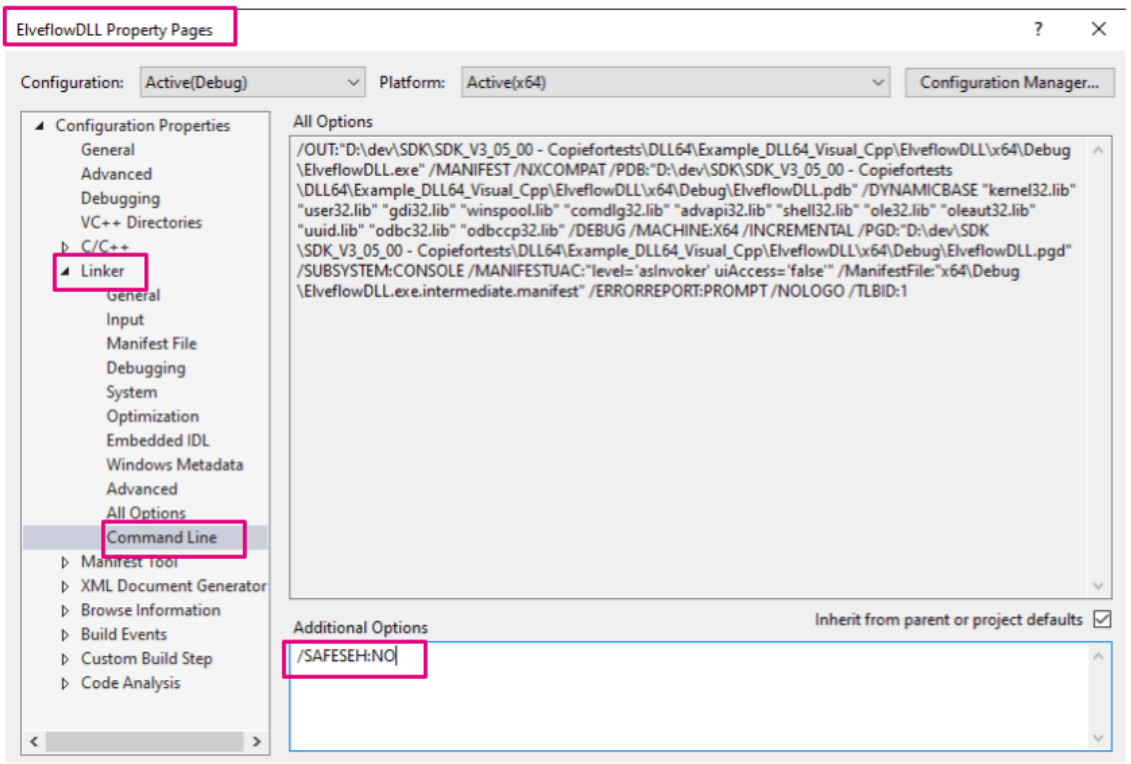
Important! Remember to add the directory that contains the dll in the additional directory (project ->

property: C++ -> general -> additional Include Directories) and to include all files in the dll folder.



Be careful: Ensure that you are in Project properties, and not in one of the CPP file properties.

For release executable add /SAFESEH:NO to the linker (Project properties -> linker -> Command lines)



C++ Quick start example

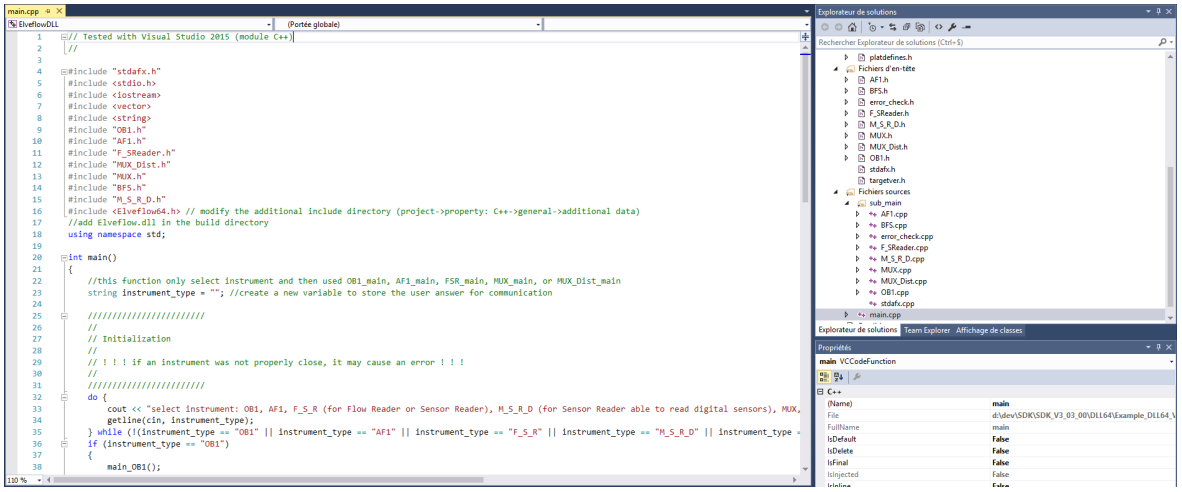
This section is here to guide you on how to use and modify the examples in Matlab. First of all, unzip the SDK file to have your uncompressed SDK folder.

All explanations described here are only for an OB1, but the principle is the same for other examples/instruments. We will consider for this quick start that we are using an OB1 MK4 with two regulators 0-200 mbar on channel 1 and 2, one regulator -1-1 bar on channel 3 and one regulator 0-8 bar on channel 4. On this OB1 we have a 1000 µL/min digital flow sensor that we want to use with H2O calibration and 16 bits resolution connected on channel 1 and a 1 bar pressure sensor connected on channel 3.

Connection and first commands

1) In your SDK folder, go to "DLL64/Example_DLL64_Visual_Cpp/ElveflowDLL" or "DLL32/Example_DLL32_Visual_Cpp/ElveflowDLL". Open the project "ElveflowDLL.vcxproj" in visual C++. Remember that MATLAB has to run in administrator mode. In this example we will consider that visual C++ has been configured as recommended previously in this User Guide (C++ section).

2) You should obtain a window similar to this one (depending on the SDK version you are running):



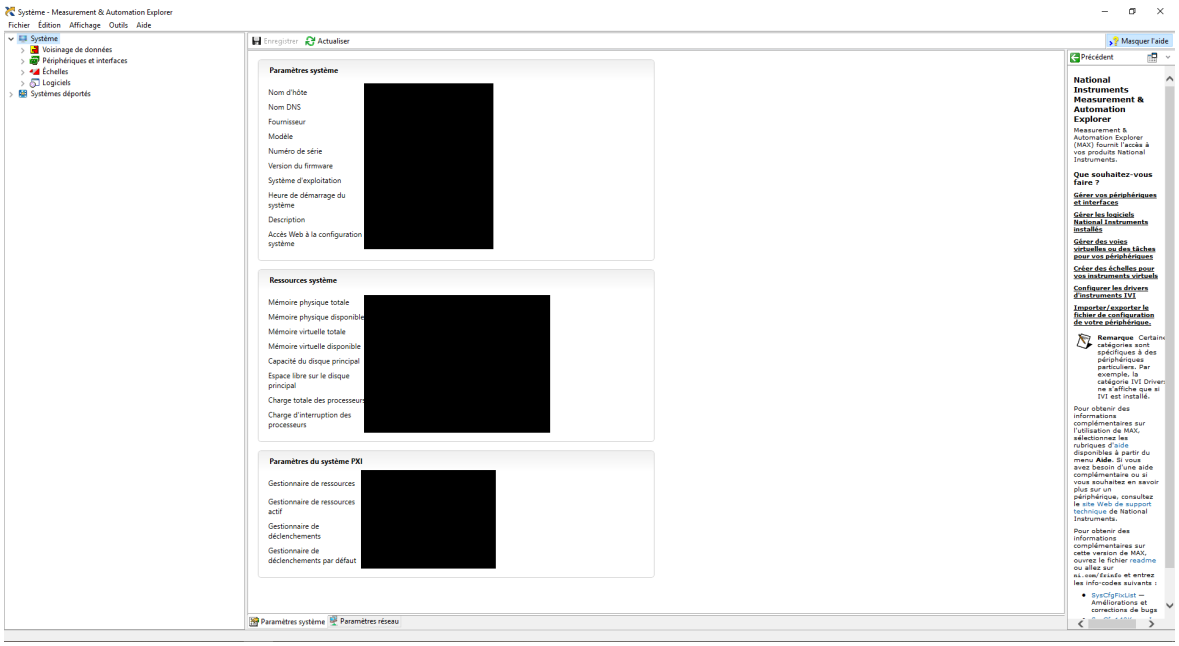
3) In the explorer on the right, search "OB1.cpp" and open it. You should obtain a code similar to this one (depending on the SDK version you are running):

4) To make this example work you need to modify the code to adapt it to your setup. Read the comments to have more details about elements to change (for other examples and this one too). First of all, you need to modify the Initialization function of your instrument here:

```
//////////////////////
//
// Initialization
//
//////////////////////
cout << "device name, regulators and sensors hardcoded in the OB1.cpp file" << endl;
//OB1_type *myOB1 = new OB1_type;
int MyOB1_ID = -1; // initialized myOB1_ID at negative value (after initialization it should become positive or =0)
//initialize the OB1 -> Use NIMAX to determine the device name
//avoid non alphanumeric characters in device name
error = OB1_Initialization("COM4", 0,0, 0, 0, &MyOB1_ID);
Check_Error(error);//error sent if not recognized
```

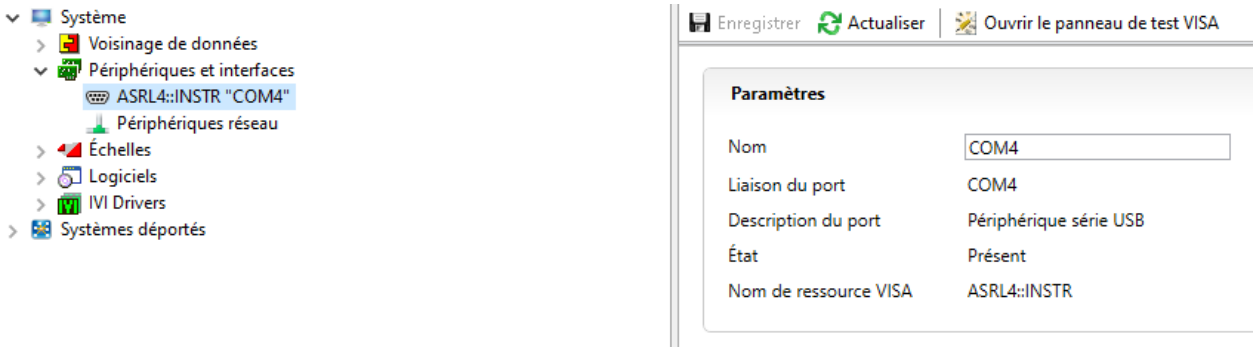
5) To modify this line (28 here) you need to open a software that has been automatically installed on your computer while installing ESI software. This is “NI MAX”. Find NI MAX on your computer by typing “NI MAX” on the Windows search for example and open it.

6) You should obtain a window similar to this one (except the black squares and language):



7) Expand the “Devices and Interfaces” tab to reveal connected instruments. Depending on the devices connected to your computer you will find multiple lines. For this example, the device is an OB1 MK4 so we have to find a COM port

Note : For an OB1 Mk3+ device we have to find a line with “NI USB-8451”. When you find it, click on it. You should obtain a window similar to this one:



8) The name of the instrument is written in the “Name” part. Here it is “COM4” but depending on the device connected (MSR, AF1 etc...) the name could be “Dev1” for example.

Please copy the name of the instrument and go back to visual C++.

9) Write the name of the instrument instead of the already written name between “xxxxx”. In the case of this instrument the modified window is as follows:

```
error = OB1_Initialization("COM4",
```

10) The second part of the Initialization function is used to define regulator type.With the new OB1 MK4 you can keep all the values to 0. For OB1 MK3+ devices, modify the function depending on your OB1. Considering the OB1 used for this example (described at the beginning of the section) the modified function should look like this:

```
error = OB1_Initialization("COM4", 0,0, 0, 0, &MyOB1_ID);
Check_Error(error);//error sent if not recognized
```

You can also use the defined variables, they are meant to be used but it is not used in this example to be able to display all information in one screenshot. Refer to the corresponding function in the User Guide (here it is “OB1_Initialization”) and the table to know which parameters to input in your case.

11) In the case of the OB1, we need to declare sensors that are connected to the OB1. If you do not have any sensor, please skip the following step to step 13. In the case of this example we need to add new lines to the code. The following lines are used to add sensors:

```
31  //error = OB1_Add_Sens(MyOB1_ID, 1, Z_sensor_type_Press_1_bar, Z_Sensor_digit_analog_Analog, Z_Sensor_FSD_Calib_H2O, Z_D_F_S_Resolution__16Bit);//
32  // ! ! ! If the sensor is not recognized a pop up will indicate it)
33  Check_Error(error);// error send if not recognized
```

12) Uncomment the “OB1_Add_Sens” and “CheckError” functions. Depending on the sensor connected the “OB1_Add_Sens” has to be modified. In our example we need to add two sensors. The new code should be as follows:

```
31      error = OB1_Add_Sens(MyOB1_ID, 1, 5, 1, 0, 7);// Add digital flow sensor with H2O Calibration
32      // ! ! ! If the sensor is not recognized a pop up will indicate it)
33      Check_Error(error);// error send if not recognized
34      error = OB1_Add_Sens(MyOB1_ID, 3, 8, 0, 0, 7);
35      Check_Error(error);
```

You can also use the defined variables, they are meant to be used but it is not used in this example to be able to display all information in one screenshot. Note that the last two parameters are unused in the case of the pressure sensor used in this example. Refer to the corresponding function in the User Guide (here it is “OB1_Add_Sens”) and the table to know which parameters to input in your case.

If you have more than two sensors, add that many lines (followed by Check_Error) up to 4 (one for each sensor physically connected). Channel number is the first parameter of the function after ID.

13) Then the example is ready to be launched. You can use default calibration or perform a new one. You can also load a calibration file but remember that calibration files generated through ESI cannot be used with SDK. Only SDK generated calibration files can be loaded using SDK.

14) Please follow instructions displayed on the screen to ask for pressure, sensor data etc...

Remote mode

15) While the example is running and asking for a new command, select 'start' which will run the function "**OB1_Start_Remote_Measurement**". The OB1 is now in remote mode, reading data in an asynchronous loop.
16) To check that the remote loop is operating correctly, now select 'read_channel' which will call the function "**OB1_Get_Remote_Data**" and will return the latest measured value from the OB1.
17) You can now immediately start a PID loop by calling 'add_PID' which will call the "add_PID" function. By calling read_channel again, you should see the regulator (control) is now following the sensor.
18) Update your new target for the PID loop by simply calling the 'set_target' at the channel regulating the PID loop, in this case channel 1, which will call the function "**OB1_Set_Remote_Target**".
19) You can confirm the PID loop is updated by reading the channel again with 'read_channel'
20) To exit the program, first stop the remote mode by selecting 'stop' which will call the function "**OB1_Stop_Remote_Measurement**".