Elveflow® SDK

USER GUIDE

July 2017

ELVEFLOW
PLUG & PLAY MICROFLUIDICS

READ THIS MANUAL CAREFULLY BEFORE USING THE SDK

This manual must be read by every person who is or will be responsible for using the SDK.

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

Important ESI 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.

Contents

Getting started

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.

Before starting

To prevent backflow in pressure regulator, always place liquid reservoirs under instrument (OB1, AF1…)

Specific Guides for Elveflow Instruments

User guides are available for every Elveflow instrument. Check the dedicated guide to correctly set up your experiment before using the Elveflow Smart Interface.

SDK

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

Important remark

For all programming languages:

Important! If MUX distributor or BFS are used, FTDI drivers are required (http://www.ftdichip.com/Drivers/D2XX.htm)

Also important! Elveflow interface Smart interface V3 has to be installed to ensure the installation of every resource required to communicate with the instrument. **For X64 libraries**, Labview 2015 x64 Run-time should be installed. It is included in the installation file (Extra Installer for X64 Libraries)

Very important! Do not use simultaneously ESI software and the SDK, some conflict would occur.

For C ++/MatLAB/Python

Important! Instruments are designated using their device name. The device name can be known and changed using Measurement and Automation Explorer. The NI MAX Software should be automatically installed with Elveflow Smart Interface,

Error handling
All functions return an error code. If this code is 0 no error occurs. Other values indicate that an error occurs. Some personalized errors were added.

Error code:	Signification :
-8000	No Digital Sensor found
-8001	No pressure sensor compatible with OB1 MK3
-8002	No Digital pressure sensor compatible with OB1 MK3+
-8003	No Digital Flow sensor compatible with OB1 MK3
-8004	No IPA config for this sensor
-8005	Sensor not compatible with AF1
-8006	No Instrument with selected ID

Other error can be found in Labview error user guide. (http://www.ni.com/pdf/manuals/321551a.pdf)

Feedback
An example function that could be used for feedback control is included in all libraries **as an illustration only**. It is provided as an example to help you to create your own regulation system.

LabVIEW

All VI are included in ElveflowLLB.llb file.

For every instrument an example to show how to use the SDK Labview is available.

__AF1_Ex__.vi for AF1, __F_S_R_Ex__.vi for Flow Reader or Sensor Reader, __MUX_Dist_Ex__.vi for Mux Distributor, __MUX_Ex__.vi for MUX, and __OB1_Ex__.vi for OB1.

C++

Not all compilers work with the DLL. Visual studio works.

An Example has been written for every instrument, to shows how to use every function of the SDK. Those examples are included in the SDK folder.

Example Using Visual C++:

Some examples are embedded within SDK:

For X32 or x64 operating system:

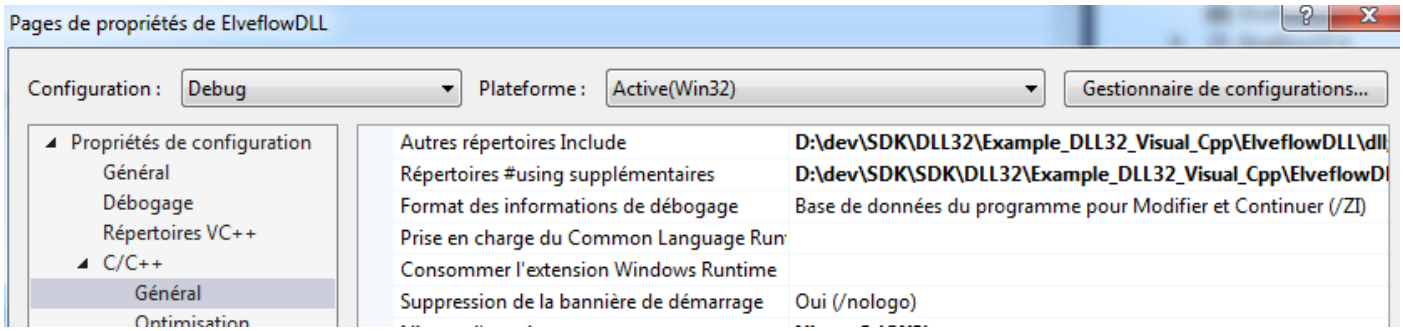
- ...\DLL32\Example_DLL32_Visual_Cpp\ElveflowDLL\Debug
- ...\DLL32\Example_DLL32_Visual_Cpp\ElveflowDLL\Release

For x64 operating system only:

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

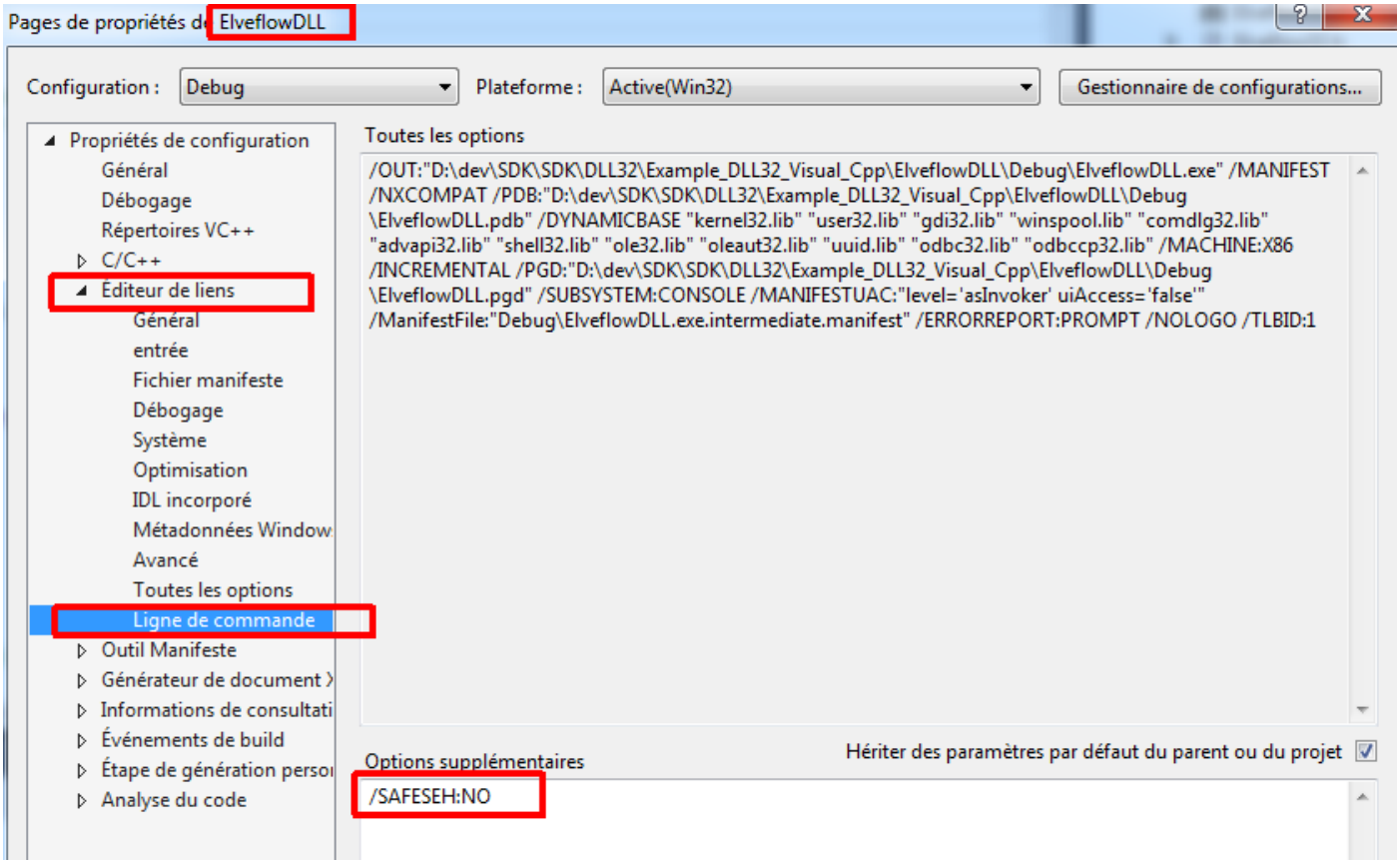
Those examples will not properly work for your specific device (because the device name and configuration are hard coded within the code) but it allows you to see if the DLL is properly working.

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)



MATLAB

Important! In order to load and use DLL, run MATLAB as administrator.

Also important! In order to load and use Elveflow DLL, the compiler should be either **Visual C++ Professional** or **Windows SDK 7.1**. To check which is the actual default compiler, type `mex -setup c++` in MATLAB command line.

Microsoft visual studio can be downloaded from the following link:

https://www.visualstudio.com/us/vs/older-downloads

To check which compilers are compatible with your version of MATLAB, check the following link:

https://mathworks.com/support/compilers.html

https://mathworks.com/support/sysreq/previous_releases.html

Once installed, set the new compiler as default using the command `mex -setup c++`.

MATLAB does not support pointers natively, therefore the function `libpointer` can be called to create them.

A description of the function is provided in the .m file. It uses a similar prototype as the C++ dll. To learn how to use them, one example for every instrument is available in MATLABXX/Example where XX is either 32 or 64 depending on your MATLAB version.

PYTHON

In order to load pointer array (as calibration) the library ctypes is used:

`C_double`*1000 for calibration (AF1 &OB1)

`c_double*4` for pressure_array_out (OB1)

`c_int32*16` for array_valve_in (MUX).

Call this variable with the function byref().

An Example has been written for every instrument, to shows how to use every function of the SDK. Those examples are included in the SDK folder (in python_XX/Example where XX is either 32 or 64 depending on your python version).

List of prototypes and description (for C++, MathLab and Python):

All instruments have initialization and destructor function and several other functions describe below.

All function will return an error code that could help to debug your software.

Constants (define Elveflow.h as uint16_t) :

Z_regulator_type:	
Z_regulator_type_none	0
Z_regulator _type__0_200_mbar	1
Z_regulator _type__0_2000_mbar	2
Z_regulator _type__0_8000_mbar	3
Z_regulator _type_m1000_1000_mbar	4

Z_sensor_type:	
Z_sensor_type_none	0
Z_sensor _type_Flow_1_5_muL_min	1
Z_sensor _type_Flow_7_muL_min	2
Z_sensor _type_Flow_50_muL_min	3
Z_sensor _type_Flow_80_muL_min	4
Z_sensor _type_Flow_1000_muL_min	5
Z_sensor _type_Flow_5000_muL_min	6
Z_sensor _type_Press_340_mbar	7
Z_sensor _type_Press_1_bar	8
Z_sensor _type_Press_2_bar	9
Z_sensor _type_Press_7_bar	10
Z_sensor _type_Press_16_bar	11
Z_sensor _type_Level	12

Z_Sensor_digit_analog:	
Z_Sensor_digit_analog_Analog	0
Z_Sensor_digit_analog_Digital	1

Z_Sensor_FSD_Calib:	
Z_Sensor_FSD_Calib_H2O	0
Z_Sensor_FSD_Calib_IPA	1

Calibration and PID Example (required for AF1 and OB1):

```
int32_t __cdecl Elveflow_Calibration_Default(double Calib_Array_out[], int32_t len);
Set default Calib in Calib cluster, len is the Calib_Array_out array length

int32_t __cdecl Elveflow_Calibration_Load(char Path[], double Calib_Array_out[], int32_t len);
Load the calibration file located at Path and returns the calibration parameters in the
Calib_Array_out. len is the Calib_Array_out array length. The function asks the user to choose the
path if Path is not valid, empty or not a path. The function indicate if the file was found.

int32_t __cdecl Elveflow_Calibration_Save(char Path[], double Calib_Array_in[], int32_t len);
Save the Calibration cluster in the file located at Path. len is the Calib_Array_in array length.
The function prompt the user to choose the path if Path is not valid, empty or not a path.

int32_t __cdecl Elveflow_EXAMPLE_PID(int32_t PID_ID_in, double actualValue, int32_t Reset, double P,
double I, int32_t *PID_ID_out, double *value);
This function is only provided for illustration purpose, to explain how to do your own feedback
loop. Elveflow does not guarantee neither efficient nor optimum regulation with this illustration of
PI regulator. With this function the PI parameters have to be tuned for every regulator and every
microfluidic circuit. In this function need to be initiate with a first call where PID_ID =-1. The
PID_out will provide the new created PID_ID. This ID should be use in further call. General remarks
of this PI regulator: The error "e" is calculate for every step as e=target value-actual value
There are 2 contributions to a PI regulator: proportional contribution which only depend on this
step and Prop=eP and integral part which is the "memory" of the regulator. This value is calculated
as Integ=integral(Iedt) and can be reset.
```

AF1 :

```
int32_t __cdecl AF1_Calib(int32_t AF1_ID_in, double Calib_array_out[], int32_t len);
Launch AF1 calibration and return the calibration array. Len correspond to the Calib_array_out
length.

int32_t __cdecl AF1_Destructor(int32_t AF1_ID_in);
Close Communication with AF1

int32_t __cdecl AF1_Get_Flow_rate(int32_t AF1_ID_in, double *Flow);
Set the external trigger of the instrument to high level if status is true and low level if status
is false.

int32_t __cdecl AF1_Get_Press(int32_t AF1_ID_in, int32_t Integration_time, double Calib_array_in[],
double *Pressure, int32_t len);
Get the pressure of the AF1 device, Calibration array is required (use Set_Default_Calib if
required). Len correspond to the Calib_array_in length.

int32_t __cdecl AF1_Get_Trig(int32_t AF1_ID_in, int32_t *trigger);
Get the trigger of the AF1 device (0=0V, 1=5V).

int32_t __cdecl AF1_Initialization(char Device_Name[], Z_regulator_type Pressure_Regulator,
Z_sensor_type Sensor, int32_t *AF1_ID_out);
IInitiate the AF1 device using device name (could be obtained in NI MAX), and regulator, and
sensor. It return the AF1 ID (number >=0) to be used with other function

int32_t __cdecl AF1_Set_Press(int32_t AF1_ID_in, double Pressure, double Calib_array_in[], int32_t
len);
Set the pressure of the AF1 device, Calibration array is required (use Set_Default_Calib if
required).Len correspond to the Calib_array_in length.

int32_t __cdecl AF1_Set_Trig(int32_t AF1_ID_in, int32_t trigger);
Set the Trigger of the AF1 device (0=0V, 1=5V).
```

BFS:

```
int32_t __cdecl BFS_Destructor(int32_t BFS_ID_in);
Close Communication with BFS device

int32_t __cdecl BFS_Get_Density(int32_t BFS_ID_in, double *Density);
Get fluid density (in g/L) for the BFS defined by the BFS_ID

int32_t __cdecl BFS_Get_Flow(int32_t BFS_ID_in, double *Flow);
Get Frow rate (in µl/min) of the BFS defined by the BFS_ID
!!! This function required an earlier density measurement!!! The density can either be measured only once at the beginning of the experiment (ensure that the fluid flow through the sensor prior to density measurement), or before every flow measurement if the density might change. If you get +inf or –inf, the density wasn't correctly measured.

int32_t __cdecl BFS_Get_Temperature(int32_t BFS_ID_in, double *Temperature);
Get the fluid temperature (in °C) of the BFS defined by the BFS_ID

int32_t __cdecl BFS_Initialization(char Visa_COM[], int32_t *BFS_ID_out);
Initiate the BFS device using device com port (ASRLXXX::INSTR where XXX is  the com port that could be found in windows device manager). It return the  BFS ID (number >=0) to be used with other function

int32_t __cdecl BFS_Set_Filter(int32_t BFS_ID_in, double Filter_value);
Elveflow Library BFS Device  Set the instruement Filter. 0.000001= maximum filter -> slow change but very low noise. 1= no filter-> fast change but noisy.  Default value is 0.1
```

Flow Reader or Sensor Reader:

```
int32_t __cdecl F_S_R_Destructor(int32_t F_S_Reader_ID_in);
Close Communication with F_S_R.

int32_t __cdecl F_S_R_Get_Sensor_data(int32_t F_S_Reader_ID_in, int32_t Channel_1_to_4, double *output);
Get the data from the selected channel.

int32_t __cdecl F_S_R_Initialization(char Device_Name[], Z_sensor_type Sens_Ch_1, Z_sensor_type Sens_Ch_2, Z_sensor_type Sens_Ch_3, Z_sensor_type Sens_Ch_4, int32_t *F_S_Reader_ID_out);
Initiate the F_S_R device using device name (could be obtained in NI MAX)  and sensors. It return the F_S_R ID (number >=0) to be used with other  function.  NB: Flow reader can only accept Flow sensor NB 2: Sensor connected to channel 1-2 and 3-4 should be the same type  otherwise they will not be taken into account and the user will be informed  by a prompt message.
```

MUX Distributor:

```
int32_t __cdecl MUX_Dist_Destructor(int32_t MUX_Dist_ID_in);
Close Communication with MUX distributor device

int32_t __cdecl MUX_Dist_Get_Valve(int32_t MUX_Dist_ID_in, int32_t *selected_Valve);
Get the active valve

int32_t __cdecl MUX_Dist_Initialization(char Visa_COM[], int32_t *MUX_Dist_ID_out);
Initiate the MUX Distributor device using device com port (ASRLXXX::INSTR  where XXX is the com port that could be found in windows device manager).  It return the MUX Distributor ID (number >=0) to be used with other  function

int32_t __cdecl MUX_Dist_Set_Valve(int32_t MUX_Dist_ID_in, int32_t selected_Valve);
Set the active valve
```

MUX & MUX Wire:

```
int32_t __cdecl MUX_Destructor(int32_t MUX_ID_in);
Close the communication of the MUX device
```

```
int32_t __cdecl MUX_Get_Trig(int32_t MUX_ID_in, int32_t *Trigger);
Get the trigger of the MUX device (0=0V, 1=5V).


int32_t __cdecl MUX_Initialization(char Device_Name[], int32_t *MUX_ID_out);
Initiate the MUX device using device name (could be obtained in NI MAX). It  return the F_S_R ID
(number >=0) to be used with other function

int32_t __cdecl MUX_Set_Trig(int32_t MUX_ID_in, int32_t Trigger);
Set the Trigger of the MUX device (0=0V, 1=5V).

int32_t __cdecl MUX_Set_all_valves(int32_t MUX_ID_in, int32_t array_valve_in[], int32_t len);
Valves are set by a array of 16 element. If the valve value is equal or below 0, valve is close, if
it's equal or above 1 the valve is open. The index in the array indicate the selected valve as shown
below :
 0  1  2  3
 4  5  6  7
 8  9  10 11
 12 13 14 15
 If the array does not contain exactly 16 element nothing happened


int32_t  __cdecl  MUX_Set_indiv_valve(int32_t  MUX_ID_in,  int32_t  Input,  int32_t  Ouput,  int32_t
OpenClose);
Set the state of one valve of the instrument. The desired valve is  addressed using Input and Output
parameter which corresponds to the  fluidics inputs and outputs of the instrument.

int32_t __cdecl MUX_Wire_Set_all_valves(int32_t MUX_ID_in, int32_t array_valve_in[], int32_t len);
Valves are set by a array of 16 element. If the valve value is equal or below 0, valve is close, if
it's equal or above 1 the valve is open. If the  array does not contain exactly 16 element nothing
happened
```

OB1:

```
int32_t  __cdecl  OB1_Add_Sens(int32_t  OB1_ID,  int32_t  Channel_1_to_4_,  Z_sensor_type  SensorType,
Z_Sensor_digit_analog Digital_or_Analog, Z_Sensor_FSD_Calib FSens_Digit_Calib);
Add sensor to OB1 device. Selecte the channel n° (1-4) the sensor type.   For Flow sensor, the type
of communication (Analog/Digital) and the  Calibration for digital version (H20 or IPA) should be
specify.  For digital version, the sensor type is automatically detected.   For other sensor, those
parameters are not taken into account.   If the sensor is not compatible with the OB1 version, or no
digital sensore  are detected a pop-up will inform the user.

int32_t __cdecl OB1_Calib(int32_t OB1_ID_in, double Calib_array_out[], int32_t len);
Launch OB1 calibration and return the calibration array. Len correspond to  the Calib_array_out
length.

int32_t __cdecl OB1_Destructor(int32_t OB1_ID);
Close communication with OB1

int32_t __cdecl OB1_Get_Press(int32_t OB1_ID, int32_t Channel_1_to_4, int32_t
Acquire_Data1True0False, double Calib_array_in[], double *Pressure, int32_t Calib_Array_len);
Get the pressure of an OB1 channel. Calibration array is required (use Set_Default_Calib if
required) and return a double. Len correspond to the Calib_array_in length.  If Acquire_data is
true, the OB1 acquires ALL regulator AND ALL analog  sensor value. They are stored in the computer
memory. Therefore, if several regulator values (OB1_Get_Press) and/or sensor values
(OB1_Get_Sens_Data) have to be acquired simultaneously, set the Acquire_Data to true only for the
First function. All the other can used the values stored in memory and are almost instantaneous.

int32_t __cdecl OB1_Get_Sens_Data(int32_t OB1_ID, int32_t Channel_1_to_4, int32_t
Acquire_Data1True0False, double *Sens_Data);
/* Read the sensor of the requested channel. ! This Function only convert data that are acquired in
OB1_Acquire_data Units: Flow sensor µl/min Pressure : mbar  If Acquire_data is true, the OB1
acquires ALL regulator AND ALL analog  sensor value. They are stored in the computer memory.
Therefore, if several regulator values (OB1_Get_Press) and/or sensor values (OB1_Get_Sens_Data) have
```

to be acquired simultaneously, set the Acquire_Data to true only for the First function. All the other can used the values stored in memory and are almost instantaneous. For Digital Sensor, that required another communication protocol, this parameter have no impact

NB: For Digital Flow Senor, If the connection is lots, OB1 will be reseted and the return value will be zero

```
int32_t __cdecl OB1_Get_Trig(int32_t OB1_ID, int32_t *Trigger);
```
Get the trigger of the OB1 (0 = 0V, 1 =3,3V)

```
int32_t __cdecl OB1_Initialization(char Device_Name[], Z_regulator_type Reg_Ch_1, Z_regulator_type Reg_Ch_2, Z_regulator_type Reg_Ch_3, Z_regulator_type Reg_Ch_4, int32_t *OB1_ID_out);
```
Initialize the OB1 device using device name and regulators type . It return the OB1 ID (number >=0) to be used with other function. If an error occurs the return value will be -1

```
int32_t __cdecl OB1_Set_Press(int32_t OB1_ID, int32_t Channel_1_to_4, double Pressure, double Calib_array_in[], int32_t len);
```
Set the pressure of the OB1 selected channel, Calibration array is required (use Set_Default_Calib if required). Len correspond to the Calib_array_in length.

```
int32_t __cdecl OB1_Set_Trig(int32_t OB1_ID, int32_t trigger);
```
Set the trigger of the OB1 (0 = 0V, 1 =3,3V)