Fluigent Software Developement Kit

User Manual

Version 24.0.0.0 09/2024

Contents

1 | Introduction to Fluigent SDK

Fluigent Software Development Kit (SDK) allows you to fully integrate Fluigent devices in your application; it has been declined in several languages, among the most popular ones in the instrumentation field (e.g. LabVIEW, C++, C#.NET, Python...).

The aim of this document is to introduce the SDK's exposed functions which can be used to interact with your Fluigent instruments.

This SDK regroups all Fluigent pressure and sensor instruments as well as an advanced regulation loop. You can still use independent SDK (MFCS, FRP, LineUP) for basic hardware set-ups or for specific software requirements.

Main advantages of using this SDK:

- all Fluigent instruments (pressure and sensor) are managed by one instance (instead of one instance per intrument type)
- if hardware is changed in many cases software code does not need to be adapted
- embedded regulation allow powerful and custom loop feedback between any pressure and sensor
- custom sensors (other than Fluigent ones) can also be pressure regulated
- features such as limits, units, calibration and detailed errors allow advanced functionalities

2 | Requirements

2.1 System requirements

The Fluigent SDK can run on the following systems:

Operating system	x86 (32 bits)	x64 (64 bits)	ARM (32 bits)	ARM64 (64 bits)
Windows 10, 11	✓	✓		
Linux[1]		✓	✓	✓
macOS		✓		

1. Requires kernel version 2.6.39 or newer and GLIBC $\geq$ 2.28 (Debian 10)

2.2 Supported instruments

By using Fluigent SDK, you have direct access to following Fluigent devices:

- MFCS™ Series: MFCS™, MFCS™-EZ, MFCS™-EX and PX pressure controllers
- LineUP Series: Link, Flow EZ™ pressure controller, flow-units XS, S, M, L and XL connected to Flow EZ™, Switch EZ module with ESS switches (M-Switch, L-Switch and Two-Switch) and P-Switch module
- Flowboard: XS, S, M, L and XL Flow Units
- Inline Pressure Sensor (Pressure Unit) S, M and XL
- Switchboard with ESS switches (M-Switch, L-Switch and Two-Switch)
- F-OEM Series: INT-OEM, Pressure Module, Switch Module, Electrovalve Module
- Non-Invasive Flow Sensor (NIFS)

3 | SDK general philosophy

The Fluigent SDK gives access to all supported Fluigent instruments, as listed in 2.2. The instruments are sorted into categories according to the functionalities they support, and those functionalities are accessed via specific functions.

All instruments are divided into two main topics:

- channels: units that are controllable independently
- controllers: instruments that provide an interface between one or more channels and the computer

The default initialization order is as listed in 2.2. If multiple instruments of same type are connected, they are sorted by their serial number in ascending order.

The following image shows an example of channels and controllers in their default initialization order:

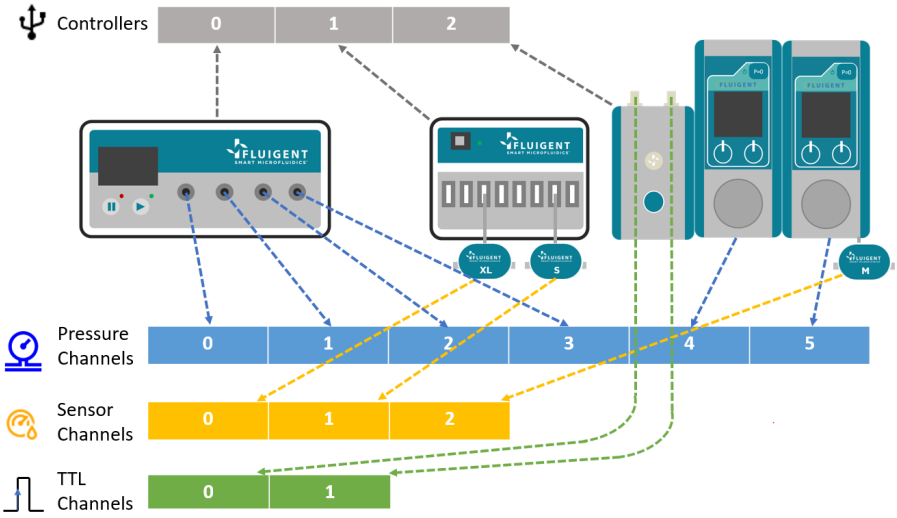

Figure 3.1: Default controllers and channel sorting

In this example there is a total of 6 pressure channels, 3 sensors (flow-units), 2 TTL channels (BNC ports) for 3 main controllers (MFCS-EZ, FRP and Link). This indexing is then used when calling related functions. For example: setting pressure on first FlowEZ will use index 4.

This is the default indexing if the instruments are initialized using fgt_init. By using fgt_initEx instead, instruments can be initialized in a different order if needed.

For example, if in Figure 3.1 the MFCS-EZ serial number is 567, the Flowboard 800 and the Link 11023, and you want LineUP instruments to appear first on the channel indexing, call fgt_initEx with [11023, 567, 800] in the array. Setting pressure on first Flow EZ will then use index 0 (instead of 4 by default).

3.1 Channels

The Fluigent SDK supports the following types of instrument channels:

- pressure controller (from MFCS™ Series or LineUP Series)
- sensor (Flow Unit, Inline Pressure Sensor, and Non-Invasive Flow Sensor)
- TTL 0-5V input/output ports (from Link, LineUP Series)
- valve ESS and LineUP valves (M-Switch, L-Switch, Two-Switch and P-Switch)

Each channel type has its own indexing starting from 0 and a dedicated set of functions that give access to the channel's functionalities. The functions take as argument the channel index within the list of channels of the same type. So, for example, fgt_get_pressure(0) returns the pressure on the first pressure channel, and fgt_get_valvePosition(0) returns the current position of the first valve channel.

3.1.1 Unique ID

Each Fluigent controller and channel initialized by the SDK is given a unique identification number, which is constructed for the instrument type (MFCS, LineUP etc.), the serial number and, in the case of channels, the type of channel and its position in the controller that contains it.

It can be used in place of the positional indices in all SDK functions. They are meant to prevent the wrong channels from being addressed if the SDK is reinitialized with a different set of instruments, which would cause the positional indices to change.

For example, if the first pressure channel in the SDK has the unique ID 287523328, the function calls fgt_get_pressure(0) and fgt_get_pressure(287523328) will both address this channel. However, the latter will continue to address the same instrument if the SDK is reinitialized with another instrument that comes first in the initialization order, and will return an error if the original instrument is no longer present.

The unique ID can be obtained using the channel information functions described in the following section.

3.1.2 Channel information

The SDK provides functions that return information on all initialized channels of a given type, including the controller serial number, device serial number (if applicable), firmware version (if available), position (index from controller), unique ID and instrument type.

There is a dedicated function for each channel type:

- fgt_get_pressureChannelsInfo,
- fgt_get_sensorChannelsInfo,
- fgt_get_TtlChannelsInfo
- fgt_get_valveChannelsInfo

The sensor and valve information functions also return an array which contains, respectively, the sensor types (fgt_SENSOR_TYPE) and the valve types (fgt_VALVE_TYPE).

3.1.3 Advanced features

More than setting\reading pressure and sensor some advanced features are also available in order to ease SDK usage and integration.

A pressure limit can be set on each pressure channel. When setting a limit, instrument will never apply an order over this value. Aim is to protect microfluidic device (chips, valves, cell stretching...). When closed loop regulation is running, limit is also taken into account. Minimal and maximal value can be set using fgt_set_pressureLimit function.

Default unit used by pressure channels is *mbar* and *µl/min* for sensor flowrate channels. Unit can be changed, then all related functions is using it. A large choice of units are accepted, moreover non SI ones are also accepted such as *ul per hour*, *nL/second*... If wrong unit is send or if it is invalid an error is returned.

When working with liquids that have different properties from water and isopropyl alcohol such as some fluorinated oils a polynomial function can be used to adjust the flow rate measurements. Scale factor is applied using following formula:

$$scaled\_value = a * sensor\_value + b * sensor\_value^2 + c * sensor\_value^3 \tag{3.1}$$

When applying a scale factor, sensor range is also changed and can fast reach big values. In order to limit sensor in the experimental range fgt_set_sensorCustomScaleEx can be used.

3.2 Controllers

Fluigent instrument controllers are the devices that connect directly to the computer:

- MFCS™ Series: MFCS™, MFCS™-EZ, MFCS™-EX and PX pressure controllers
- Flowboard
- LineUP Series: Link
- IPS (if used as a stand-alone sensor)
- Switchboard
- INT-OEM
- NIFS (if used as a stand-alone sensor)

Instruments are initialized when calling fgt_init or fgt_initEx. However directly calling any function automatically initializes the session. By default, instruments are sorted by their type then by their serial number in ascending order. The order of the types is as listed above and in fgt_INSTRUMENT_TYPE.

Initialization order also defines channel indexing 3.1. To initialize controllers in a specific order, call fgt_initEx, passing their serial numbers in the desired order.

Like channels, controllers also have a unique ID. It can be used to access the same instrument in different sessions, even if other instruments are present. Call fgt_get_controllersInfo to obtain the unique IDs of the initialized controllers, as well as their serial numbers, types and firmware versions (if available).

Before exiting your application, call fgt_close to close the SDK session. This frees the allocated memory and ensures that running threads are stopped properly. If this function is not called, the host application might throw an exception when exiting.

3.3 Regulation

Fluigent SDK embeds sensor regulation which automatically adjust pressure in order to reach sensor setpoint value.

Regulation can be started by calling fgt_set_sensorRegulation function. It links a pressure source to a sensor (basically a flow-unit). When running, sensor value is read then an algorithm computes required pressure command in order to reach sensor setpoint and a pressure controller applies this value. Regulation uses a sensor and a pressure channel (identified by their index or unique ID), multiples instances can be launched by calling same function with different index. Pressure controller or sensor can be changed on the run, without need of start/stop process.

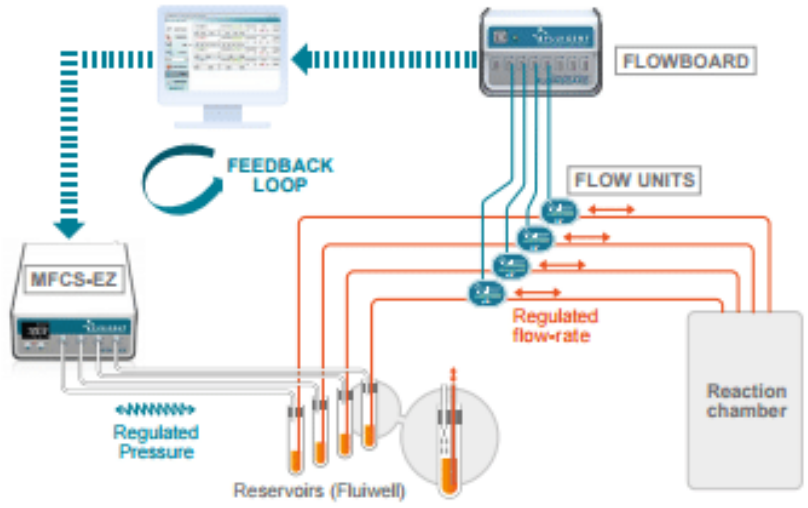

Figure 3.2: Sensor regulation feedback loop

The feedback loop is a “self-learning” algorithm to overcome typical issues in microfluidic flow rate control such as calibration, multi-channel interactions and resistance changes during experiments. Algorithm will adapt to any case and try to apply best pressure profile. If conditions are not optimal or if pressure controller range is not high enough to reach sensor setpoint, detailed information can retrieved when calling fgt_get_sensorStatus function. *infoCode* parameter is dedicated to regulation status, it is a mask of 8 bits:

infoCode	Description
0x00	No regulation is running on this sensor
0x01	Regulation is running in nominal state
0x02	Invalid microfuidic set-up
0x04	Low fluidic resistance
0x08	High fluidic resistance
0x10	Pressure limit reached
0x20	Flow-Rate limit reached
0x40	Command is not achievable
0x80	Reservoir may be empty

Microfluidic set-up, especially sensor sense has to be considered. Algorithm logic expects that sensor sense points out from the reservoir. In most cases, sensor sense has to be positive when fluid is going out of the reservoir. There is an exception on vacuum MFCS™ Series instruments, sensor sense has to be inverted from nominal case: sensor has to point towards the reservoir.

In order to stop a running regulation send a pressure order on regulated pressure channel. When exiting (by calling fgt_close function) regulation is also stopped.

3.3.1 Custom sensor regulation

Custom sensors, other than Fluigent ones can be used for pressure feedback loop regulation. It uses same algorithm than flow-rate regulation but can be used with different kind of sensors (pressure, liquid level, light...). Algorithm will try to adjust a Fluigent pressure controller in order to reach custom sensor setpoint.

fgt_set_customSensorRegulation function is dedicated to this purpose. It requires 4 parameters: custom sensor read value, setpoint, sensor maximum range and used pressure channel. Function has to be called at 1Hz or higher rate otherwise regulation is stopped. In this case data is considered not enough accurate to sustain a stable regulation.

There is no security on this feature and we do not guarantee full compatibility with used sensor.

3.4 Status management

When called, each function returns a status code. It is a byte casted in fgt_ERROR_CODE enum. If command was properly executed 0 value (Ok) is returned, otherwise a value indicates error status.

Pressure and sensor have dedicated advanced status functions:

- fgt_get_pressureStatus
- fgt_get_sensorStatus

In addition to error code, channel type, controller serial number, an information code and a detailed message are returned. This allow a detailed troubleshooting and display of custom messages.

Dll wrapper implements three status display functions:

- fgt_Manage_Pressure_Status
- fgt_Manage_Sensor_Status
- fgt_Manage_Generic_Status

You can decide a different action or different message display by modifying those functions.

4 | Software layers

The Fluigent SDK is based on a set of native libraries for each operating system. These libraries handle low-level communication with the supported instruments. Calling the native libraries directly is possible, but is recommended only for advanced users who are proficient in C or C++. When using the native libraries directly, the function signatures and descriptions can be found in the accompanying fgt_SDK.h header file. The same header file can be used for all versions of the library.

Additionally, more friendly packages and examples are provided for five major programming languages: C++, C#, Python, LabVIEW and MATLAB. They are collectively referred to as Middleware in this manual.

We strongly recommend using the Middleware if your programming language of choice is supported. It is open source, so you can modify it to suit your needs.

4.1 Fluigent SDK native shared libraries

The native shared libraries are sorted into folders by the operating system and processor architecture they target:

- **windows**/
  - x86/
    - fgt_SDK.dll
  - x64/
    - fgt_SDK.dll
- **linux**/
  - x64/
    - libfgt_SDK.so
  - arm/
    - libfgt_SDK.so
  - arm64/
    - libfgt_SDK.so
- **mac**/
  - x64/
    - libfgt_SDK.dylib
- fgt_SDK_32.dll
- fgt_SDK_64.dll
- fgt_SDK.h

Note that the fgt_SDK_32.dll and fgt_SDK_64.dll files are the same as the files in the windows/x86 and windows/x64 folders, respectively. They are copied for the sake of backwards compatibility and because this naming convention is more convenient for some programming languages, such as LabVIEW.

When targeting a single platform, you can keep only the corresponding library. To target multiple platforms, you have to write the logic to identify the platform and select the appropriate library to load. See the Middleware source code for examples on how to do that.

The shared libraries use the default calling convention for each operating system and architecture. In particular, __stdcall is used on Windows x86 (32 bits). On other systems, there is generally no need to specify the calling convention. Any language that interfaces with C should be able to access the library functions, as demonstrated in the Middleware source code.

The library functions are generally non-blocking and return immediately, with the exception of the functions that explicitly wait for the instruments, such as fgt_init and fgt_set_valvePosition when the "wait" parameter is set to true. Some functions can also become blocking if they are called too frequently compared to the data exchange rate with the instruments.

The data refresh rate varies between the instrument families:

- MFCS™, FRP™, ESS™: 100ms
- LineUP™: 20ms
- IPS: 10ms

These values represent expected response time both when reading and when setting values on the instrument. Calling "get" functions more frequently than these delays will simply return the same value repeatedly until it is updated by the instrument. Calling "set" functions more frequently might cause the library to block while it waits for the instrument to process the commands.

4.2 Middleware

The SDK middleware is a set of packages that make it easier to use the SDK with various programming languages.

The following programming languages and environments are supported:

Language	Package	Windows	Linux	macOS
C++	CMake project containing middleware and examples	✓	✓	✓
C#	fgt_SDK.cs middleware file fgt_SDK.sln Visual Studio complete solution containing middleware and examples	✓	✓	✓
Python	Fluigent.SDK package pip installer package	✓	✓	✓
LabVIEW	VIPM toolkit installer package	✓		
MATLAB	Toolbox installer package	✓		

The middleware packages provide the following functionalities for your convenience:

- Identify, locate and load the appropriate SDK native library according to the platform

- Convert data types to the native types used by each language to keep the user code clean
- Handle errors and display formatted error messages in program output

The middleware matches the conventions of each programming language while keeping the interface as similar as possible across all supported languages.

The following sections contain installation and usage instructions for each language.

4.3 Installation

The Fluigent SDK is available as a GitHub repository at https://github.com/Fluigent/fgt-SDK. The repository itself contains the source code of the Middleware, and the Releases page contains the compiled packages.

Feel free to open issues in the repository to give feedback, report problems, request features or ask for help. Please note that the repository issues are public. Do not include sensitive information in the issues. If your request must contain sensitive information, please send it to us directly using the contact information provided at the end of this manual.

On Linux, the system usually does not allow access to peripheral devices without superuser rights. It is possible to make an exception for Fluigent instruments, so the SDK can detect and communicate with them when run by a normal user. You can run the script `linux-udev.sh` that we provide, which automatically makes the necessary changes to the system. The script must be run with superuser rights.

4.3.1 C++

The C++ middleware consists of a CMake project that declares an interface library fgt_SDK_Cpp containing the SDK and the necessary logic to build on all supported operating systems. Linking your project against this library will give you access to the SDK functions and types. Examples are also included to demonstrate both the project creation with CMake and the SDK features.

The CMake project is compatible with Visual Studio 2019 on Windows. Please read the Readme.md file in the C++ directory for instructions on how to build the project from the command line and from various IDEs.

The middleware requires C++11 or later, and the CMake project requires CMake 3.13 or later.

Language specifics:

- The middleware function names are capitalized (e.g. Fgt_init() instead of fgt_init()) to avoid name collisions with the shared library functions that are included in the namespace. You can also call the shared library directly if you prefer.
- enum types are displayed as string by overriding « std::ostream operator
- The middleware calls functions such as Fgt_Manage_Pressure_Status after every function call, to report errors. By default, they print error messages and context (e.g. instrument type and index) to the console. You can modify these functions if you want a different form of error handling. The error code is also returned by the function.
- A post-build command is available to copy the appropriate shared library to the output directory, so the executable can find it. This is demonstrated in the examples.

4.3.2 C#

The C# middleware consists of a Visual Studio solution (fgt_sdk_csharp.sln) containing:

- a .NET Core 3.1 middleware fgt_SDK_Cpp.csproj project file

- several .NET Core examples, in the form of projects, demonstrating from basic to advanced features

You can add the middleware project to your own solution and reference it to have access to the SDK functions. When you build your project, the middleware copies the necessary native shared libraries to the output directory, so you can deploy them with your application. The middleware assembly is compatible with .NET Core 3.1 and .NET 5 and later.

A NuGet package is also provided. It is built from the middleware project mentioned above. The package is currently not available on any online package managers, but you can create a local feed to use it by following the instructions on https://docs.microsoft.com/en-us/nuget/hosting-packages/local-feeds.

Language specifics:

- one method manages low level returned codes. It is private to fgtSdk class: ErrCheck. You may want to change its behavior in order to throw exceptions that you can catch in your business layer

4.3.3 Python

The Python package is provided as a .zip file that can be installed using the pip or easy_install modules. This package can be installed in all supported operating systems and includes the necessary shared libraries. It is compatible with Python 3.1 and later.

```
python -m pip install -user fluigent_sdk-21.0.0.zip
python -m easy_install -user fluigent_sdk-21.0.0.zip
```

The –user option causes the package to be installed in the user's home directory, so that only the current user has access to it. In that case the installation does not require superuser rights. If the option is not used, the package will be installed in the system directory and all users will have access to it, and the installation will require superuser rights.

If you wish to bundle the package into your project instead of installing it, extract the .zip file and place the Fluigent folder in your project directory. As long as you work from that directory, the package will be available as if it were installed. This is convenient for redistributing projects that use the SDK.

Language specifics:

- The functions are located in the Fluigent.SDK module, as shown in the examples. They have the same name as the corresponding native library functions, but they return data by value (as a tuple if there is more than one value), instead of returning by reference like the C API.
- The "extended" functions (e.g. fgt_get_pressureEx) have been merged with the corresponding regular functions through the use of default arguments. Read the function docstrings for details.
- functions that return values (such as the functions starting with `fgt_get_*`) do not return the error code by default, to keep the user code more simple. They can be made to return the error code by setting the optional paramter `get_error` to True, in which case the functions will return a tuple with the error code as the first element.
- The Fluigent.SDK.exceptions module defines how the DLL error codes are handled. The default behavior is to log warning messages when errors occur. If you wish to change this behavior (e.g., raise exceptions for certain error codes), you can patch the functions inside this module, either at run time in your application or by editing the `exceptions.py` file.

4.3.4 LabVIEW

LabVIEW toolkit is only supported on Windows operating. It supports both 32 and 64 bit versions of LabVIEW starting with 2016 edition.

fgt_SDK toolkit for LabVIEW was built using JKI VI Package Manager (VIPM) software. JKI VIPM Free Edition can be downloaded from http://jki.net/vipm. 2016 or higher version is required to install the package. In

order to install the toolkit open or double click the VI Package (.vip) file. Select the LabVIEW version to install the palette in at the top of the VIPM window. Click the install (or upgrade) button and follow the wizard.

You can also clone the repository and use the code directly, by opening the LabVIEW project file in LabVIEW 2016 or higher.

Language specificities:

- error handling is automatically called each time a VI of the wrapper is called. Error numbers are copied from low level dlls call; you may want to change this behavior by modifying ErrorCodeToErrorStringConverter.vi in order to apply an offset on error codes

4.3.5 MATLAB

To install the toolbox, open the toolbox installer file "Fluigent SDK.mltbx". Fluigent toolbox is compatible with MATLAB R2015a and higher. Please contact us if you are using previous MATLAB versions. As mentioned at the start of this chapter, the MATLAB toolbox can only be used in the Windows operating system.

After the Toolbox installation is complete, all functions are available directly in the command window and scripts. The `help` and `doc` commands also work normally. You can also access the complete documentation in the Help menu (F1 keyboard shortcut). The documentation also includes examples that you can run and edit. Type `doc Fluigent` to open the list of toolbox functions and enumeration types.

Language specifics:

- The "extended" functions (e.g. fgt_get_pressureEx) have been merged with the corresponding regular functions through the use of MATLAB's `varargin` and `varargout` arguments. Read the function documentation for details on how to call each version of the function.
- the functions manage_generic_status, manage_pressure_status and manage_sensor_status define how the DLL error codes are handled. The default behavior is to generate warning messages when errors occur. If you wish to change this behavior (e.g., throw exceptions for certain error codes), you can do so by editing these functions. They are located in the Fluigent folder, along with the other SDK functions.
- most functions will return the error code returned by the shared library as the last returned value. See the function help for examples.

5 | Fluigent SDK Functions

5.1 Types definition

1. fgt_ERROR_CODE

   Returned status code when calling a function.

Value	Enum	Description
0	OK	No error
1	USB_error	USB communication error
2	Wrong_command	Wrong command was sent
3	No_module_at_index	There is no module initialized at selected index
4	Wrong_module	Wrong module was selected, unavailable feature
5	Module_is_sleep	Module is in sleep mode, orders are not taken into account
6	Master_error	Controller error
7	Failed_init_all_instr	Some instruments failed to initialize
8	Wrong_parameter	Function parameter is not correct or out of the bounds
9	Overpressure	Pressure module is in overpressure protection
10	Underpressure	Pressure module is in underpressure protection
11	No_instr_found	No Fluigent instrument was found
12	No_modules_found	No Fluigent pressure controller was found
13	No_pressure_controller_found	No Fluigent pressure controller was found
14	Calibrating	Pressure or sensor module is calibrating, read value may be incorrect
15	Dll_dependency_error	Some dependencies are not found
16	Processing	M-Switch is still rotating

2. fgt_INSTRUMENT_TYPE

   Type of available instruments.

Value	Enum	Description
0	None	None
1	MFCS	MFCS™ series instrument
2	MFCS_EZ	MFCS™-EZ instrument
3	FRP	Flowboard instrument
4	LineUP	LineUp series instrument (Link, Flow EZ)
5	IPS	Inline Pressure Sensor modules
6	ESS	Switchboard instrument
7	F_OEM	F-OEM instrument
9	NIFS	Non-Invasive Flow Sensor

3. fgt_SENSOR_TYPE

   Type of available sensors.

Value	Enum	Description
0	None	None
1	Flow_XS_single	XS flow-unit, H2O calibration
2	Flow_S_single	S flow-unit, H2O calibration
3	Flow_S_dual	S flow-unit, dual calibration H2O and IPA
4	Flow_M_single	M flow-unit, H2O calibration
5	Flow_M_dual	M flow-unit, dual calibration H2O and IPA. On FlowEZ also accepts HFE, FC40 and OIL
6	Flow_L_single	L flow-unit, H2O calibration
7	Flow_L_dual	L flow-unit, dual calibration H2O and IPA
8	Flow_XL_single	XL flow-unit, H2O calibration
9	Pressure_S	Inline Pressure Sensor range S
10	Pressure_M	Inline Pressure Sensor range M
11	Pressure_XL	Inline Pressure Sensor range XL
12	Flow_M_plus_dual	M+ flow-unit, dual calibration H2O and IPA. On FlowEZ also accepts HFE, FC40 and OIL
13	Flow_L_plus_dual	L+ flow-unit, dual calibration H2O and IPA. On FlowEZ also accepts HFE, FC40 and OIL
15	Flow_L_NIFS	Non-Invasive Flow Sensor

4. fgt_SENSOR_CALIBRATION

   Sensor available calibration table.

Value	Enum	Description
0	None	None
1	H2O	Water
2	IPA	Isopropanol
3	HFE	Hydrofluoroether
4	FC40	Fluorent electronic liquid
5	OIL	Oil

5. fgt_POWER

   Power state of the device.

Value	Enum	Description
0	POWER_OFF	Device is powered off
1	POWER_ON	Device is powered on
2	SLEEP	Device is in sleep mode

6. fgt_TTL_MODE

   TTL mode is used for TTL ports configuration. TTL low output is at 0V and high output at 5V. Ports can be configured as input or output (signal generator).

Value	Enum	Description
0	DETECT_RISING_EDGE	Detect a rising edge input signal
1	DETECT_FALLING_EDGE	Detect a falling edge input signal
2	OUTPUT_PULSE_LOW	Generate a low pulse for 100ms
3	OUTPUT_PULSE_HIGH	Generate a high pulse for 100ms

7. fgt_VALVE_TYPE

   Type of available valves.

Value	Enum	Description
0	None	None
1	MSwitch	M-Switch, 11/10 rotating valve
2	TwoSwitch	Two-Switch, 3/2 valve
3	LSwitch	L-Switch, 6/2 rotating valve
4	PSwitch	LineUP P-Switch, pressure toggle
5	M_X	M-X, 11/10 OEM rotating valve
6	Two_X	2-X, 3/2 OEM valve
7	L_X	L-X, 6/2 OEM rotating valve
8	Bypass	Bypass valve included in the NIFS sensors

8. fgt_SWITCH_DIRECTION

   Switch direction for M-Switches

Value	Enum	Description
0	Shortest	Direction of the shortest path
1	Anticlockwise	Always decrease the position
2	Clockwise	Always increase the position

9. fgt_CHANNEL_INFO

   This structure contains pressure and sensor identification and details.

Name	Data type	Description
ControllerSN	unsigned short	Serial number of this channel's controller
firmware	unsigned short	Firmware version of this channel (0 if not applicable)
DeviceSN	unsigned short	Serial number of this channel (0 if not applicable)
position	unsigned int	Position on controller
index	unsigned int	Channel index within its physical quantities family
indexID	unsigned int	Unique channel identifier
InstrType	fgt_INSTRUMENT_TYPE	Type of the instrument

10. fgt_CONTROLLER_INFO

    This structure contains controller identification and details.

Name	Data type	Description
SN	unsigned short	Controller serial number
Firmware	unsigned short	Controller firmware version
id	unsigned int	Index
InstrType	fgt_INSTRUMENT_TYPE	Type of the instrument

5.2 SDK Wrapper

Initialization and close

5.2.1 fgt_init

```
fgt_ERROR_CODE fgt_init(void);
```

Initialize or reinitialize (if already opened) Fluigent SDK instance. All detected Fluigent instruments (MFCS, MFCS-EZ, FRP, LineUP, IPS) are initialized. This function is optional, directly calling a function will automatically create the instance. Only one instance can be opened at a time. If called again, any new instruments are added to the same instance.

Returns

fgt_ERROR_CODE	enum	Returned status of function execution.

5.2.2 fgt_close

```
fgt_ERROR_CODE fgt_close(void);
```

Close communication with Fluigent instruments and free memory. This function is mandatory, if not called the dll will generate an exception when exiting your application. Using this function will remove session preferences such as units and limits. If any regulation is running it will stop pressure control.

Returns

fgt_ERROR_CODE	enum	Returned status of function execution.

5.2.3 fgt_detect

```
unsigned char fgt_detect(unsigned short SN[256], fgt_INSTRUMENT_TYPE type[256]);
```

Detects all connected Fluigent instrument(s), return their serial number and type.

Output

SN[256]	unsigned short	Array of controllers serial number. This is a 256 pre-allocated table tailed with 0's when no instrument
type[256]	fgt_INSTRUMENT_TYPE	This is a 256 pre-allocated table tailed with 'None' value when no instrument

Returns

return	unsigned char	Total number of detected instruments

5.2.4 fgt_initEx

```
unsigned char fgt_initEx(unsigned short SN[256]);
```

Initialize specific Fluigent instrument(s) from their unique serial number. This function can be used when multiple instruments are connected in order to select your device(s).

Output

SN[256]	unsigned short	Array of controllers serial numbers to be initialized. Fill with 0's if for no instrument .

Returns

fgt_ERROR_CODE	enum	Returned status of function execution.

Channels information

5.2.5 fgt_get_controllersInfo

```
fgt_ERROR_CODE fgt_get_controllersInfo(fgt_CONTROLLER_INFO info[256]);
```

Retrieve information about session controllers. Controllers are MFCS, Flowboard, Link, IPS in an array.

Output

info[256]	fgt_CONTROLLER_INFO	Array of structure containing information about each initialized controller. See details of fgt_CONTROLLER_INFO.

Returns

fgt_ERROR_CODE	enum	Returned status of function execution.

5.2.6 fgt_get_pressureChannelCount

```
fgt_ERROR_CODE fgt_get_pressureChannelCount(unsigned char* nbPChan);
```

Get total number of initialized pressure channels. It is the sum of all MFCS, MFCS-EZ and FlowEZ pressure controllers.

Output

nbPChan	unsigned char	Total number of initialized pressure channels

Returns

fgt_ERROR_CODE	enum	Returned status of function execution.

5.2.7 fgt_get_sensorChannelCount

```
fgt_ERROR_CODE fgt_get_sensorChannelCount(unsigned char* nbSChan);
```

Get total number of initialized sensor channels. It is the sum of all connected flow-units on Flowboard and FlowEZ, and IPS sensors.

Output

nbSChan	unsigned char	Total number of initialized sensor channels

Returns

fgt_ERROR_CODE	enum	Returned status of function execution.

5.2.8 fgt_get_TtlChannelCount

```
fgt_ERROR_CODE fgt_get_TtlChannelCount(unsigned char* nbTtlChan);
```

Get total number of initialized sensor channels. It is the sum of all connected flow-units on Flowboard and FlowEZ.

Output

nbTtlChan	unsigned char	Total number of initialized TTL channels

Returns

fgt_ERROR_CODE	enum	Returned status of function execution.

5.2.9 fgt_get_valveChannelCount

```
fgt_ERROR_CODE fgt_get_valveChannelCount(unsigned char* nbValveChan);
```

Get total number of initialized valve channels. It is the sum of all connected Two-Switch, L-Switch and M-Switch valves connected to Switchboard or Switch EZ devices, as well as individual P-Switch outputs (8 per device).

Output

nbValveChan	unsigned char	Total number of initialized valve channels

Returns

fgt_ERROR_CODE	enum	Returned status of function execution.

5.2.10 fgt_get_pressureChannelsInfo

```
fgt_ERROR_CODE fgt_get_pressureChannelsInfo(fgt_CHANNEL_INFO info[256]);
```

Retrieve information about each initialized pressure channel. This function is useful in order to get channels order, controller, unique ID and instrument type. By default this array is built with MFCS first, MFCS-EZ second and FlowEZ last. If only one instrument is used, index is the default channel indexing starting at 0. You can initialize instruments in specific order using fgt_initEx function.

Output

info[256]	fgt_CHANNEL_INFO	Array of structure containing channel details

Returns

fgt_ERROR_CODE	enum	Returned status of function execution.

5.2.11 fgt_get_sensorChannelsInfo

```
fgt_ERROR_CODE fgt_get_sensorChannelsInfo(fgt_CHANNEL_INFO info[256],
    fgt_SENSOR_TYPE sensorType[256]);
```

Retrieve information about each initialized sensor channel. This function is useful in order to get channels order, controller, unique ID and instrument type. By default this array is built with FRP Flow Units first, followed by Flow EZ Flow Units, followed by IPS modules. If only one instrument is used, index is the default channel indexing starting at 0. You can initialize instruments in specific order using fgt_initEx function.

Output

info[256]	fgt_CHANNEL_INFO	Array of structure containing channel details
sensorType[256]	fgt_SENSOR_TYPE	Array containing sensor types

Returns

fgt_ERROR_CODE	enum	Returned status of function execution.

5.2.12 fgt_get_TtlChannelsInfo

```
fgt_ERROR_CODE fgt_get_TtlChannelsInfo(fgt_CHANNEL_INFO info[256]);
```

Retrieve information about each initialized TTL channel. This function is useful in order to get channels order, controller, unique ID and instrument type. TTL channels are only available for LineUP Series, 2 ports for each connected Link.

Output

info[256]	fgt_CHANNEL_INFO	Array of structure containing channel details

Returns

fgt_ERROR_CODE	enum	Returned status of function execution.

5.2.13 fgt_get_valveChannelsInfo

```
fgt_ERROR_CODE fgt_get_valveChannelsInfo(fgt_CHANNEL_INFO info[256],
    fgt_valve_t valveType[256]);
```

Retrieve information about each initialized valve channel. This function is useful in order to get channels order, controller, unique ID and instrument type. By default this array is built with LineUp valves first (connected to SwitchEz or P-Switch) followed by ESS Switchboard valves. If only one instrument is used, index is the default channel indexing starting at 0. You can initialize instruments in specific order using fgt_initEx function.

Output

info[256]	fgt_CHANNEL_INFO	Array of structure containing channel details
valveType[256]	fgt_VALVE_TYPE	Array containing valve types

Returns

fgt_ERROR_CODE	enum	Returned status of function execution.

5.2.14 fgt_set_pressure

```
fgt_ERROR_CODE fgt_set_pressure(unsigned int pressureIndex, float pressure);
```

Send pressure command to selected device.

Parameters

pressureIndex	unsigned int	Index of pressure channel or unique ID
pressure	float	Pressure order in selected unit, default is "mbar"

Returns

fgt_ERROR_CODE	enum	Returned status of function execution.

5.2.15 fgt_get_pressure

```
fgt_ERROR_CODE fgt_get_pressure(unsigned int pressureIndex, float *pressure);
```

Read pressure value of selected device.

Parameter

pressureIndex	unsigned int	Index of pressure channel or unique ID

Output

pressure	float	Read pressure value in selected unit, default is "mbar"

Returns

fgt_ERROR_CODE	enum	Returned status of function execution.

5.2.16 fgt_get_pressureEx

```
fgt_ERROR_CODE fgt_get_pressureEx(unsigned int pressureIndex, float *pressure,
    unsigned short *timeStamp);
```

Read pressure value and time stamp of selected device. Time stamp is the device internal timer.

Parameter

pressureIndex	unsigned int	Index of pressure channel or unique ID

Output

pressure	float	Read pressure value in selected unit, default is "mbar"
timeStamp	unsigned short	Hardware timer in ms

Returns

fgt_ERROR_CODE	enum	Returned status of function execution.

5.2.17 fgt_set_sensorRegulation

```
fgt_ERROR_CODE fgt_set_sensorRegulation(unsigned int sensorIndex, unsigned int
    pressureIndex, float setpoint);
```

Start closed loop regulation between a sensor and a pressure controller. Pressure will be regulated in order to reach sensor setpoint. Call again this function in order to change the setpoint. Calling fgt_set_pressure on same pressureIndex will stop regulation.

Parameters

sensorIndex	unsigned int	Index of sensor channel or unique ID
pressureIndex	unsigned int	Index of pressure channel or unique ID
setpoint	float	Regulation value to be reached in selected unit, default is "µl/min" for flowrate sensors

Returns

fgt_ERROR_CODE	enum	Returned status of function execution.

5.2.18 fgt_get_sensorValue

```
fgt_ERROR_CODE fgt_get_sensorValue(unsigned int sensorIndex, float *value);
```

Read sensor value of selected device.

Parameter

sensorIndex	unsigned int	sensorIndex Index of sensor channel or unique ID

Output

value	float	Read sensor value in selected unit, default is "µl/min" for flowrate sensors and "mbar" for pressure sensors

Returns

fgt_ERROR_CODE	enum	Returned status of function execution.

5.2.19 fgt_get_sensorValueEx

```
fgt_ERROR_CODE fgt_get_sensorValueEx(unsigned int sensorIndex, float* value,
    unsigned short* timeStamp);
```

Read sensor value and timestamp of selected device. Time stamp is the device internal timer.

Parameter		
sensorIndex	unsigned int	Index of sensor channel or unique ID
Output		
value	float	Read sensor value in selected unit, default is "µl/min" for flowrate sensors
timeStamp	unsigned short	Hardware timer in ms
Returns		
fgt_ERROR_CODE	enum	Returned status of function execution.

5.2.20 fgt_get_sensorAirBubbleFlag

```
fgt_ERROR_CODE fgt_get_sensorAirBubbleFlag(unsigned int sensorIndex,
    unsigned char* detected);
```

Read the flag indicating whether the flow rate sensor detects an air bubble. Only available on Flow Unit sensor ranges M+ and L+.

Parameter		
sensorIndex	unsigned int	sensorIndex Index of sensor channel or unique ID
Output		
detected	unsigned char	1 if an air bubble was detected, 0 otherwise.
Returns		
fgt_ERROR_CODE	enum	Returned status of function execution.

5.2.21 fgt_get_valvePosition

```
fgt_ERROR_CODE fgt_get_valvePosition(unsigned int valveIndex, int*
    position);
```

Read the position of a specific valve channel.

Parameter		
valveIndex	unsigned int	Index of valve channel or unique ID
Output		
position	int	Current position of the valve
Returns		
fgt_ERROR_CODE	enum	Returned status of function execution.

5.2.22 fgt_set_valvePosition

```
fgt_ERROR_CODE fgt_set_valvePosition(unsigned int valveIndex, int
    position, fgt_SWITCH_DIRECTION direction, int wait);
```

Set the position of a specific valve channel.

Parameters

valveIndex	unsigned int	Index of valve channel or unique ID
position	int	Desired valve position
direction	fgt_SWITCH_DIRECTION	Direction of the movement (applies only for M-Switch valve type)
wait	int	Flag indicating if function should wait until the desired position is reached or not

Returns

fgt_ERROR_CODE	enum	Returned status of function execution.

5.2.23 fgt_set_allValves

```
fgt_ERROR_CODE fgt_set_allValves(unsigned int controllerIndex, unsigned
    int moduleIndex, int position);
```

Set the position of all two positional valves connected to specified controller / module.

Parameters

valveIndex	unsigned int	Index of valve channel or unique ID
position	int	Desired valve position
direction	int	Direction of the movement (applies only for M-Switch valve type)
wait	int	Flag indicating if function should wait until the desired position is reached or not

Returns

fgt_ERROR_CODE	enum	Returned status of function execution.

Unit, calibration and limits

5.2.24 fgt_set_sessionPressureUnit

```
fgt_ERROR_CODE fgt_set_sessionPressureUnit(std::string unit);
```

Set pressure unit for all initialized channels, default value is "mbar". If type is invalid an error is returned.
Every pressure read value and sent command will then use this unit.
Example of type: "mbar", "millibar", "kPa" ...

Parameters

unit	std::string	Unit string

Returns

fgt_ERROR_CODE	enum	Returned status of function execution.

5.2.25 fgt_set_pressureUnit

```
fgt_ERROR_CODE fgt_set_pressureUnit(unsigned int presureIndex, std::string
    unit);
```

Set pressure unit on selected pressure device, default value is "mbar". If type is invalid an error is returned. Every pressure read value and sent command will then use this unit.
Example of type: "mbar", "millibar", "kPa" ...

Parameters

presureIndex	unsigned int	Index of pressure channel or unique ID
unit	std::string	Channel unit string

Returns

fgt_ERROR_CODE	enum	Returned status of function execution.

5.2.26 fgt_get_pressureUnit

```
fgt_ERROR_CODE fgt_get_pressureUnit(unsigned int presureIndex, std::string
    *unit);
```

Get used unit on selected pressure device, default value is "mbar". Every pressure read value and sent command use this unit.

Parameter

presureIndex	unsigned int	Index of pressure channel or unique ID

Output

unit	std::string	Channel unit string

Returns

fgt_ERROR_CODE	enum	Returned status of function execution.

5.2.27 fgt_set_sensorUnit

```
fgt_ERROR_CODE fgt_set_sensorUnit(unsigned int sensorIndex, std::string unit);
```

Set sensor unit on selected sensor device, default value is "µl/min" for flowrate sensors and "mbar" for pressure sensors. If type is invalid an error is returned. Every sensor read value and regulation command will then use this unit.
Example of type: "µl/h", "ulperDay", "microliter/hour" ...

Parameters

sensorIndex	unsigned int	Index of sensor channel or unique ID
unit	std::string	Channel unit string

Returns

fgt_ERROR_CODE	enum	Returned status of function execution.

5.2.28 fgt_get_sensorUnit

```
fgt_ERROR_CODE fgt_get_sensorUnit(unsigned int sensorIndex, std::string *unit);
```

Get used unit on selected sensor device, default value is "µl/min" for flowunits and "mbar" for pressure sensors. Every sensor read value and regulation command use this unit.

Parameters

sensorIndex	unsigned int	Index of sensor channel or unique ID

Output

unit	std::string	Channel unit string

Returns

fgt_ERROR_CODE	enum	Returned status of function execution.

5.2.29 fgt_set_sensorCalibration

```
fgt_ERROR_CODE fgt_set_sensorCalibration(unsigned int sensorIndex,
    fgt_SENSOR_CALIBRATION calibration);
```

Set sensor internal calibration table. Function is only available for IPS (to set new reference value "zero") and specific flowrate sensors (dual type) such as the flow-unit M accepting H2O and IPA

Parameters

sensorIndex	unsigned int	Index of sensor channel or unique ID
fgt_SENSOR_CALIBRATION	enum	Channel calibration table

Returns

fgt_ERROR_CODE	enum	Returned status of function execution.

5.2.30 fgt_get_sensorCalibration

```
fgt_ERROR_CODE fgt_get_sensorCalibration(unsigned int sensorIndex,
    fgt_SENSOR_CALIBRATION *calibration);
```

Get internal calibration table used by the sensor. Only applicable to Flow Unit sensors.

Parameter

sensorIndex	unsigned int	Index of sensor channel or unique ID

Output

fgt_SENSOR_CALIBRATION	enum	Channel calibration table

Returns

fgt_ERROR_CODE	enum	Returned status of function execution.

5.2.31 fgt_set_sensorCustomScale

```
fgt_ERROR_CODE fgt_set_sensorCustomScale(unsigned int sensorIndex, float a,
    float b, float c);
```

Apply a custom scale factor on flow rate sensor measurement. This function is useful in order to calibrate the sensor to a specific liquid that is not officially supported. The scale factor is applied using following formula:

$$scaled\_value = a * sensor\_value + b * sensor\_value^2 + c * sensor\_value^3$$

Note that this scale is also used for the regulation. Only supported by Flow Unit sensors.
The polynomial is always applied to the value in µL/min, regardless of the unit set on the sensor, to prevent accidentally changing the calibration by changing the unit. If you calculated your polynomial for a different unit, you will have to adapt the "b" and "c" coefficients accordingly. The "a" coefficient is linear, so it is the same for all units.

Parameters

sensorIndex	unsigned int	Index of sensor channel or unique ID
a	float	Proportional multiplier value
b	float	Square multiplier value
c	float	Cubic multiplier value

Returns

fgt_ERROR_CODE	enum	Returned status of function execution.

5.2.32 fgt_set_sensorCustomScaleEx

```
fgt_ERROR_CODE fgt_set_sensorCustomScaleEx(unsigned int sensorIndex, float a,
    float b, float c, float SMax);
```

Apply a custom scale factor on flow rate sensor measurement. This function is useful in order to calibrate the sensor to a specific liquid that is not officially supported. Scale factor is applied using following formula:

$$scaled\_value = a * sensor\_value + b * sensor\_value^2 + c * sensor\_value^3$$

The result is then clamped between -SMax and SMax.
This function is intended to limit the sensor's scaled range to a reasonable value when only part of its original range is used with the given liquid, especially if the polynomial is not well conditioned outside of this range.
Note that the scaled value is also used for the regulation. Only supported by Flow Unit sensors.
The polynomial is always applied to the sensor measurement in µL/min, and the SMax value is also assumed to be in µL/min, regardless of the unit set on the sensor, to prevent accidentally changing the calibration by changing the unit. If you calculated your polynomial for a different unit, you will have to adapt the "b" and "c" coefficients and the SMax value accordingly. The "a" coefficient is linear, so it is same for all units.

Parameters

sensorIndex	unsigned int	Index of sensor channel or unique ID
a	float	Proportional multiplier value
b	float	Square multiplier value
c	float	Cubic multiplier value
SMax	float	After scale maximal value (saturation)

Returns

fgt_ERROR_CODE	enum	Returned status of function execution.

5.2.33 fgt_calibratePressure

```
fgt_ERROR_CODE fgt_calibratePressure(unsigned int presureIndex);
```

Calibrate internal pressure sensor depending on atmospheric pressure. After calling this function 0 pressure value corresponds to atmospheric pressure. During calibration step no pressure order is accepted. Total duration vary from 3s to 8s.

Parameters

presureIndex	unsigned int	Index of pressure channel or unique ID

Returns

fgt_ERROR_CODE	enum	Returned status of function execution.

5.2.34 fgt_set_customSensorRegulation

```
fgt_ERROR_CODE fgt_set_customSensorRegulation(float measure, float setpoint,
    float maxSensorRange, unsigned int pressureIndex);
```

Start closed loop regulation between a sensor and a pressure controller. Pressure will be regulated in order to reach sensor setpoint. Custom sensors, outside Fluigent ones, can be used such as different flow-units, pressure, level... However we do not guarantee full compatibility with all sensors. Regulation quality is linked to sensor precision and your set-up.
In order to use this function, custom used sensor maximum range and measured values has to be updated at least once per second. Directly setting pressure on same pressureIndex will stop regulation.
This function must be called at 1Hz minimum or the regulation will stop.

Parameters

measure	float	Custom sensor measured value, no unit is required
setpoint	float	Custom sensor regulation goal value, no unit is required
maxSensorRange	float	Custom sensor maximum range, no unit is required
pressureIndex	unsigned int	Index of pressure channel or unique ID

Returns

fgt_ERROR_CODE	enum	Returned status of function execution.

5.2.35 fgt_get_pressureRange

```
fgt_ERROR_CODE fgt_get_pressureRange(unsigned int pressureIndex, float *Pmin,
    float *Pmax);
```

Get pressure controller minimum and maximum range. Returned values takes into account set unit, default value is 'mbar'.

Parameter

pressureIndex	unsigned int	Index of pressure channel or unique ID

Output

Pmin	float	Minimum device pressure
Pmax	float	Maximum device pressure

Returns

fgt_ERROR_CODE	enum	Returned status of function execution.

5.2.36 fgt_get_sensorRange

```
fgt_ERROR_CODE fgt_get_sensorRange(unsigned int sensorIndex, float* Smin,
    float* Smax);
```

Get sensor minimum and maximum range. Returned values takes into account set unit, default value is 'µl/min' in case of flow-units and 'mbar' for pressure sensors.

Parameter

sensorIndex	unsigned int	Index of sensor channel or unique ID

Output

Smin	float	Minimum measured sensor value
Smax	float	Maximum measured sensor value

Returns

fgt_ERROR_CODE	enum	Returned status of function execution.

5.2.37 fgt_get_valveRange

```
fgt_ERROR_CODE fgt_get_valveRange(unsigned int valveIndex, int* posMax);
```

Get valve maximum position. Position indexing starts at 0.

Parameter

valveIndex	unsigned int	Index of valve channel or unique ID

Output

posMax	int	Maximum valve position

Returns

fgt_ERROR_CODE	enum	Returned status of function execution.

5.2.38 fgt_set_pressureLimit

```
fgt_ERROR_CODE fgt_set_pressureLimit(unsigned int pressureIndex, float PlimMin,
    float PlimMax);
```

Set pressure working range and ensure that pressure will never exceed this limit. It takes into account current unit, default value is 'mbar'.

Parameters

sensorIndex	unsigned int	Index of sensor channel or unique ID
PlimMin	float	Minimum admissible device pressure
PlimMax	float	Maximum admissible device pressure

Returns

fgt_ERROR_CODE	enum	Returned status of function execution.

Regulation settings

5.2.39 fgt_set_sensorRegulationResponse

```
fgt_ERROR_CODE fgt_set_sensorRegulationResponse(unsigned int sensorIndex,
    unsigned int responseTime);
```

Set on a running regulation pressure response time. Minimal value is 2 for FlowEZ, 6 for MFCS controllers.

Parameters

sensorIndex	unsigned int	Index of sensor channel or unique ID
responseTime	unsigned int	Pressure response time in seconds

Returns

fgt_ERROR_CODE	enum	Returned status of function execution.

5.2.40 fgt_set_sensorRegulationInverted

```
fgt_ERROR_CODE fgt_set_sensorRegulationInverted(unsigned int
    sensorIndex, unsigned char inverted);
```

Specify whether the sensor is inverted in the physical setup, i.e., if an increase in pressure causes the sensor value to decrease and vice-versa.

Parameters

sensorIndex	unsigned int	Index of sensor channel or unique ID
inverted	unsigned char	0: not inverted, 1: inverted

Returns

fgt_ERROR_CODE	enum	Returned status of function execution.

5.2.41 fgt_set_pressureResponse

```
fgt_ERROR_CODE fgt_set_pressureResponse(unsigned int sensorIndex, unsigned char
    value);
```

Set pressure controller response. This function can be used to customise response time for your set-up.
For FlowEZ available values are 0: use of fast switch vales or 1: do not use fast switch vales. Default value is 0.
For MFCS available values are from 1 to 255. Higher the value, longer is the response time. Default value is 5.

Parameters

sensorIndex	unsigned int	Index of sensor channel or unique ID
value	unsigned char	Desired pressure controller response time, this depends on controller type

Returns

fgt_ERROR_CODE	enum	Returned status of function execution.

Status information

5.2.42 fgt_get_pressureStatus

```
fgt_ERROR_CODE fgt_get_pressureStatus(unsigned int pressureIndex,
    fgt_INSTRUMENT_TYPE *type, unsigned short *controllerSN, unsigned char
    *infoCode, std::string *detail);
```

Get detailed information of pressure channel status. This function is meant to be invoked after calling a pressure related function which returns an error code.
Retrieved information of last error contains controller position and a string detail.

Parameter

pressureIndex	unsigned int	Index of pressure channel or unique ID

Output

type	fgt_INSTRUMENT_TYPE	Controller type
controllerSN	unsigned short	Serial number of controller (such as Link, MFCS)
infoCode	unsigned char	Information status code, 1 if pressure module is controller locally
detail	std::string	Detailed string about the error or state

Returns

fgt_ERROR_CODE	enum	Returned status of function execution.

5.2.43 fgt_get_sensorStatus

```
fgt_ERROR_CODE fgt_get_sensorStatus(unsigned int sensorIndex,
    fgt_INSTRUMENT_TYPE* type, unsigned short* controllerSN, unsigned char*
    infoCode, std::string* detail);
```

Get detailed information of sensor status. This function is ment to be invoked after calling a sensor related function which returns an error code.
Retrieved information of last error contains sensor position and a string detail.

Parameter

sensorIndex	unsigned int	Index of sensor channel or unique ID

Output

type	fgt_INSTRUMENT_TYPE	Controller type
controllerSN	unsigned short	Serial number of controller (such as Link, Flowboard)
infoCode	unsigned char	Information status code about regulation See infoCode for more details
detail	std::string	Detailed string about the error or state

Returns

fgt_ERROR_CODE	enum	Returned status of function execution.

5.2.44 fgt_set_power

```
fgt_ERROR_CODE fgt_set_power(unsigned short controllerIndex, fgt_POWER
    powerState);
```

Set power ON or OFF on a controller (such as Link, MFCS, Flowboard).
Not all controllers support this functionality.

Parameter

controllerIndex	unsigned int	Index of controller or unique ID
powerState	fgt_POWER	Power mode to set.

Returns

fgt_ERROR_CODE	enum	Returned status of function execution.

5.2.45 fgt_get_power

```
fgt_ERROR_CODE fgt_get_power(unsigned short controllerIndex, fgt_POWER
    *powerState);
```

Get power information about a controller (such as Link, MFCS, Flowboard).
Not all controllers support this functionality.

Paramete

controllerIndex	unsigned int	Index of controller or unique ID

Output

powerState	fgt_POWER	Power mode of the device.

Returns

fgt_ERROR_CODE	enum	Returned status of function execution.

TTL functions

5.2.46 fgt_set_TtlMode

```
fgt_ERROR_CODE fgt_set_TtlMode(unsigned int TtlIndex, fgt_TTL_MODE mode);
```

Configure a specific TTL port (BNC ports) as input, output, rising or falling edge.

Parameters

TtlIndex	unsigned int	TtlIndex Index of TTL port or unique ID
mode	fgt_TTL_MODE	TTL mode to set.

Returns

fgt_ERROR_CODE	enum	Returned status of function execution.

5.2.47 fgt_read_Ttl

```
fgt_ERROR_CODE fgt_read_Ttl(unsigned int TtlIndex, unsigned int *state);
```

Read TTL port (BNC port) if set as input.

Parameter

TtlIndex	unsigned int	TtlIndex Index of TTL port or unique ID

Output

state	unsigned int	0: no edge was detected; 1: an edge is detected

Returns

fgt_ERROR_CODE	enum	Returned status of function execution.

5.2.48 fgt_trigger_Ttl

```
fgt_ERROR_CODE fgt_trigger_Ttl(unsigned int TtlIndex);
```

Trigger a specific TTL port (BNC ports) if set as output.

Parameter

TtlIndex	unsigned int	TtlIndex Index of TTL port or unique ID

Returns

fgt_ERROR_CODE	enum	Returned status of function execution.

Specific functions

5.2.49 fgt_set_purge

```
fgt_ERROR_CODE fgt_set_purge(unsigned short controllerIndex, unsigned char
    purge);
```

Activate/deactivate purge function.
This feature is only available on MFCS devices equipped with special valve.

Parameters

controllerIndex	unsigned int	Index of controller or unique ID
purge	unsigned char	0: OFF, 1:ON

Returns

fgt_ERROR_CODE	enum	Returned status of function execution.

5.2.50 fgt_set_manual

```
fgt_ERROR_CODE fgt_set_manual(unsigned int pressureIndex, float value);
```

Manually set the voltage of the pressure channel's input solenoid valve.
This stops pressure regulation until a new pressure command is sent.

Parameters

pressureIndex	unsigned int	Index of pressure channel or unique ID
value	float	applied valve voltage from 0 to 100(%)

Returns

fgt_ERROR_CODE	enum	Returned status of function execution.

5.2.51 fgt_set_digitalOutput

```
fgt_ERROR_CODE fgt_set_digitalOutput(unsigned short controllerIndex,
    unsigned char port, unsigned char state);
```

Set a digital output ON or OFF on a controller
This feature is only available on the F-OEM device.

Parameters

controllerIndex	unsigned int	Index of controller or unique ID
port	unsigned char	Address of the digital output to toggle. For F-OEM: 0: Pump, 1: LED
state	unsigned char	0: OFF, 1:ON

Returns

fgt_ERROR_CODE	enum	Returned status of function execution.

5.2.52 fgt_get_inletPressure

```
fgt_ERROR_CODE fgt_get_inletPressure(unsigned int pressureIndex, float
    *pressure);
```

Returns the pressure measured at the device's inlet. This feature is only available on LineUP Flow EZ and FOEM Pressure Module instruments.

Parameter

pressureIndex	unsigned int	Index of pressure channel or unique ID

Output

pressure	float	Inlet pressure value in selected unit, default is "mbar"

Returns

fgt_ERROR_CODE	enum	Returned status of function execution.

5.2.53 fgt_create_simulated_instr

```
fgt_ERROR_CODE fgt_create_simulated_instr(fgt_INSTRUMENT_TYPE type,
    unsigned short serial, unsigned short version, int[] config, int
    length);
```

Creates a simulated Fluigent instrument, which can be detected and initialized like a real one, for the purposes of testing and demonstrations.

Parameters

type	fgt_INSTRUMENT_TYPE	Type of instrument to simulate
serial	unsigned short	Serial number for the simulated instrument
version	unsigned short	Firmware version for the simulated instrument. Set to 0 to use the default version
config	int[]	Array describing the instrument's configuration. See chapter 7 for details
length	unsigned short	Length of the config array

Returns

fgt_ERROR_CODE	enum	Returned status of function execution.

5.2.54 fgt_remove_simulated_instr

```
fgt_ERROR_CODE fgt_create_simulated_instr(fgt_INSTRUMENT_TYPE type,
    unsigned short serial);
```

Removes a simulated instrument that had been previously created. If it had already been initialized by the SDK, the controller and channels will remain in the respective lists, but they will act as if the instrument is missing. This is equivalent to physically disconnecting a real instrument.

Parameters

type	fgt_INSTRUMENT_TYPE	Type of instrument to remove
serial	unsigned short	Serial number of the simulated instrument

Returns

fgt_ERROR_CODE	enum	Returned status of function execution.

5.2.55 fgt_get_differentialPressureRange

```
fgt_ERROR_CODE fgt_get_differentialPressureRange(unsigned int
    sensorIndex, float* Pmin, float* Pmax);
```

Returns the range of the differential pressure sensor in mbar. This feature is only available on NIFS devices.

Parameter

sensorIndex	unsigned int	Index of sensor channel or unique ID

Output

Pmin	float	Minimum differential pressure in mbar
Pmax	float	Maximum differential pressure in mbar

Returns

fgt_ERROR_CODE	enum	Returned status of function execution.

5.2.56 fgt_get_differentialPressure

```
fgt_ERROR_CODE fgt_get_differentialPressureRange(unsigned int
    sensorIndex, float* Pdiff);
```

Returns the current differential pressure measurement in mbar. This feature is only available on NIFS devices.

Parameter

sensorIndex	unsigned int	Index of sensor channel or unique ID

Output

Pdiff	float	differential pressure in mbar

Returns

fgt_ERROR_CODE	enum	Returned status of function execution.

5.2.57 fgt_get_absolutePressureRange

```
fgt_ERROR_CODE fgt_get_absolutePressureRange(unsigned int sensorIndex,
    float* Pmin, float* Pmax);
```

Returns the range of the absolute pressure sensor in mbar. This feature is only available on NIFS devices.

Parameter

sensorIndex	unsigned int	Index of sensor channel or unique ID

Output

Pmin	float	Minimum absolute pressure in mbar
Pmax	float	Maximum absolute pressure in mbar

Returns

fgt_ERROR_CODE	enum	Returned status of function execution.

5.2.58 fgt_get_absolutePressure

```
fgt_ERROR_CODE fgt_get_absolutePressureRange(unsigned int sensorIndex,
    float* Pdiff);
```

Returns the current absolute pressure measurement in mbar. This feature is only available on NIFS devices.

Parameter

sensorIndex	unsigned int	Index of sensor channel or unique ID

Output

Pabs	float	absolute pressure in mbar

Returns

fgt_ERROR_CODE	enum	Returned status of function execution.

5.2.59 fgt_get_sensorBypassValve

```
fgt_ERROR_CODE fgt_get_sensorBypassValve(unsigned int sensorIndex,
    unsigned char* state);
```

Returns the current state of the bypass valve. This feature is only available on NIFS devices.

Parameter

sensorIndex	unsigned int	Index of sensor channel or unique ID

Output

state	unsigned char	1 if the valve is open, 0 if it is closed.

Returns

fgt_ERROR_CODE	enum	Returned status of function execution.

5.2.60 fgt_set_sensorBypassValve

```
fgt_ERROR_CODE fgt_set_sensorBypassValve(unsigned int sensorIndex,
    unsigned char state);
```

Sets the state of the sensor's bypass valve. This feature is only available on NIFS devices.

Parameter

sensorIndex	unsigned int	Index of sensor channel or unique ID
state	unsigned char	1 to open, 0 to close

Returns

fgt_ERROR_CODE	enum	Returned status of function execution.

5.2.61 fgt_set_log_verbosity

```
fgt_ERROR_CODE fgt_set_log_verbosity(unsigned int verbosity);
```

Sets the verbosity of the logging feature, i.e., how much data is logged.

Parameter

verbosity	unsigned int	The amount of data to log. Set to 0 to disable logging (default).

Returns

fgt_ERROR_CODE	enum	Returned status of function execution.

5.2.62 fgt_set_log_output_mode

```
fgt_ERROR_CODE fgt_set_log_output_mode(unsigned char output_to_file,
    unsigned char output_to_stderr, unsigned char output_to_queue);
```

Sets how the SDK outputs the log entries.

Parameter

output_to_file	unsigned char	Output log entries to a file in the current directory. 1 to enable, 0 to disable. Default: enabled.
output_to_stderr	unsigned char	Output log entries to the stderr pipe (console). 1 to enable, 0 to disable. Default: disabled.
output_to_queue	unsigned char	Store log entries in memory. They can be retrieved via the fgt_get_next_log function. 1 to enable, 0 to disable. Default: disabled.

Returns

fgt_ERROR_CODE	enum	Returned status of function execution.

5.2.63 fgt_get_next_log

```
fgt_ERROR_CODE fgt_get_next_log(char log[4000]);
```

Returns the next log entry stored in memory, if any, and removes it from the queue. Will return an error if the queue is empty. Logs are only stored in memory if the corresponding option is set with the fgt_set_log_output_mode function. Call this function repeatedly until an error is returned to retrieve all log entries.

Parameter

log	char[4000]	Array provided by the user, on which the log string will be copied. Must have at least 4000 bytes of available space.

Returns

fgt_ERROR_CODE	enum	Returned status of function execution.

5.3 Type equivalence

The following table presents the types various programming languages which are equivalent to the types used by the shared library functions, for the purposes of interoperability. If you use the middleware, these conversions are already taken care of.

Bits	C++	C#	Python	LabVIEW	MATLAB
32	long int	int	c_ulong	I32	Int32
64	unsigned long long	ulong	c_ulonglong	U64	Uint64
	pointer to unsigned short	ushort by ref	byref(c_ushort)	pointer U16	Uint16Ptr
8	unsigned char	byte	c_ubyte	U8	Uint8
	pointer to unsigned char	byte by ref	byref(c_ubyte)	U8	Uint8Ptr
32	float	float	c_float	SGL	float
16	unsigned short	ushort	c_ushort	U16	Int16
	char[]	byte[]	Array(c_uchar *)	pointer to array of U8	Uint8[]

6 | Examples

Multiple examples are provided with the middleware. We made them as much as possible, easily available in each IDE. Examples goes from basic use to more advanced features. We first recommend to get working basic examples before going on to more advanced ones.

6.1 Basic Read Sensor Data

This example shows how to retrieve a data from the sensor channel

Hardware setup: One or more connected devices with sensor channel(s) or standalone sensor device(s).

Pseudo-code	Wrapper call
1. Initialize session	fgt_init
2. Get total number of initialized sensor channel(s)	fgt_get_sensorChannelCount
3. Get information about the connected sensor channel(s)	fgt_get_sensorChannelsInfo
5. Retrieve sensor(s) unit	fgt_get_sensorUnit
6. Get sensor(s) range	fgt_get_sensorRange
7. Read sensor(s) data in loop	fgt_get_sensorValue
7. Close session	fgt_close

6.2 Basic Set Pressure

This example shows how to set a pressure order and generate a ramp on the first pressure module of the chain.

Required hardware: - at least one Fluigent pressure controller (MFCS, MFCS-EZ or FlowEZ)

Pseudo-code	Wrapper call
1. Initialize session	fgt_init
2. Set pressure	fgt_set_pressure
3. Wait 5 seconds letting pressure to establish	
4. Read and display pressure	fgt_get_pressure
5. Get pressure controller range	fgt_get_pressureRange
6. Send a pressure ramp profile and read value	fgt_set_pressure fgt_get_pressure
7. Close Fluigent SDK session	fgt_close

6.3 Basic Sensor Regulation

This example shows how to set a sensor regulation and generate a sinusoidal profile on the first sensor and pressure module of the chain
Required hardware: -at least one Fluigent pressure controller (MFCS, MFCS-EZ or FlowEZ) and at least one Fluigent sensor (flow-unit connected to FRP or FlowEZ)

Pseudo-code	Wrapper call
1. Initialize session	fgt_init
2. Get first initialized sensor range for future command	fgt_get_sensorRange
3. Read sensor value	fgt_get_sensorValue
4. Start sensor regulation using first initialized pressure controller. Setpoint is to to 10% of sensor range.	fgt_set_sensorRegulation
5. Wait 5 seconds letting regulation to reach setpoint	
6. Read and display sensor value	fgt_get_sensorValue
7. Regulate flow-rate in form of a sinusoidal wave.	fgt_set_sensorRegulation
Loop every 1 second and read sensor value	fgt_get_sensorValue
8. Send a pressure command stopping running regulation	fgt_set_pressure
7. Close Fluigent SDK session	

6.4 Basic Set Valve Position

This example shows how to change the position of a valve.

Hardware setup: at least one Fluigent valve (M-Switch, L-Switch, 2-Switch or P-Switch)

Pseudo-code	Wrapper call
1. Initialize session	fgt_init
2. Get the number of valves	Fgt_get_valveChannelCount
3. For each valve:	
4. Get the maximum reachable position	fgt_get_valveRange
5. Set the valve to each position	fgt_set_valvePosition
6. Read back the valve position	fgt_get_valvePosition
7. Close Fluigent SDK session	fgt_close

6.5 Basic Get Instruments Info

The example shows how to retrieve information about Fluigent instruments: type, controller, serial number and unique ID.
Aim is to retrieve total number of channels and controllers then get their detailed information. This also shows how to use structures such as *fgt_CHANNEL_INFO*.

6.6 Advanced Specific Multiple Instruments

The example shows how to use specific channels ID and multiple connected instruments.
fgt_initEx function can be used in order to initialize specific instruments in a defined order.
Unique ID is used to address specific pressure channels. Both index (stating at 0) and unique ID can be used as function parameter.

6.7 Advanced Parallel Pressure Control

The example shows how to send concurrent pressure orders using threads. Dll handle parallel calls, functions can be called simultaneous. This demonstrate thread handling, same result is obtained using successive calls as al functions call is executed instantly (within few µs).

6.8 Advanced Features

The example shows advanced features such as limits, units and calibration features.

6.9 Advanced Custom Sensor Regulation

The example shows how to use a custom sensor, different from Fluigent ones and regulate pressure in order to reach setpoint. Different sensor type and range can be used (e.g. liquid pressure, water level, l/min flow meter...) however we do not guarantee full compatibility with all sensors.
For the demonstration a Fluigent flow-unit is used for more simplicity.

7 | Simulated Instruments

The Fluigent SDK allows you to create simulated instruments using the function fgt_create_simulated_instr. This function takes as input an array of integers describing the configuration of the instrument to be created. This chapter explains the required format for the configuration array for each type of instrument.

7.0.1 Pressure controller range

When creating simulated pressure channels, use the following values in the configuration according to the desired range. Not all ranges are supported by all instrument types.

Value	Range
0	None
1	25 mbar
2	69 mbar
3	345 mbar
4	1000 mbar
5	2000 mbar
6	7000 mbar
7	-25 mbar
8	-69 mbar
9	-345 mbar
10	-800 mbar
11	±1000 mbar

7.1 MFCS and MFCS-EZ

The array must contain 8 values, to indicate the pressure range for each port (1-8). See the *Pressure controller range* table at the beginning of this chapter for the available ranges and their codes. Only positive pressure ranges are supported for the MFCS controller.

Example:

```
int[8] config = {5, 5, 4, 3, 0, 0, 0, 0};
```

The configuration above will simulate an MFCS with range 2000 mbar for channels 1 and 2, 1000 mbar for channel 3, and 345 mbar for channel 4. The remaining channels will be empty.

7.2 FRP

The array must contain 8 values, to indicate the type of sensor in each port (1-8). See the *fgt_SENSOR_TYPE* enum for the possible values. Only flow rate sensors are allowed. A port can be left empty by setting its value to 0 (None).

Example:

```
int[8] config = {5, 0, 1, 8, 0, 0, 0, 3};
```

The configuration above will simulate a Flowboard with a Flow Unit M dual calibration on port 1, a Flow Unit XS single calibration on port 3, a Flow Unit XL single calibration on port 4, and a Flow Unit S dual calibration on port 8. The remaining ports will be empty.

7.3 LineUP

The array must contain 10 values for each LineUP module to be created. The 10-position slice for each module must contain the module type in position 0, the serial number in position 1, and the firmware version in position 2. The firmware version can usually be set to 0. The SDK will replace it with the default value for each instrument.

7.3.1 Flow EZ

The module type for the Flow EZ is 1. Positions 3 and 4 in the module's 10-position slice of the array must indicate, respectively, the pressure range and the type of flow rate sensor. See the *Pressure controller range* table at the beginning of this chapter for the available pressure ranges and their codes. The pressure range cannot be None. See the *fgt_SENSOR_TYPE* enum for the available sensor types. Only flow rate sensor types are allowed. The flow rate sensor can be omitted by setting the value to 0 (None).

7.3.2 P-Switch

The module type for the P-Switch is 3. No further configuration is needed, so the remaining values in the 10-position slice of the array can be set to 0.

7.3.3 Switch EZ

The module type for the Switch EZ is 4. Positions 3 to 8 in the module's 10-position slice of the array indicate, respectively the type of valve to be connected ports 1 to 6 of the module. See the *fgt_VALVE_TYPE* enum for the valve types and their codes. The switch type must be compatible with the Switch EZ instrument (M-Switch, L-Switch, or 2-Switch). A port can be left empty by setting its value to 0 (None).

Example:

```
int[30] config = {1, 100, 0, 3, 7, 0, 0, 0, 0, 0,
                  3, 101, 0, 0, 0, 0, 0, 0, 0, 0,
                  4, 102, 0, 2, 0, 0, 0, 0, 1, 0};
```

The configuration above will simulate a LineUP chain with the following 3 modules:

- Flow EZ with serial number 100, default version number, pressure range 345 mbar and Flow Unit L dual calibration.
- P-Switch with serial number 101 and default version number.
- Switch EZ with serial number 102, default version number, a 2-switch valve on port 1 and an M-Switch valve on port 6. The other ports are empty.

7.4 IPS

The array must contain 1 value, which indicates the pressure sensor range to use. See the *fgt_SENSOR_TYPE* enum for the available sensor types. Only pressure sensor ranges are allowed. The sensor type value cannot be None.

Example:

```
int[1] config = {9};
```

The configuration above will simulate an IPS instrument with range S (345 mbar).

7.5 ESS

The array must contain 12 values: 4 to indicate the type of valve in each rotating valve port (A-D), and 8 to indicate the type of valve in each two-switch port (1-8). See the *fgt_VALVE_TYPE* enum for the valve types and their codes. The switch type must match the port it is assigned to (M-Switch or L-Switch for ports A-D, 2-Switch for ports 1-8). A port can be left empty by setting its value to 0 (None).

Example:

```
int[12] config = {1, 0, 0, 3, 2, 0, 2, 0, 0, 0, 0, 0};
```

The configuration above will simulate a Switchboard with an M-Switch on port A, an L-Switch on port D, and 2-Switches on ports 1 and 3. The remaining ports will be empty.

7.6 FOEM

The array must contain 10 values for each FOEM module to be created. The 10-position slice for each module must contain the module type in position 0, the serial number in position 1, and the firmware version in position 2. The firmware version can usually be set to 0. The SDK will replace it with the default value for each instrument.

7.6.1 Pressure Module

The module type for the Pressure Module is 1. Positions 3 and 4 in the module's 10-position slice of the array must indicate, respectively, the pressure range and the type of flow rate sensor. See the *Pressure controller range* table at the beginning of this chapter for the available pressure ranges and their codes. The pressure range cannot be None. See the *fgt_SENSOR_TYPE* enum for the available sensor types. Only flow rate sensor types are allowed. The flow rate sensor can be omitted by setting the value to 0 (None).

7.6.2 Switch Module

The module type for the Switch Module is 4. Positions 3 to 8 in the module's 10-position slice of the array indicate, respectively, the type of valve to be connected ports 1 to 6 of the module. See the *fgt_VALVE_TYPE* enum for the valve types and their codes. The switch type must be compatible with the FOEM Switch Module instrument (M-X, L-X, or 2-X). A port can be left empty by setting its value to 0 (None).

Example:

```
int[30] config = {1, 100, 0, 3, 7, 0, 0, 0, 0, 0,
                  1, 101, 0, 11, 5, 0, 0, 0, 0, 0,
                  4, 102, 0, 2, 0, 0, 0, 0, 1, 0};
```

The configuration above will simulate an FOEM with the following 3 modules:

- Pressure Module with serial number 100, default version number, pressure range 345 mbar and Flow Unit L dual calibration.
- Push Pull Module with serial number 101, default version number, pressure range ±1000 mbar and Flow Unit M dual calibration.
- Switch Module with serial number 102, default version number, a 2-X valve on port 1 and an M-X valve on port 6. The other ports are empty.

7.6.3 Electrovalve Module

The module type for the Electrovalve Module is 3. No further configuration is needed, so the remaining values in the 10-position slice of the array can be set to 0.

7.7 NIFS

The array should be empty. The NIFS device is only available in one range (10000 µL/min), so no configuration is needed.

FLUIGENT

O'kabé bureaux

55-77, avenue de Fontainebleau 94270

Le Kremlin-Bicêtre

FRANCE

Phone: +331 77 01 82 68

Fax: +331 77 01 82 70

www.fluigent.com

Technical support:

support@fluigent.com

Phone : +331 77 01 82 65

General information:

contact@fluigent.com