FLUIGENT

SMART MICROFLUIDICS™

Fluigent Software Developement Kit

User Manual

Version 20.0.0.0 12/2020

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 Software

OS	Bitness	Comment
Windows 7 Windows 8 Windows 10	32 and 64	In order to use the development kit, minimum hardware requirements are: 512 Mo of RAM, Intel Pentium 1.6 GHz processor or equivalent

2.2 Hardware

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™
- Flowboard: XS, S, M, L and XL flow-units
- Inline Pressure Sensor

3 | SDK general philosophy

Fluigent SDK regroups all Fluigent pressure controllers and sensors in one library. All instruments are divided into two main topics:

- channels: pressure controllers, sensors and TTL ports
- controllers: master controller connected to the computer (Link, MFCS, Flowboard)

As all instruments are grouped together they are sorted by their type.

Default order is MFCS Series, FRP LineUP, and finally IPS. All detected MFCS are added first, then all MFCS-EZ, then Flowboards followed by all found LineUP instruments, and finally IPS modules. If multiple instruments of same type are connected, they are sorted by their serial number starting with lowest value first.

Following image shows an example of channels and controllers sorted by default:

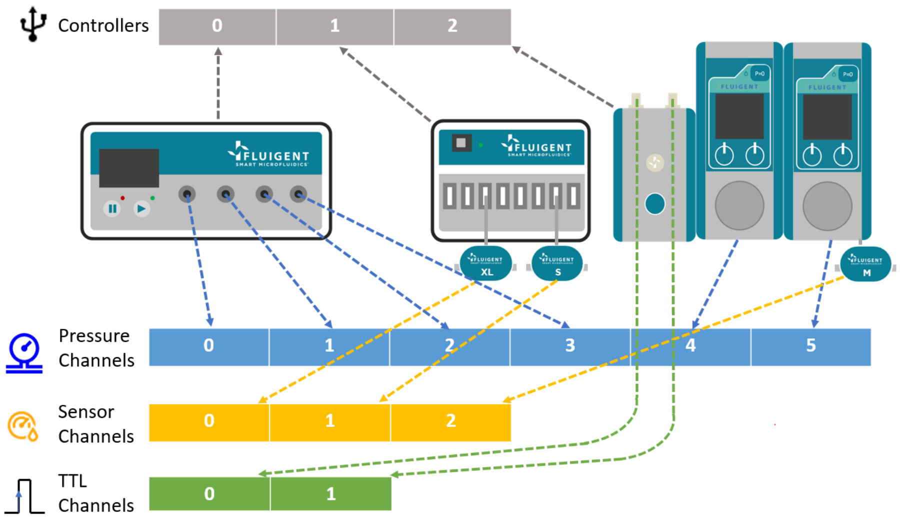

Figure 3.1: Default controllers and channel sorting

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 indexation if not specified by dedicated function *fgt_initEx*. By using this functionality, instruments can be initialized in desired order. Controllers serial numbers has to be entered explicitly.
For example if in 3.1 MFCS-EZ serial number is 567, Flowboard 800 and Link 11023 and you want LineUP instruments to be first on the channel indexing (pressure and flowrate), call *fgt_initEx* function with [11023, 567, 800] array parameters. Setting pressure on first FlowEZ will then use index 0 (instead of 4 by default).

3.1 Channels

Fluigent instrument channels are of three sorts:

- pressure controller (from MFCS™ Series or LineUP Series)
- sensor (flow-units from Flowboard or LineUP Series, IPS module)
- TTL 0-5V input/output ports (from Link, LineUP Series)

Each channel type has it's own indexing starting from 0 and an unique ID. Each channel type has dedicated functions which are labeled *pressure*, *sensor* or *Ttl* because of unique properties.

3.1.1 Unique ID

Unique identification is an unique number which can be used to identify a specific channel. This can be useful when set-up changes avoiding new indexing. However this unique ID is not versatile, if an instrument is replaced, software code has to be adapted. Following image shows how unique ID is computed.

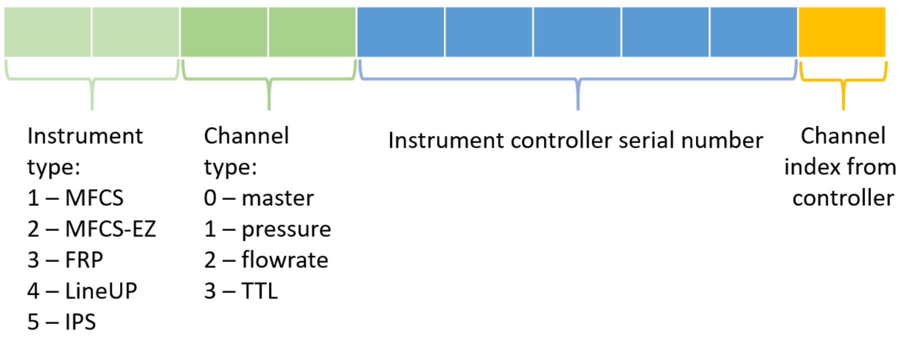

Figure 3.2: Unique channel ID format

Here is an example of a MFCS-EZ SN 567, first pressure channel ID:

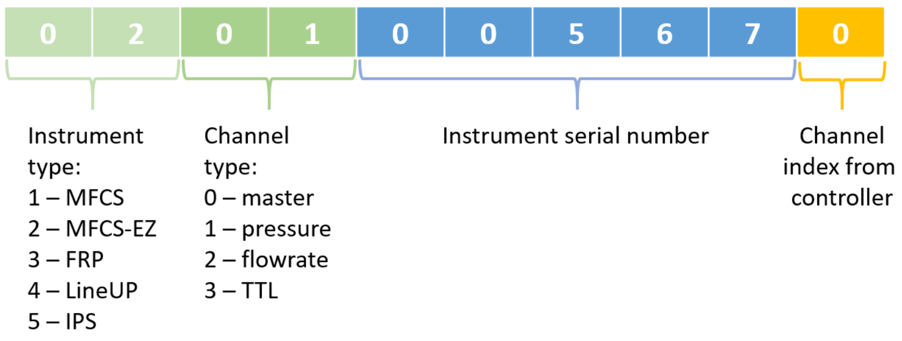

Figure 3.3: Example of MFCS-EZ pressure channel ID

Unique ID value can be computed by hand or retried using channel information dedicated function.

3.1.2 Channel information

Detailed information about each channel (pressure, sensor and TTL) can be retrieved. It concerns controller serial number, device serial number, firmware, position (index from controller), unique ID and instrument type. Sensor has an additional field: it's type.

There are three dedicated functions for each channel type: fgt_get_pressureChannelsInfo, fgt_get_sensorChannelsInfo and fgt_get_TtlChannelsInfo. Returned parameter is a structure of elements.

Some channels do not have a dedicated serial number or firmware, in this case returned value is 0.

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 of three sorts:

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

Each controller has it's own indexing starting from 0 and an unique ID.

Instruments are initialized when calling fgt_init or fgt_initEx function. However directly calling any function automatically initialize the session. Channels order is linked to controllers initialization order. By default instruments are sorted by type then by their type then by their serial number (lower number comes first).

Instruments are by default initialized in following order: all MFCS™ Series, all Flowboards and finally all LineUP. Initialization order also defines channel indexing 3.1. If a specific order of instruments is wanted, fgt_initEx function can be used. Controllers serial number order has to be informed.

As for channels, controllers also have an unique ID and detailed information about their serial number, firmware and type can be retrieved. Unique ID allow control of a specific instrument, regardless it's indexing.

Before exiting your application, Fluigent SDK session has to be closed. This free used memory and closes all running instances (such as regulation). Call fgt_close function before exiting; if not called a fatal error exception is thrown.

3.3 Regulation

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

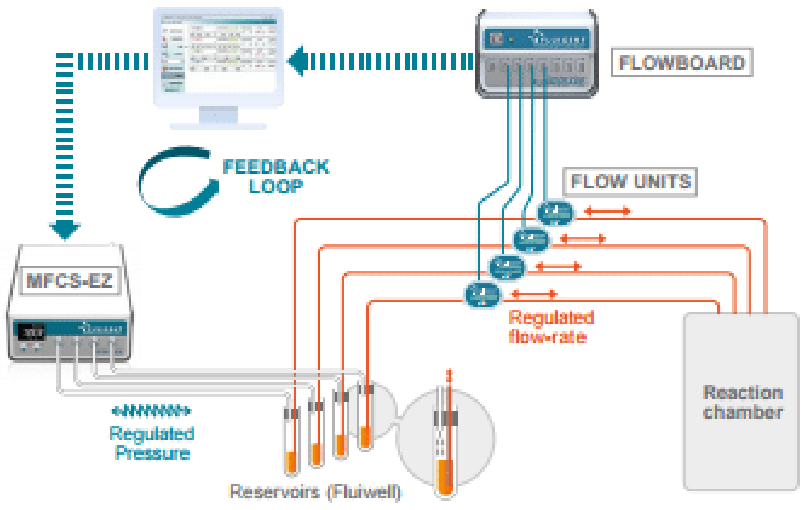

Figure 3.4: Sensor regulation feedback loop

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. Note that regulation using Inline Pressure Sensor is not supported.

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

Fluigent SDK is based on a dynamically linked library (dll). This library contains raw functions. Directly calling the dll is desirable for advanced users and custom applications.

Additionally a more friendly middleware is proposed for five major programming languages: C++, C#, Python, LabVIEW and MATLAB.

When directly working with provided fgt_SDK_*.dll detailed help and function description is provided in fgt_SDK.h header file.

We highly recommend using provided middleware which is open source and can later be adapted to your needs. This document function description concerns the middleware.

4.1 Fluigent SDK dynamic library (dll)

Fluigent SDK, is based upon the following files:

- fgt_SDK_32.dll
- fgt_SDK_64.dll
- fgt_SDK.h

Additionally, as Fluigent SDK interacts with Fluigent devices following dependencies are required:

- mfcs_c_32.dll
- mfcs_c_64.dll
- frp_c_32.dll
- frp_c_64.dll
- LineUP_c_32.dll
- LineUP_c_64.dll
- ips_c_32.dll
- ips_c_64.dll

Note that depending on your development environment 32 or 64 bit, only required files can be kept.

The calling convention is C/C++ __stdcall which means that the parameters are passed by a function onto the stack in the same order as they appear in the function declaration. You can use any language in order to access library functions.

The SDK DLL is in charge to handle the communication with the instrument(s) – MFCS™, FRP™, LineUP™ and IPS. Calling a function of the DLL is a non-blocking operation; it takes only a few 100 µs. Taking data exchange rate into account, it is safe to assume that:

- When calling a “read” function – e.g. read pressure operation – the measurement returned is not older than 100ms for the MFCS™ and FRP™, 20ms for LineUP™, 10ms for IPS;
- When calling a “write” function – e.g. write pressure operation – the instrument gets notified of the new order in less than 100ms for the MFCS™ and FRP™, 20ms for LineUP™, 10ms for IPS.

Calling the DLL in a loop running at a faster rates is ineffective.

4.2 Dll wrapper

The SDK’s wrapper is an abstraction layer allowing the developer to manipulate high level concepts rather than low level function exposed by the provided dll.

Few major programming environments are supported:

Language	**Package**
C++	fgt_SDK_Cpp.cpp middleware file fgt_SDK_Cpp.sln Visual Studio complete solution 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

Middleware layer main aim is to :

- call the fgt_SDK_*.dll file
- adapt variable type for a more friendly use
- display pretty error status if occurs

Each programming language has it’s own specificities however SDK Wrapper 5.2 tries at best to be the same for each environment.

Following sections describe installation steps and specificities for provided middleware.

4.3 Windows installation

Fluigent SDK can be installed from Fluigent Software Platform wizard. Files are copied to *C:\Program Files (x86)\Fluigent\SDK* folder as read only. Please copy or install packages from this location.

Latest version is available on our open source git hub platform: `https://github.com/Fluigent/fgt-SDK`. Please feel free to give your feedback and request specific informations.

4.3.1 C++

Windows C++ Visual Studio solution comes with a middleware file fgt_SDK_Cpp.cpp and several examples showing from basic to advanced features.

Solution was developed and tested on Visual Studio 2019.

There is no need of installation, directly opening the solution and adapting it to your platform allow quick test in a console terminal.

Language specificities:

- each middleware function name starts with a capital f letter (e.g. Fgt_init()) because header file fgt_SDK.h is included in project
- enum types are displayed as string by overriding « std::ostream operator
- three functions manage returned code (sucha as Fgt_Manage_Pressure_Status), change them for custom error handling
- Dll files are copied in project folder using a post-build command

4.3.2 C#

C# wrapper comes with a Visual Studio solution (fgt_sdk_csharp.sln) containing:

- a .NET standard middleware fgt_SDK_Cpp.csproj project file
- several .NET Core examples, in the form of projects, demonstrating from basic to advanced features

Solution was developed and tested on Visual Studio 2019. No fancy installation is required – only .NET core and .NET standard; directly building the solution and adapting it to your platform allow quick test in a console terminal.

Language specificities:

- 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

To install the Python package on Windows, use the MSI or EXE installer according to your version of Windows (and not to your version of Python). If you have more than one version of Python, the installer will ask which one you wish to install the package for. You might have to insert the path to your Python installation manually if it is not detected automatically.

You can also use the easy_install script and is useful if you wish to install the package in a virtual environment (e.g., if you use the Anaconda distribution), but the Python directory must be included in the Windows path for it to work.

```
python -m pip install -user fluigent_sdk-20.0.0.zip
python -m easy_install -user fluigent_sdk-20.0.0.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 use the package on a single project without 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.

Language specificities:

- The functions are located in the Fluigent.SDK module, as shown in the examples. They have the same name as the corresponding DLL functions.
- 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.
- 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

In order to install the toolbox: start MATLAB with Administrator privileges. Go the root of the Toolbox location. Run the toolbox installer “Fluigent SDK.mltbx”. Fluigent toolbox is compatible with MATLAB R2015a and higher. Please contact us if you are using previous MATLAB versions.

After the Toolbox installation is complete, all functions are available as any other MATLAB ones. The Toolbox documentation will be visible from the help and doc functions or the Help Documentation (F1). You can find examples in the documentation. Type `doc Fluigent` to open the list of toolbox functions and enumerations.

Language specificities:

- 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.

4.4 Linux

UNIX based platforms is not yet supported.

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

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

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	Inline Pressure Sensor type

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_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

8. 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 once. If called again, session is reinitialized.

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_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.

5.2.10 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.

5.2.11 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.

5.2.12 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.

5.2.13 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.

5.2.14 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.

5.2.15 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. Not supported by the IPS

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.

5.2.16 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.

5.2.17 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.

Unit, calibration and limits

5.2.18 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.

5.2.19 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.

5.2.20 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.

5.2.21 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.

5.2.22 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.

5.2.23 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.

5.2.24 fgt_get_sensorCalibration

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

Get internal calibration table used by the sensor. Not supported by IPS.

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.

5.2.25 fgt_set_sensorCustomScale

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

Apply a custom scale factor on sensor read value. This function is useful in order to adapt read sensor value to physical measurement.
For example if a flow-unit is used with a special oil and it's calibration table is set to H2O, read flowrate is not correct. Scale factor is applied using following formula:

$$scaled\_value = a * sensor\_value + b * sensor\_value^2 + c * sensor\_value^3 \tag{5.1}$$

.
Note that this scale is also used for the regulation. Not supported by IPS.

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.

5.2.26 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 flowrate sensor measurement. This function is useful in order to adapt read sensor value to physical measurement. For example if a flow-unit is used with a special oil and it's calibration table is set to H2O, read flowrate is not correct. Scale factor is applied using following formula:

$$scaled\_value = a * sensor\_value + b * sensor\_value^2 + c * sensor\_value^3 \tag{5.2}$$

When applying a custom scale factor, sensor range may increase very rapidly, SMax parameter is meant to limit this maximal value. This function purpose is to be used with the regulation in order to avoid too high maximum range on the sensor.

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.

5.2.27 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.

5.2.28 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. Not supported by IPS.
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.

5.2.29 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.

5.2.30 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.

5.2.31 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

5.2.32 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. Not supported by IPS.

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.

5.2.33 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

5.2.34 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.

5.2.35 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.

5.2.36 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.

5.2.37 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

5.2.38 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.

5.2.39 fgt_read_Ttl

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

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

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.

5.2.40 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

5.2.41 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.

5.2.42 fgt_set_manual

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

Manually activate internal electrovalve. This stops pressure regulation.
This feature is only available on MFCS and MFCS-EZ devices.

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.

5.3 Types equivalency

Provided library was developed in C/C++ language and uses “__stdcall” calling convention. Following table exposes variable equivalencies in different languages, allowing function cast (number of bytes have to be identical):

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_uchar	U8	Uint8
	pointer to unsigned char	byte by ref	byref(c_uchar)	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.	fgt_set_sensorRegulation
Setpoint is to to 10% of sensor range.	
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 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*.

6.5 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.

6.6 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).

6.7 Advanced Features

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

6.8 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.

FLUIGENT
SMART MICROFLUIDICS™

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