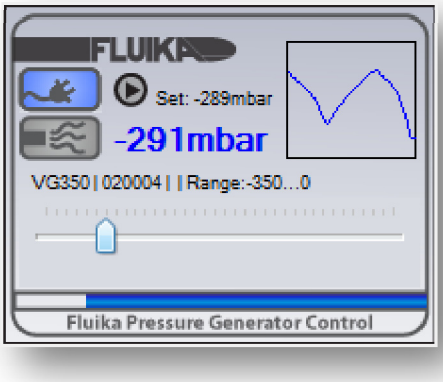
FLUIKA™ Miniature Pneumatic Control Kit

.NET library for FLUIKA pressure and vacuum generators

Pressure Generator (PG)
Vacuum Generator (VG)
Vacuum-Pressure Generator (VPG)

Document: FD005
Version: 1
Date: 2013-02-10



Disclaimer

We (“Fluika OÜ”) believe that our products are safe, while used in the intended manner and under normal conditions. However it is entirely your (“Buyer’s”) responsibility to ensure safety of your application, setup or eventual system, where our products have been used as components. As well as it is your responsibility to ensure, that your eventual system meets with your specification. Our products are not intended for critical applications, where failure of the device may result in hazard to life or compromise any other ways safety of person or property (Life support and safety applications). Any unintended use is entirely at the risk of buyer, where we have no liability. We disclaim all liability arising from this information and its use.

IMPORTANT

Due to constant development of our devices, upgraded firm- and software, your actual device or software behavior may differ from that in this documentation. Therefore it is important to obtain latest documentation from our web site (www.fluika.com). All documents are indexed and named FD followed by three digit document number and eventually version number (eg. “FD001-v1.pdf”). Also source codes, project examples and tutorial videos can be found on our web site.

CONTENT

This document contains detailed information how to use .NET library to control FLUIKA™ Pressure / Vacuum / Vacuum-Pressure Generators (“devices”). For General information and guidance, please read first General User’s Manual of FLUIKA™ Kit (Document: FD001) and hardware manual of pressure and vacuum generators (Document: FD003). Present document describes:

- User interface
- .NET programming
- An example

USER INTERFACE

For easy and convenient development of your custom control applications, Fluika .NET library provides graphical user interface components, which can be quickly integrated into your program.

For use in your .NET project, place Fluika library containing FluikaPG.dll, FluikaDeviceManager.dll and folder FDM_Settings into your development folder (and eventual folder of your application). Add component FluikaPG into your toolbox (Toolbox --> Right mouse click --> "Choose Items ..." from drop down menu -->.NET Framework components -->Browse --> Choose FluikaPG.dll file)

Now you can place a graphical control element (type: **FluikaPG.FluikaPGControl**) into your software (Figure 1). Use as many control elements as many different pressure controllers are needed in the project. The actual software design can be slightly different weather you use one or multiple controllers at the same time. See later examples.

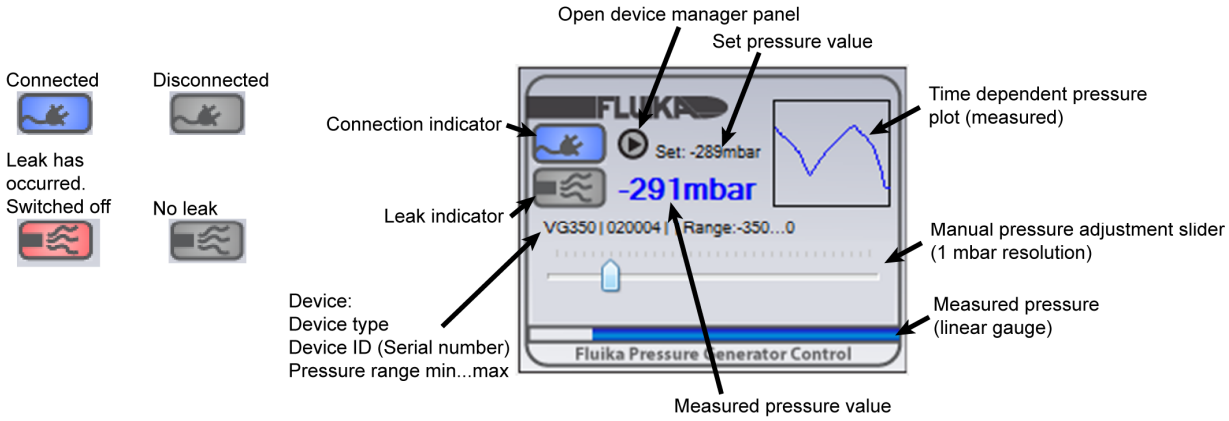


Figure 1 | Graphical user element component (size: 215 x 169 pix) containing everything for manual connection and control of the device.

PROGRAMMING

Syntax here is given for .NET C++, but component can be equally used with other languages such as C# or Visual Basic. See example projects! Components and examples have been developed with Visual Studio 2008 and are working on both 32- and 64-bit Windows computers.

Properties

`int setPressure` Sets active pressure in mbar-s. Resolution is 1 mbar. Vacuum values are with minus sign.
`String^ searchID` Is 6 character long string of device ID (Serial number), which this component shall connect. * can be used as a wildcard to replace characters, which shall be not considered in search.
`bool enableAutoConnectByID` If automated searching of devices with ID specified in property *searchID* allowed.

Methods

`void activate()` Activates FLUIKA device system. This function shall be called first before others. It initializes the component.
`int getPressure()` Returns latest measured pressure value in mbar. Resolution is 1 mbar. Vacuum values are with minus sign.
`String^ getLastError()` Returns code and message of the last error. Error codes are • `"[0] No Error"` If there is no error • `"[1] Data communication error"` If data communication error has occurred • `"[3] Sensor error"` If sensor error has occurred (device restart is required)

`array<array<String^>^>^ getDeviceList()`
Returns the latest list of connected devices. This list is a string array with size 256 x 5. Each of 245 lines represents on COM port counted from 1 to 256. Each line has 5 String values:
• First is by port number (1 to 256) as a string.
• Second port status, which can have4 different states:
  o `"OFFLINE"` No device connected to this port.
  o `"BUSY"` Port could not be opened. Either hardware problem or port is already opened by same or another program.
  o `"NOREPLY"` Device do not reply as fluika device. Either connected device is not fluika device, or hardware failure has occurred.
  o `"FLUIKA"` Device is recognized as FLUIKA device.
• Third is deviceID (SN: 6-digits), if it is fluika device, otherwise empty string
• Fourth is device type (eg. "PG500")
• Fifth is device alias name (*this is becoming obsolete!*)

`int getResponseTime()` Returns the latest response time. It is updated before the ResponseTimeReceived event is called. If pressure is set either by the function setPressure or by slider than time is counted between setting the value and until it is reached. Unit is ms.
`int previouosSetPressure()` Returns the previous set pressure value. Not the current, but one before it.
`void calibrationSettings(array<int>^ value)` Loads a set of calibration constants into device. This feature is currently under development. But will allow in future tuning device performance without chaning the firmware.
`void setLeakTime(int value)` Sets leak time. This is maximum continous allowed pumping time before pumps shuts off. Time is given in seconds. Minimum 1, maximum 100 seconds. Default is 10s. This is to protect pressure generator from continuous empty run, which may shorten its lifetime.
`void deviceOff()` Switches off all pneumatic components in device. It minimizes power consumption and pressure Returns to atomospheric pressure, 0.
`void pumpingOff()` Switches off pumping, but does not release valves. Pressure returns slowly to 0 due to leaks.
`int getMinPressure()` Gets minimum pressure value of currently connected device

`int getMaxPressure()` Gets maximum pressure value of currently connected device
`String^ deviceID()` Gets device ID (Serial number) of currently connected device
`String^ deviceType()` Gets device type of currently connected device. Eg. "PG500" or "VG350"
`String^ devicePort()` Gets COM port of currently connected device
`bool isConnected()` Is device connected currently
`void updateDeviceList()` Gets device ID (Serial number) of currently connected device
`void connectDevice(int portNr)` Connect to a device with following COM port number
`void disconnectDevice()` Disconnect device

Events

`BecameConnected` This event is called when device became connected
`BecameDisconnected` This event is called when device became disconnected, either by programmed disconnection or by phyiscal loss of connection (eg. cable was pulled out)
`PressureChanged` This event is called when set pressure value has been changed, either through code or user interface
`PressureReadingUpdated` This event is called when read pressure value has been updated
`ErrorOccured` This event is called when error occured
`LeakOccured` This event is called when leak occured. When leak occurs, pump is switching off, until pressure value is set again
`UpdatedDeviceList` This event is called when device list became updated
`ResponseTimeReceived` This event is called when set pressure value has been reached and response time has been determined

EXAMPLE

Following describes a simple example with one pressure controller. Place control component (fluikaPGControl1) on your form (Form1). Take form Load event and insert following code.

```
private: System::Void Form1_Load(System::Object^  sender,
            System::EventArgs^  e)
{
            fluikaPGControl1->activate(); //Activate component
            fluikaPGControl1->searchID="******"; //Connect to any device
            fluikaPGControl1->enableAutoConnectByID=true;//Allow autoconnect
}
```

This program would automatically connect to any pressure controller, which is connected to computer. You can try to connect the device physically to USB port and then disconnect and connect again. This program shall automatically detect the presence of the device and connect to it.

If you have more than one Fluika device in your project, then you need to use **FluikaInstrument** component to manage connectivity of all Fluika devices properly. Please see manual. Otherwise there is risk that multiple components will access same ports simultaneously, which can lead to malfunction.

See also source codes of sample projects!