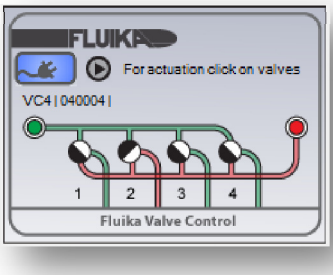
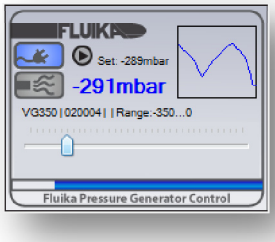
FLUIKA™ Miniature Pneumatic Control Kit

.NET library FluikaInstrument

Document: FD007
Version: 1
Date: 2013-02-12





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 build a program to control setup or instrument, which contains multiple FLUIKA™ units. For General information and guidance, please read first General User’s Manual of FLUIKA™ Kit and individual manuals of hard- and software parts. Present document describes two modes:

- Introduction
- .NET programming
- Examples

INTRODUCTION

In order to handle multiple Fluika units in same design, **FluikaInstrument** library shall be used. This provides a framework, which takes care connecting all devices and ensure that multiple independent components would not crash with each other. FluikaInstrument contains two classes `FInstrumentA` and `FInstrumentB`. If you would like to use individual control modules on your user interface and access their all possible functions, then **A** class shall be used. B class, instead, is optimized for shortest code, where the class contains all device handling, but it allows only essential functions and would not allow custom placing of control components on your GUI.

PROGRAMMING

FInstrumentA

Methods

`void loadInstrumentDefinition(array<String^>^ inputL)`
Loads the instrument definitions table, which tells, which software component (identified by a name) is connected with which Fluika unit (identified by ID/Serial number). Syntax of the strings in the table is following `NAME(ID)` where NAME is user defined name for the component (used later in software) and ID is 6-digit ID/Serial number from the physical unit. Each line in this table can contain only one such a definition. Line can be empty and can contain also comments denoted by C++ style //.
Example array could be:
`MYVALVE(040005)`
`MYP(010010)`
`MYV(020007)`
Instrument definition shall be loaded before actual components are connected!

`void loadInstrumentDefinitionFile(String^ fileName)` Loads the instrument definitions table from text file, which has same syntax as in previous function.
`void connectComponent(String^ name, Object^ comp)` Connect a component to instrument. `name` denotes the component name as described above. This name has to match the name loaded (case-sensitive). If name hasn't been loaded before an exception will be thrown. `comp` is pointer to a Fluika pressure or valve control (Type eg. `FluikaPGControl` or `FluikaVCControl`)
`void activate()` Activates the FluikaInstrument. Thereafter it tries to find and connect with all defined units.
`bool isConnected()` Checks if all units in the instrument are connected.

Events

`BecameConnected` This event is called when entire instrument (all units) became connected
`BecameDisconnected` This event is called when instrument becomes to be not fully or partly connected

EXAMPLE

Following describes a simple example with one pressure controller (`fluikaPGControl1`) and one valve controller (`fluikaVCControl1`). Place control component (`fInstrumentA1`) on your form (Form1). Take form Load event and insert following code.

```
private: System::Void Form1_Load(System::Object^  sender,
            System::EventArgs^  e)
{
            array<String^>^ instDef=gcnew array<String^>(2);
            instDef[1]="MYVALVE(040005)";
            instDef[2]="MYP(010010)";
            fInstrumentA1->loadInstrumentDefinition(instDef);
            fInstrumentA1->connectComponent("MYVALVE",fluikaVCControl1);
            fInstrumentA1->connectComponent("MYP",fluikaPGControl1);
            fInstrumentA1->activate();
}
//This is full initialization of two components. Later connection state can
be probed with isConnected() function
```

This program would automatically connect both units!

See also source codes of sample projects!

FInstrumentB

Properties

`bool show` Visualize separate control panel where each unit has its own component.

Methods

`void loadInstrumentDefinition(array<String^>^ inputL)` Loads the instrument definitions table similarly with A class, but syntax is slightly different containing also type of component "valves" or "pressure", like: `pressure(NAME,ID)` and `valves(NAME,ID)` . Other aspects of syntax are same as in previous class. After loading activation happens automatically.
`void loadInstrumentDefinitionFile(String^ fileName)` Loads the instrument definitions table from text file, which has same syntax as in previous function.
`bool isConnected()` Checks if all units in the instrument are connected.
`void setPressure(String^ devName, int value)` Sets the pressure in unit which has defined name given by input `devName`. Pressure value is given in mbar-s.
`int getPressure(String^ devName)` Gets the pressure in unit which has defined name given by input `devName`. Pressure value is given in mbar-s.
`void setValve(String^ devName, int valve, bool state)`

Set the valve in unit, which has defined name given by input `devName`. Variable `valve` is defininig which of the valves is actuated (1..4) and `state` is valve state (true=on, false=off)

EXAMPLE

Following describes a simple example with one pressure controller and one valve controller. Place control component (fluikaVCControl1) on your form (Form1). Take form Load event and insert following code. Then place a button and define click event. Now, once device is connected pressing the button would set pressure to 250 mbar and switch on valve 2.

```
private: System::Void Form1_Load(System::Object^  sender,
            System::EventArgs^  e)
{
            array<String^>^ instDef=gcnew array<String^>(2);
            instDef[1]="valves(MYVALVE,040005)";
            instDef[2]="pressure(MYP,010010)";
            fInstrumentB1->loadInstrumentDefinition(instDef);
            fInstrumentB1->show=true; //Make the control panel visible
}
//This is full initialization of two components. Later connection state can
be probed with isConnected() function
private: System::Void button1_Click(System::Object^  sender,
System::EventArgs^  e) {
            fInstrumentB1->setPressure("MYP",250);
            fInstrumentB1->setValve("MYVALVE",2,true);
                        }
```

This program would automatically connect both units!

See also source codes of sample projects!