Writing a Tango class

T
A
N
G

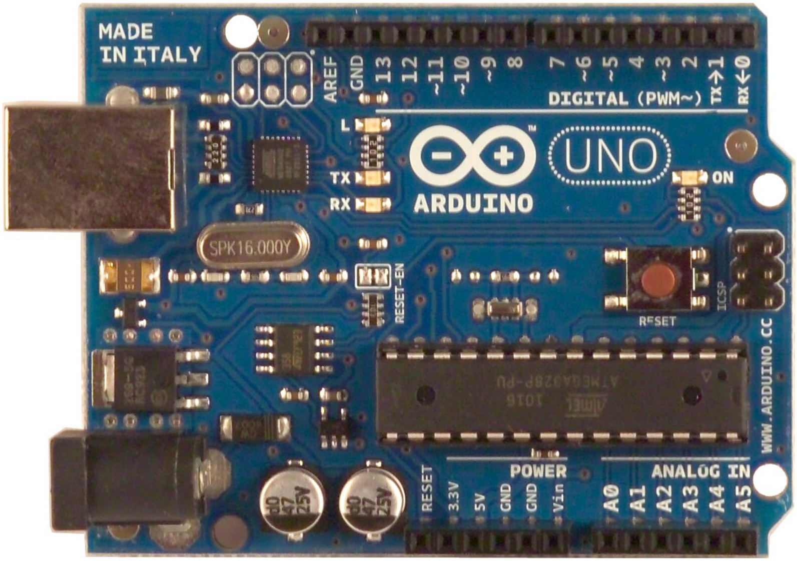
- The hardware (Arduino)
- A basic class using Pogo
- A Python Tango class for the temperature sensor
- The same Tango class in Cpp

The hardware

- A temperature controller
  - Based on Arduino UNO board
    - http://arduino.cc/en/
  - Temperature sensor is a LM 35 chip from NI
  - Arduino UNO board connected to your PC using a USB port

The hardware

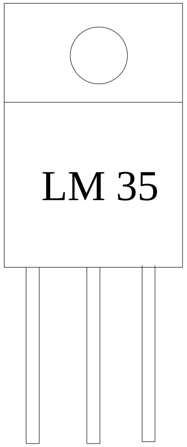
USB



LM 35 Vs pin

LM 35 Vout pin

LM 35 GND pin



Vs

Vout

GND

The hardware

- Arduino UNO
  - Atmega 38 microcontroller 8 bits
    - 16 Mhz – 32 KB flash + 2KB SRAM + 1 KB EEPROM
  - 14 digital I/O (6 PWM)
  - 6 analog inputs
    - 10 bits / 0 – 5 Volts
  - 1 Serial line
    - External or via USB
- LM 35
  - 10 mV / Celcius deg

The hardware

- A serial line is simulated in the USB connection
- A small program running in the Arduino controller chip has been written
  - If the Arduino receives the character '**T**', it returns the temperature (in Celsius degree) coded as a float number in a string (eg: '22.34')
  - 'Protocol error' is returned for all other character

Writing a Tango class

- Chat with the equipment responsible
  - Which device states?
  - Which commands?
  - Which attributes?
- No real rules to decide what has to be implemented using commands or attributes

Writing a Tango class

- Writing Tango device class need some glue code. We are using a code generator with a GUI called **POGO:** **P**rogram **O**bviously used to **G**enerate **O**bject
- Following some simple rules, it's possible to use it during all the device class development cycle (not only for the first generation)
- POGO generates
  - C++, Python Tango device class glue code
  - Makefile (C++)
  - Basic Tango device class documentation (HTML)

Writing a Tango class

- Using Pogo you
  - Give Tango class name
  - Define Tango device state(s)
  - Define Tango device command(s)
  - Define Tango device attribute(s)
  - Define Tango device state machine
  - Define Tango device property(ies)
  - Enter Tango device documentation

A simple Tango class

- Let's generate a SkiLift Tango class
  - 3 states: ON, OFF, FAULT
  - 3 commands

Name	In	Out	Allowed
Reset	Void	Void	If FAULT
On	Void	Void	If OFF
Off	Void	Void	Always

  - 3 attributes

Name	Type	Format	Writable
Speed	double	scalar	read/write
Wind_speed	double	scalar	read
Seats_pos	long	spectrum	read

The temperature sensor Tango class

- Tango class name: **GrenobleTemp**
- 2 states: **OFF, ON** (ALARM)
- 1 attribute:
  - **Temp**
    - Scalar, float, read
    - Label = Temperature, unit = deg
    - Quality factor invalid if state != ON, ALARM
- 2 commands
  - **On** allowed only if OFF state
  - **Off** allowed only in ON state
- 1 device property: **SerialLine** - string
- Python as language
- Start “pogo-6” from a shell window

Device server in database

- By default, POGO generates
  - Device Server name = Tango class name
    - Server name = class name = “GrenobleTemp”
- Start Jive to register device server process in database
- Click “Edit / Create server” to register your device server
- Choose one instance name
- Choose a device name following Tango device name syntax

domain/family/member

Device Server startup sequence

- Connect to DB device using TANGO_HOST env. variable (or /etc/tangorc)
- Send to DB device server executable name, instance name and Tango class
- DB returns device name list
- FOR each device(s)
  - Ask DB for device properties
  - Create device
  - Send device network connection parameters to DB

Client create connection

- Connect to DB device using TANGO_HOST env. variable (or /etc/tangorc)
- Ask DB what are the network connection parameters for device “domain/family/member”
- Create direct connection to the device

Coding a Tango class

- Four things to do
  - Device creation
  - Implementing commands
  - Reading attribute(s)
  - Writing attribute(s)

Coding a Tango class

- Which methods can I use within a Tango class?
  - Your class inherits from a Tango library class named Device_<x>Impl
    - All methods from Device_<x>Impl class (mapped to Python)
  - Some methods received a Attribute or Wattribute object
    - All the methods of these two classes wrapped to Python
- Doc available at http://www.tango-controls.org
  - Document/Tango kernel/PyTango for Python
  - Document/Tango kernel/Tango device server classes for C++

Creating the device

- A init_device() method to construct the device
  - **GrenobleTemp.init_device()**
- A delete_device() to destroy the device
  - **GrenobleTemp.delete_device()**
- All resources acquired in init_device() must be returned in delete_device()

Creating the device

- The init_device() method
  - Init state (and status if required)
  - Init (create) local data

```
#------------------------------------------------------------------
#                    Device initialization
#------------------------------------------------------------------
    def init_device(self):
            print "In ", self.get_name(), "::init_device()"
            self.set_state(PyTango.DevState.OFF)
            self.get_device_properties(self.get_device_class())

            self.ser = serial.Serial(self.SerialLine,9600)
```

Implementing a command

- A hook → always_executed_hook()
  - **GrenobleTemp.always_executed_hook()**
- If state management is needed, one is_xxx_allowed() method
  - **bool GrenobleTemp.is_On_allowed()**
- One method per command
  - **GrenobleTemp.On()**

Implementing a command

- GrenobleTemp.is_On_allowed()

```
    def is_On_allowed(self):
        if self.get_state() in [PyTango.DevState.ON]:
#                 End of Generated Code
#                 Re-Start of Generated Code
            return False
        return True
```

Implementing a command

- GrenobleTemp.On command coding

```
    def On(self):
        print "In ", self.get_name(), "::On()"
#               Add your own code here
        self.set_state(PyTango.DevState.ON)
```

Reading attribute

- A hook → always_executed_hook()
  - **GrenobleTemp.always_executed_hook()**
- One method to read hardware
  - **GrenobleTemp.read_attr_hardware(data)**
- If state management is needed, one is_xxx_allowed() method
  - **bool GrenobleTemp.is_Temp_allowed(req_type)**
- One method per attribute
  - **GrenobleTemp.read_Temp(Attribute)**

Reading attribute

- read_attr_hardware() method

```
#------------------------------------------------------------------
#                Read Attribute Hardware
#------------------------------------------------------------------
   def read_attr_hardware(self,data):
        print "In ", self.get_name(), "::read_attr_hardware()"
```

Reading attribute

- read_Temp() method

```
    def read_Temp(self, attr):
         print "In ", self.get_name(), "::read_Temp()"

#                         Add your own code here

        sta = self.get_state()
        if sta in [PyTango.DevState.ON,PyTango.DevState.OFF]
            self.ser.write('T')
            answer = self.ser.readline()
            stripped_answer = answer.rstrip()

            self.debug_stream("Temperature returned by arduino = ",stripped_answer

            try:
                attr_Temp_read = float(stripped_answer)
                attr.set_value(attr_Temp_read)
            except ValueError:
                PyTango.Except.throw_exception("GrenobleTemp_WrongAnswer",
                "Wrong answer from Arduino. Can't be converted to float","GrenobleTemp.read_Temp")
        else:
            attr.set_quality(PyTango.AttrQuality.ATTR_INVALID)
```

Writing attribute

- A hook → always_executed_hook()
  - **GrenobleTemp.always_executed_hook()**
- If state management is needed, one is_xxx_allowed() method
  - **bool GrenobleTemp.is_Temp_allowed(req_type)**
- One method per attribute
  - **GrenobleTemp.write_xxx(WAttribute)**

Writing attribute

- write_xxx() method

```
def write_xxxx(self, attr):
        print "In ", self.get_name(), "::write_Speed()"
        data = attr.get_write_value()
        .....
```

Tango class in C++

- Use POGO 7 (latest release) to generate C++ Tango class
  - Support Tango class inheritance
  - Support Multi-Tango class device server
  - Based on a DSL (Xtext - Xpand)

GrenobleTemp in C++

- Use the already existing Tango class to control serial line.
  - Available in SourceForge tango-ds project
  - Still in CVS!
    - Module name = SerialLine
  - Doc not available in pink site due to actual migration / merging task between CVS and SVN

GrenobleTemp in C++

- GrenobleTemp uses a SerialLine Tango device to control the serial line
  - Use Pogo “Tools/Multi Classes Manager” to create the device server embedding 2 Tango classes
  - GrenobleTemp device property is now SerialDevice and initialized to the Serial line Tango device name
- GrenobleTemp Tango class is a client of the serial line Tango device.