

## **CMIDAQ DEVELOPMENT: TINE + PYTHON.**

- WHY WE GO FOR PYTHON.**

- TINE CLIENT - GUI IN PYTHON**

IGOR RUBINSKIY, CFEL/UHH/CUI CONTROLLED MOLECULE  
IMAGING GROUP

8.06.2015, DESY, TINE USERS MEETING

# WHAT DO WE NEED ON THE CMI DAQ SIDE 1/2

- Full (remote) control over the Hardware Units:
  - Digitiser (upto 400 MB/s uncompressed)
    - data reduction on CPU/server: pedestal subtraction, peak finder (Time, Signal Integral)
  - Pixel Camera (1 Mpixel, 12 bit, 120 fps - 180 MB/s uncompressed)
    - data reduction on CPU/server: blob finder, centroiding (x,y, amplitude)
  - plenty of "slow" devices: Delay Generator, Translation/Rotation stages, Temperature, Humidity, Vacuum sensors.
  - quick-device-implementation interface
- Easy device selection, configuration, and data storage/access, tree structure:
  - practical need: prepare a list of "step-by-step" instructions = how the group of detectors can do a multi-dimensional scan over two (standard scan) or more parameters - a la "shoot-and-forget"
  - data storage format and mechanism is not critical (preference to HDF5), but we want to move high level analysis tools as close as possible to the beginning of the data acquisition chain - rich graphics (!)

# WHAT DO WE NEED ON THE CMI DAQ SIDE 2/2

- centralised control system over many experiments in the labs
  - safety nets and provisions for device moves without cross-talks between setups, but not being "difficult"
- 1 user friendly GUI for a variety of experiments
  - plenty high-level data analysis including graphical input and output
  - 2D and nD histograms, projections, slices
  - gates, signal expressions (math),
- user defined scans:
  - multi-dimensional scans (iterative acquisition, small increments of single parameter)
  - user extensibility (should be easy for students to add new functional)
  - secondary analysis

# PYTHON:

Interpreted & Object oriented,

High-level built-in data structures

Dynamic typing & dynamic binding

- is acknowledged for Prototyping, Rapid Application Development, scripting, and "glue" language to bring different components together

In general the syntax is simple and easy to learn (good for students!)

The syntax rules force one to write clear and readable code

- reduces the maintenance cost

Python code is built around modules and packages

- encourages program modularity and code re-use

Exception-based error handling

Extensive standard libraries and large variety of third-party modules for virtually every task

Embeddable within applications as a scripting/interactive interface



# Language. Conclusion

Java

Python

Easy to write clear code

Difficult to create unclear code

Clear

Language specification  
JCP  
Static typing  
Other specifications

Formal

Lambda calculus  
A lot of lang features

Flexible

procedure/module

Maintainable

Code Fast

Backward capability  
Static typing  
No cyclic reference problem

Dynamic typing  
A lot of good language features  
A lot of libraries



# Coding. Crawler



## Java

```
public class CrawlerExample {  
  
    public static void main(String[] args) throws IOException {  
        PrintWriter textFile = null;  
        try {  
            textFile = new PrintWriter("result.txt");  
            System.out.println("Enter the URL you wish to crawl..");  
            System.out.print("@> ");  
            String myUrl = new Scanner(System.in).nextLine();  
  
            String response = getContentByUrl(myUrl);  
  
            Matcher matcher = Pattern  
                .compile("href=[\\''](.[^\\'']+)\\'']").matcher(response);  
            while (matcher.find()) {  
                String url = matcher.group(1);  
                System.out.println(url);  
                textFile.println(url);  
            }  
        } finally {  
            if(textFile != null) {  
                textFile.close();  
            }  
        }  
    }  
  
    private static String getContentByUrl(String myUrl)  
        throws IOException {  
        URL url = new URL(myUrl);  
        URLConnection urlConnection = url.openConnection();  
        BufferedReader in = null;  
        StringBuilder response = new StringBuilder();  
        try {  
            in = new BufferedReader(new InputStreamReader  
                (urlConnection.getInputStream()));  
            String inputLine;  
            while ((inputLine = in.readLine()) != null) {  
                response.append(inputLine);  
            }  
        } finally {  
            if(in != null) {  
                in.close();  
            }  
        }  
    }  
}
```

## Python

```
if __name__ == '__main__':  
    with open("result.txt", "wt") as textFile:  
        print("Enter the URL you wish to crawl..")  
        myUrl = input("@> ")  
        for i in re.findall("href=[\\''](.[^\\'']+)\\'\"",  
            urllib.request.urlopen(myUrl).read().decode(), re.I):  
            print(i)  
            textFile.write(i+'\n')
```





# Coding. Functional programming



## Java

```

public class CommandPatternExample {

    public static void main(String[] args) {
        Executor executor = new Executor();
        executor.execute(new Command() {
            @Override
            public void execute(int parameter) {
                System.out.println(parameter + parameter);
            }
        });
    }

    class Executor {
        private int parameter = 100;
        public void execute(Command command) {
            if(command != null) {
                command.execute(parameter);
            }
        }
    }

    interface Command {
        public void execute(int parameter);
    }
}

```

## Python

```

class Executor:
    parameter = 100
    def execute(self, command):
        if command is not None:
            command(self.parameter)

if __name__ == '__main__':
    executor = Executor()
    executor.execute(lambda parameter: print(parameter + parameter))

```

The  
**Lambda**  
 CALCULUS



# Coding. Functional programming



## Java

## Python

```

def win(arr):
    chars = list(set(arr))
    if len(chars) == 1 and chars[0] != '.':
        return chars[0]
    return '.'

def checkio(game_result):
    game_lines = []

    for i in range(3):
        horizontal_line = game_result[i]
        game_lines.append(horizontal_line)

    for i in range(3):
        vertical_line = [game_result[x][i] for x in range(3)]
        game_lines.append(vertical_line)

    diagnoal_right = [game_result[0][0], game_result[1][1],
                      game_result[2][2]]
    game_lines.append(diagnoal_right)

    diagnoal_left = [game_result[2][0], game_result[1][1],
                    game_result[0][2]]
    game_lines.append(diagnoal_left)

    for line in game_lines:
        w = win(line)
        if w != '.': return w

    return 'D'

```

```

checkio=lambda r,j="".join:"DOX"[sum(m*3in[j(r[:e])[:4]for e
in(1,-1)]+r+list(map(j,zip(*r)))for m in"XOX")]

```

**Solutions for "Xs and Os Referee". Lambda hell like RegEx**





# Coding. Functional programming



## Java

## Python

```
for(Iterator<Product> iter = list.iterator(); iter.hasNext();) {  
    Product product = iter.next();  
    if(product.isDeleted()) {  
        iter.remove();  
    }  
}
```

```
double price = 0.0;  
for (Product product : productList) {  
    price += product.getPrice() * product.getQuantity();  
}
```

```
Point[] f = new Point[]{new Point(1,2), new Point(3,4), new Point(5,6)};  
double h = 0.0;  
for (Point p : f) {  
    h += Math.sqrt(p.x*p.x+p.y*p.y);  
}
```

```
filter(lambda product: product.deleted == False, productList)
```

```
reduce((lambda p1, p2: p1 + p2),  
       map(lambda product: product.price*product.qty, productList))
```

```
f = [(1,2), (3,4), (5,6)]  
h = reduce(lambda x,y: x+y, [math.sqrt(x*x+y*y) for x,y in f])
```

Look ma, no loop!





# Parallelism. Multiprocessing



Java

```
final BlockingQueue<SomeObject> queue
    = new LinkedBlockingQueue<>();

new Thread(new Runnable() {
    @Override
    public void run() {
        queue.add(new SO(42, null, "hello"));
    }
}).start();

try {
    System.out.println(queue.take());
} catch (InterruptedException e) {
    throw new RuntimeException(e);
}
```

[42, None, 'hello']

Python

```
def f(q):
    q.put([42, None, 'hello'])

if __name__ == '__main__':
    q = Queue()
    p = Process(target=f, args=(q,))
    p.start()
    print(q.get())
    p.join()
```

[42, None, 'hello']

Output:

# PYQTGRAPH

## PURE PYTHON GUI AND GRAPHICS BASED ON QT AND NUMPY

## SUPPORTED ON: WINDOWS LINUX MACOS

**PyQtGraph**  
Scientific Graphics and GUI Library for Python  
Documentation and API Reference · Developer Forum · Wiki

**pyqtgraph development version**

Source package	pyqtgraph-0.9.10.tar.gz
Debian/Ubuntu package	python-pyqtgraph_0.9.10-1_all.deb
Windows installers	pyqtgraph-0.9.10.win32.exe pyqtgraph-0.9.10.win-amd64.exe

Or fork the code from github:  
<https://github.com/pyqtgraph/pyqtgraph>

recent changes · older releases

PyQtGraph is a pure-python graphics and GUI library built on PyQt4 / PySide and numpy. It is intended for use in mathematics / scientific / engineering applications. Despite being written entirely in python, the library is very fast due to its heavy leverage of numpy for number crunching and Qt's GraphicsView framework for fast display. PyQtGraph is distributed under the MIT open-source license.

**Main Features:**

- **Basic 2D plotting in interactive view boxes**
  - Line and scatter plots
  - Data can be panned/scaled by mouse
  - Fast drawing for realtime data display and interaction
- **Image display with interactive lookup tables and level control**
  - Displays most data types (int or float; any bit depth; RGB, RGBA, or luminance)
  - Functions for slicing multidimensional images at arbitrary angles (great for MRI data)
  - Rapid update for video display or realtime interaction
- **3D graphics system** (requires Python-OpenGL bindings)
  - Volumetric data rendering
  - 3D surface and scatter plots
  - Mesh rendering with isosurface generation
  - Interactive viewports rotate/zoom with mouse
  - Basic 3D scenegraph for easier programming
- **Data selection/marketing and region-of-interest controls**
  - Interactively mark vertical/horizontal locations and regions in plots
  - Widgets for selecting arbitrary regions from images and automatically slicing data to match
- **Easy to generate new graphics**
  - 2D graphics use Qt's GraphicsView framework which is highly capable and mature.
  - 3D graphics use OpenGL.
  - All graphics use a scenegraph for managing items; new graphics items are simple to create.
- **Library of widgets and modules useful for science/engineering applications**
  - Flowchart widget for interactive prototyping. Interface similar to LabView (nodes connected by wires).
  - Parameter tree widget for displaying/editing hierarchies of parameters (similar to those used by most GUI design applications).
  - Interactive python console with exception catching. Great for debugging/introspection as well as advanced user interaction.
  - Multi-process control allowing remote plotting, Qt signal connection across processes, and very simple in-line parallelization.
  - Dock system allowing the user to rearrange GUI components. Similar to Qt's dock system but a little more flexible and programmable.
  - Color gradient editor
  - SpinBox with SI-unit display and logarithmic stepping

**Installation:**

PyQtGraph does not really require any installation scripts. All that is needed is for the pyqtgraph folder to be placed someplace importable. Most people will prefer to simply place this folder within a larger project folder. If you want to make pyqtgraph available system-wide, use one of the methods listed below:

- **Debian, Ubuntu, and similar Linux:**  
Download the .deb file linked at the top of the page.
- **Other Linux:**  
Many people have generated packages for non-debian Linux distributions, including Arch, Suse, and Gentoo. Check your distribution repository for pyqtgraph packages.
- **Windows:**  
Download and run the .exe installer file linked at the top of the page.
- **Everybody (Including OSX):**  
Download the .tar.gz source package linked at the top of the page, extract its contents, and run "python setup.py install" from within the extracted directory (pyqtgraph is a pure-python library, so no compiling occurs during this installation). Or, install from pip using "pip install pyqtgraph".

**Requirements:**

PyQtGraph is known to run on Linux, Windows, and OSX. It should, however, run on any platform which supports the following packages:

- Python 2.7 and 3+
- PyQt 4.8+ or PySide
- NumPy
- python-opengl bindings are required for 3D graphics

**Documentation:**

Documentation is hosted [here](#).  
If you would like to request a specific section of documentation, please ask on the [forum](#). There are

A variety of plotting capabilities. (taken from examples/plotting.py)

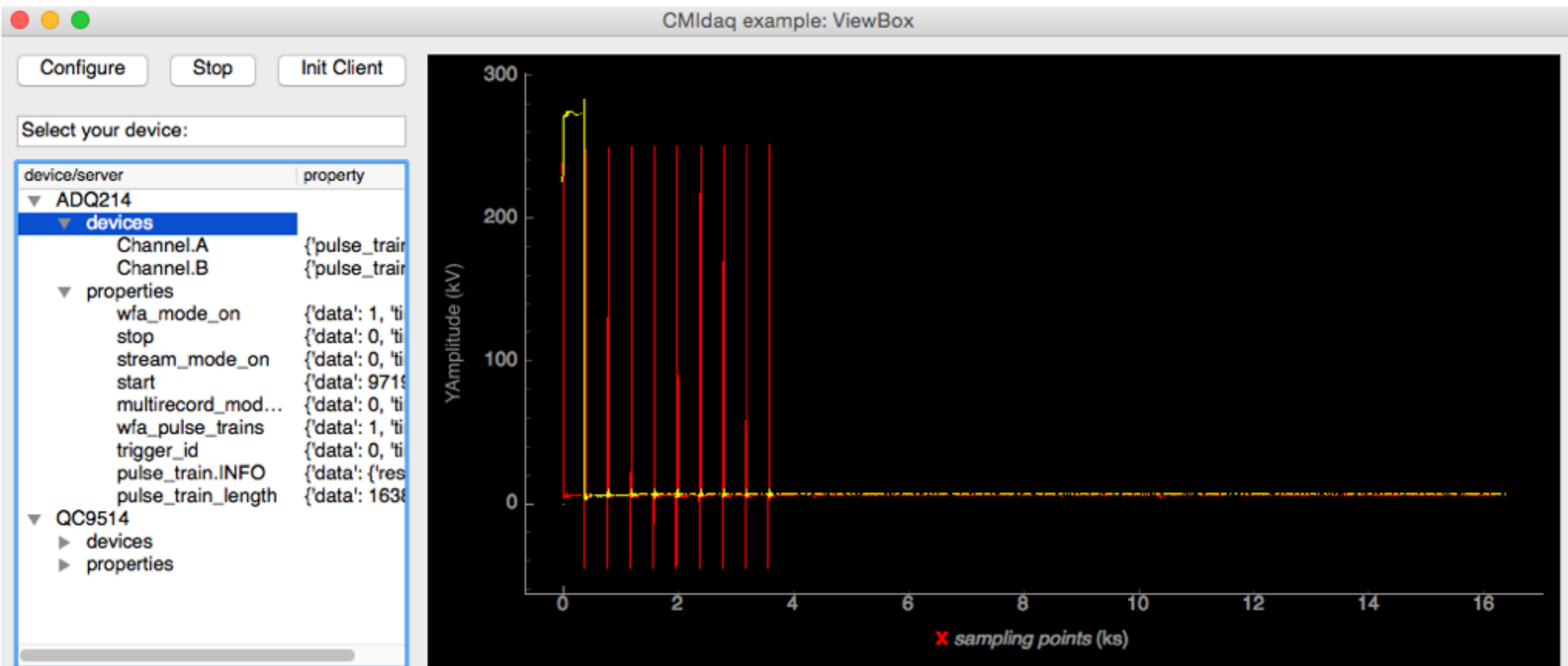
Image analysis with automated data slicing.

3D graphics: volumetric rendering, surface plots, scatter plots, and isosurfaces.

A variety of ROI types. Each ROI selects data from the underlying image and redispays it below. (taken from examples/test\_ROItypes.py)

# GUI USING PYQTGRAPH / VIRTUAL SCOPE

- Digitiser TineServer at 10 Hz (two channels) are sending a pulse 16384 (x4 byte, x2 channels) 128 KB/s
  - Python GUI manages fine, target data stream  $\times 10^3$
  - this GUI realisation has 702 lines of code (including comments and redundant code)



# GUI USING PYQTGRAPH / CODE EXAMPLES

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
### Local Variables:
### fill-column: 100
### truncate-lines: t
### End:

import initExample ## Add path to library (just for examples; you do not need this)

import pyqtgraph as pg
import pyqtgraph.exporters
from pyqtgraph.Qt import QtCore, QtGui
from PyQt4.QtCore import QObject, pyqtSignal, pyqtSlot

import threading
lock = threading.Lock();

import os
import ctypes as C

import h5py
import numpy as np

import PyTine as tine
tine.debug("set debug=1")

import time

# ----- #
context_name      = "CFEL.CMI"
experiment_name   = "FEL-2-simulator"
run_name          = "run000001"
servers={}

```

```
# ----- #
app = QtGui.QApplication([])
p_A = pg.PlotDataItem(pen='r')
p_B = pg.PlotDataItem(pen='y')
vb = pg.ViewBox()
model = QtGui.QStandardItemModel()
tw = QtGui.QTreeView()
# ----- #

```

```
## Always start by initializing Qt (only once per application)
def startApplication():
    mw = QtGui.QMainWindow()
    mw.setWindowTitle('CMIdaq example: ViewBox')
    mw.show()
    mw.resize(1000, 400)

    ## Define a top-level widget to hold everything
    w = QtGui.QWidget()
    mw.setCentralWidget(w);

    ## Create some widgets to be placed inside
    btn_init = QtGui.QPushButton('Init Client')
    btn_init.clicked.connect( initialise );

    btn_configure = QtGui.QPushButton('Configure')
    btn_configure.clicked.connect( configure_btn_clicked);

    btn_stop = QtGui.QPushButton('Stop')
    btn_stop.clicked.connect( stop_btn_clicked);

    text = QtGui.QLineEdit('Select your device:')
    listw = QtGui.QListWidget()
    plot = pg.PlotWidget()

    ## Create a grid layout to manage the widgets size and position
    layout = QtGui.QGridLayout()
    w.setLayout(layout)

    ## Add widgets to the layout in their proper positions
    layout.addWidget(btn_configure, 0, 0) #
    layout.addWidget(btn_stop, 0, 1) #
    layout.addWidget(btn_init, 0, 2) #
    layout.addWidget(text, 1, 0, 1, 3) #

    # operation on global tw:
    addTreeWidget()
    layout.addWidget(tw, 2, 0, 1, 3) # list widget goes in bottom-left

    ## Display the widget as a new window
    w.show()

```

```

gv = pg.GraphicsView()
layout.addWidget(gv, 0, 3, 3, 1) # plot goes on right side, spanning 3 rows

l = QtGui.QGraphicsGridLayout()
l.setHorizontalSpacing(0)
l.setVerticalSpacing(0)

vb.addItem(p_A)
vb.addItem(p_B)

rect = movableRect(QtCore.QRectF(0, 0, 1, 1))
rect.setPen(QtGui.QPen(QtGui.QColor(100, 200, 100)))
vb.addItem(rect)

l.addItem(vb, 0, 1)
gv.centralWidget.setLayout(l)

xScale = pg.AxisItem(orientation='bottom', linkView=vb)
l.addItem(xScale, 1, 1)
yScale = pg.AxisItem(orientation='left', linkView=vb)
l.addItem(yScale, 0, 0)

xScale.setLabel(text="<span style='color: #ff0000; font-weight: bold'>X</span> <i>sampling points</i>", units="s")
yScale.setLabel('<span>Y</span>Amplitude', units='V')

app.exec_();

```

# CMIDQA HDF5 DATA MODEL (STANDARD TOOL "HDF5VIEW" IN JAVA)

Tine context name, contains serves ADQ214, QC9514

the servers are used (NOW) within an experiment called "FEL-2-simulator" - run tag "run000001", contains

- Configuration settings of the HW units
- Data as HDF5 datasets for every DAQ

not related to the GUI, but rather to the Tine tools used

- every "experiment" access rights have to be restricted only to the devices assigned to this experiment
- [say only 5 out of 50 devices]

The screenshot displays the HDF5VIEW application interface. On the left, a file tree shows the hierarchy: `cfelcml_fel2simulator_001.h5` > `CFEL.CMI` > `FEL-2-simulator` > `run000001` > `Configuration` > `ADQ214` > `properties`. Below this, another branch shows `QC9514` > `Data` > `ADQ214` > `devices` > `Channel.A` > `Channel.B`. The main window displays a table of data for the selected path, showing columns for '0' and '0'. A status bar at the bottom indicates 'data (31528, 30)' and '32-bit integer, 1'.

```

# ----- #
context_name      = "CFEL.CMI"
experiment_name   = "FEL-2-simulator"
run_name          = "run000001"
servers={}

def get_tine_property(device_name, channel_name, prop_name):
    addr = "/" + context_name + "/" + device_name + "/" + channel_name
    print("get tine prop: ", addr, " ", prop_name)
    return tine.get(address = addr, property = prop_name)

def get_device_structure(device):
    global servers

    print("get_device_structure for %s and device %s " % (context_name,device));
    try:
        t=tine.list(context_name,device)
    except:
        print("device %s does not exist in the context %s " %(device, context_name) )
        return -1
    servers[device]={}

    servers[device]["devices"]={}
    for i in t["devices"]:
        servers[device]["devices"][i]= {}
        if "ADQ" in device:
            servers[device]["devices"][i]["pulse_train"] = 0 ; # get_tine_property( device, i, "pulse_train")

    servers[device]["properties"]={}
    for i in t["properties"]:
        if "pulse_train" != i:
            servers[device]["properties"][i]=get_tine_property(device, "Channel.A", i)

    return 0

def get_context_structures():
    print("get_context_structures")
    context_servers = tine.list(context_name)
    print("opening Context: ",context_name,"\n list of registered servers:\n",context_servers,"\n")
    for device in context_servers["servers"]:
        if "ADQ" in device:
            get_device_structure(device);
        if "QC" in device:
            get_device_structure(device);

```



```

def create_file(fname):
    print("create_file")
    # Create a new file using default properties.
    #
    file = h5py.File(fname,'w')
    #
    # Create a dataset under the Root group.
    #
    #print( "Creating a group Configuration in the file...")
    group_configuration_name = "/" + context_name + "/" + experiment_name + "/" + run_name + "/Configuration/"
    group_configuration = file.require_group( group_configuration_name )
    print( "Creating dataset ADQ214 in group [", group_configuration_name, "] for file: ", file)

    for i0 in servers:
        group0 = group_configuration.require_group(i0)
        for i1 in servers[i0]:
            group1 = group0.require_group(i1)
            for i2 in servers[i0][i1]:
                group2 = group1.require_group(i2)
                for i3 in servers[i0][i1][i2]:
                    if "pulse_train" not in i3:
                        #group2 = group1.require_group(i3)
                        value = 0
                        try:
                            value = servers[i0][i1][i2][i3]['data']
                        except:
                            value = servers[i0][i1][i2][i3]
                        value_len = 1
                        try:
                            value_len = len(value)
                            print( i3, " / ", value[0:10] )
                        except:
                            # print("value :",i3, " is not an array, set its length to 1")
                            print( i3, " / ", value )
                    print("require_dataset for ", i3, " shape: " , (value_len,), " dtype:", get_type(value) )
                    dataset = group2.require_dataset(name=i3, shape=(value_len,), dtype= get_type(value) )
                    dataset = value

```

# Many Thanks!

to Philip Duval and Mark Lomperski for the introductory course to TINE and helping me to sort out several not really obvious problems in the implementation and possibly misuse of the framework.

to-do:

- user & experiment & server restriction rules
- existing Tine services are not used by us a lot yet (wish to use as much as applicable in our case)

Comments and advices are welcome and will be highly appreciated.

Thank you!