Pals3D Documentation

Release 0.1

Aurélie Vancraeyenest

Mar 29, 2019

Contents:

Pals3D is a graphical application to support the use of a Time-Correlated Single Photon Counting system for application to Positron Annihilation Lifetime Spectroscopy.

CHAPTER 1

Purpose

1.1 Introduction

The Pals3D software is meant for performing Positron Annihilation Lifetime Spectroscopy measurements with two different modes for double and triple coincidences. It is designed to support the use of a PicoQuant Time-Correlated Single Photon Counting (TCSPC) TimeHarp 260 Pico PCIe card as acquisition system. The TimeHarp 260 (TH260) allows the accurate determination of arrival times of photons with high counting rates. Timestamped events are then filtered and sorted to produce lifetime spectrum histograms as output files, that could be further processed by dedicated tools for lifetime extraction (not provided by Pals3D).

1.2 Installation

Pals3D source code is freely available on Github

To start using Pals3D, you can simply clone the repository:

```
git clone https://github.com/avancra/Pals3D.git
cd Pals3D/pals3D
python pals3D.py
```

1.3 Citation

If you used Pals3D in research work of any kind or in a publication, please acknowledge its author(s) by one of the following methods

If you want to cite Pals3D, please use the DOI 10.5281/zenodo.2590978

For citation format, you can either use the one below, or go to the project publication page to get other citation export options.

Default citation format:

Aurélie Vancraeyenest. (2019, March 12). Pals3D - A data acquisition software for positron annihilation lifetime spectroscopy (Version v1.0). Zenodo. http://doi.org/10.5281/zenodo.2590978

1.4 License

Host software

The present software is distributed under the GNU General Public License version 3 (GPL-3.0) :

Copyright (c) 2018-2019 Aurelie Vancraeyenest

Pals3D is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

Pals3D is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with Pals3D. If not, see <http://www.gnu.org/licenses/>.

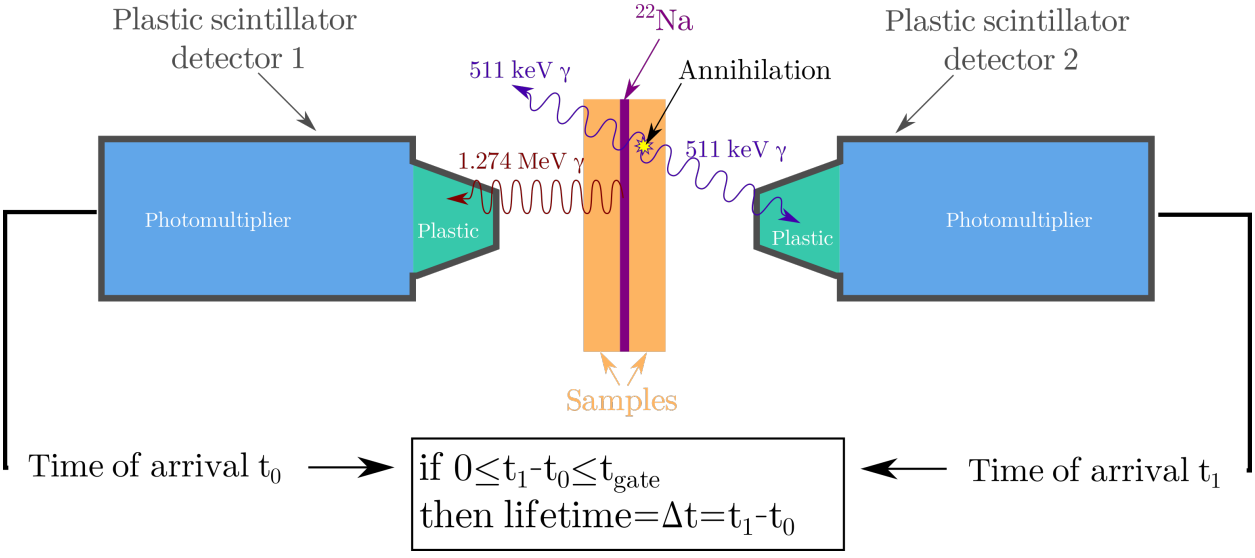
Firmware The PicoQuant DLL is not included with the current software and is subject to its own License and restriction. It can be purchase from PicoQuant directly together with the TimeHarp 260 system.

CHAPTER 2

PALS in brief

2.1 Standard double coincidence:

Positron Annihilation Lifetime Spectroscopy (PALS) is a well-known method to probe the nature of vacancy type defects in a given material structure. Indeed the positron lifetime before its annihilation in a given material is directly linked to the presence and the nature of the defects. PALS is thus an experimental technique that aims at measuring as accurately as possible the positron lifetime inside materials. To measure the positron lifetime, one has to determine simultaneously the time of creation of the positron and the time of its annihilation with one of the surrounding electrons. To determine those times efficiently, it is very common to use scintillator detectors as they have a very fast time response to energy deposition. As a positron source, ^{22}Na isotope is commonly used and it is sandwiched in between two pieces of the material to study. Below, is a schematic view of the experimental setup.

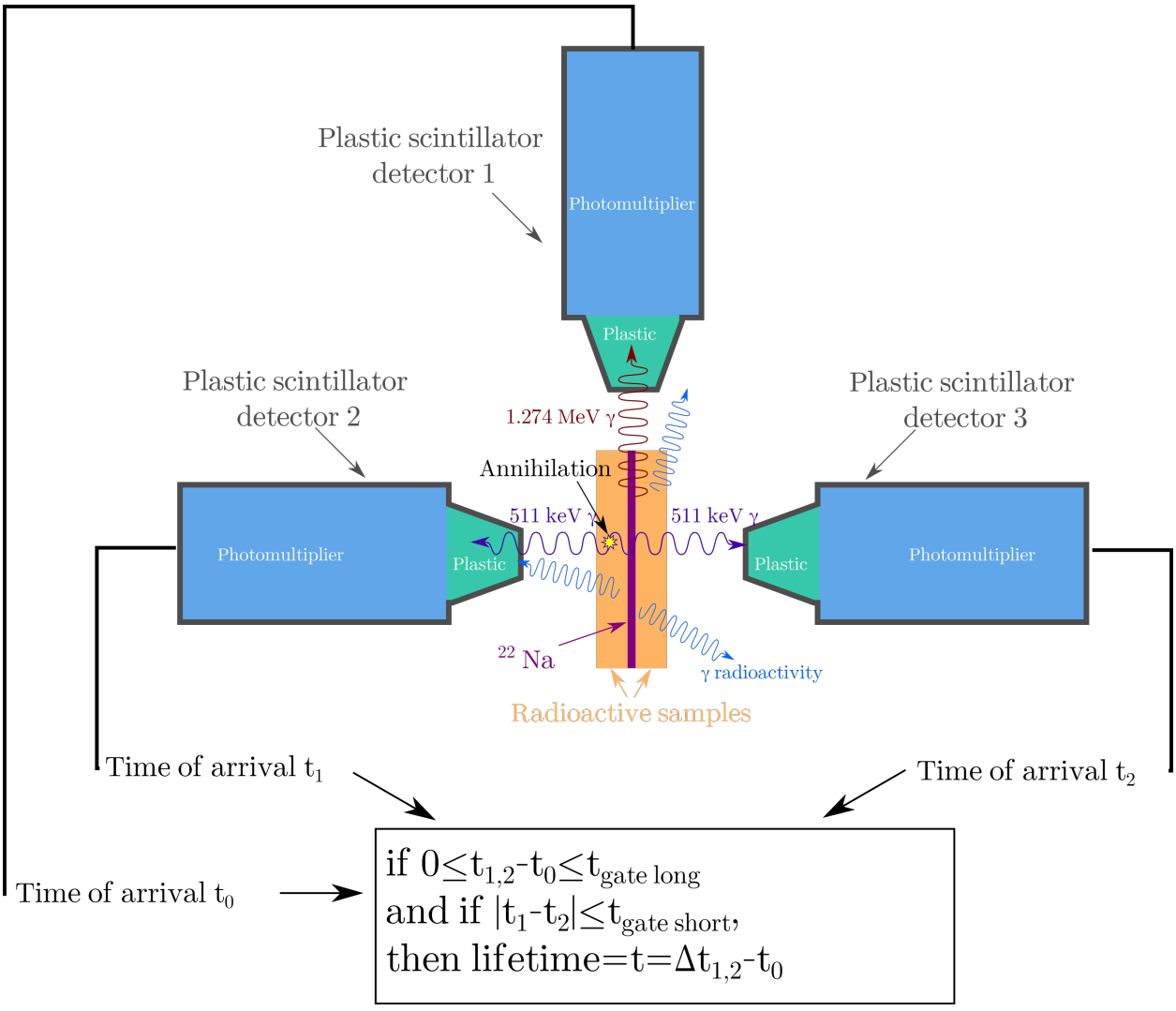


The positron is emitted first, nearly at the same time as the 1.274 keV photon which gives the start time (or t_0). After some time, depending on the nature of the material and its defects, the positron will annihilate emitting two almost co-linear 511 keV photons. In case of standard double coincidence PALS measurements the detection of one of these

511 keV photons gives the stop time (or t_1). Accumulating a large number of events, one can then determine the average lifetime of the positron, and gather information about the defects.

2.2 PALS for radioactive materials:

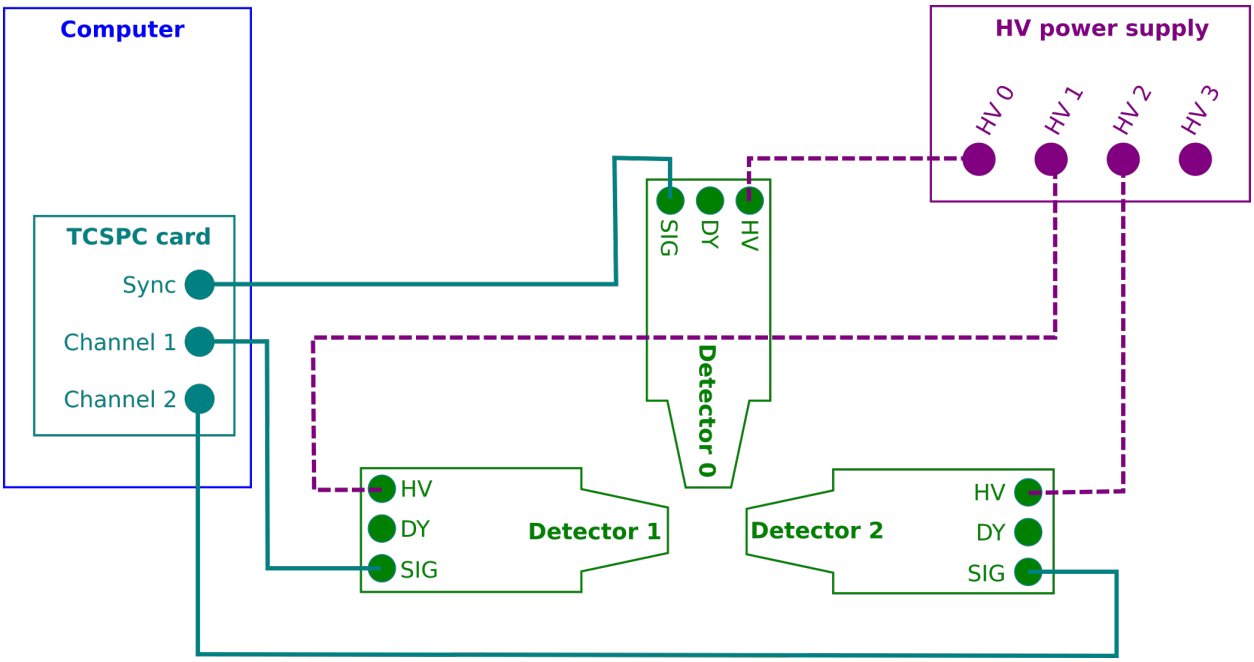
For the case of studying radioactive materials, it is needed to require the detection of both 511 keV photons. Indeed, gamma-radioactivity is producing false coincidence events that could distort the collected spectra. Thus, standard double coincidence PALS experiments can be performed only for low-activity materials (activity below 100 kBq). To overcome such difficulties, one may artificially reduce the radioactive background by adding a third coincidence trigger within the acquisition system, by coupling three scintillator detectors instead of two to detect simultaneously the two 511 keV photons and the 1274 keV one. A schematic of PALS principle in triple coincidence mode is given in the figure below.



CHAPTER 3

Hardware

Pals3D has been developed to control a Positron Annihilation Lifetime Spectroscopy experimental device. During the development phase of the software, the instrumental system was composed of 3 different hardware components: plastic scintillator detectors, a high voltage power supply and a Time-Correlated Single Photon Counting (TCSPC) PCIe-card. These hardware components are discussed in more details in the following sections. Here is a schematic view of the measurement setup and connections.



3.1 PicoQuant TimeHarp 260 Pico

3.1.1 Device:

The TCSPC card controlled by Pals3D is a PicoQuant TimeHarp 260 TCSPC system (version PICO) with PCIe Interface. The reader is referred to the hardware corresponding User's Manual and Technical Data supplied with the hardware for its handling and installation instructions, and more detailed information.

The TimeHarp 260 consists in 3 independent timing inputs (Sync channel, channel 1 and channel 2). The time of arrival of each input signal is measured with a resolution of 25 ps by means of a Constant Fraction Discriminator (CFD) coupled with a Time to Digital Converter (TDC) on each input. Fast rise/fall time for input signal must be within the [1;10] ns time range and its amplitude should be within the [-100;-200] mV range to be optimal (full range is between 0 and -1.2 V). Maximum counting rates of 40 Million counts per second (Mcps) with a deadtime of 25 ns can be achieved.

Two operating modes are available:

- The first one is histogram mode, where only time differences between the Sync channel and channel 1 or 2 are recorded. In case of PALS application, this mode is only suitable for double coincidence measurements.
- The second is the **Time-Tagged Time-Resolved mode** (TTTR) which consists in recording all events directly to disk. The TH260 support two TTTR mode, i.e. T2 and T3. In T3 mode, the sync channel is dedicated to a periodic signal used for synchronization of the other channels, whereas in T2 mode, no distinction is made between the sync channel and others. All timing input will be recorded in list mode, i.e. recording a timestamp and the channel number of each event. For this mode the TimeHarp software only allows to save the data stream to disk, resulting in huge amount of unsorted raw data. Pals3D is also using the TTTR T2 mode, but in addition it implements the data processing on-the-fly and produces time difference histograms for each channel couples.

Note: Note that using the TTTR T2 mode is mandatory when considering triple coincidence experiments.

3.1.2 Settings

Few settings are really mandatory to perform experiments using the TimeHarp 260 PCIe card, but for optimization purpose, we recommend that the CFD parameters and offsets are set as follows:

CFD parameters

The CFD zero cross is of little influence for PALS measurements and can therefore be set as indicated in the user manual of the TH260 device. On the other hand the CFD levels should be optimized carefully to allow the distinction between the 1274 keV photon and the 511 keV photons (see *PALS in brief* for more details about the PALS technique).

Note: Unlike standard PALS instruments that use an energy gate for start and stop photons, the TimeHarp 260 device only allows for a lower threshold. This is partly compensated by requiring that the three photons are detected in a given time gate that can be adjusted.

Channel offsets

For each TDC, an internal adjustable delay (called channel offset) can be set in order to finely tune the time difference between every channel. The channel offset should be set in order to time-align the different channels of the device. The time alignment can be performed using a ^{60}Co radioactive source.

- The channels 1 and 2 have to be aligned in such a way as to obtain a zero time difference in average.
- The channel 1/2 are set to be delayed by about 1 ns regarding to the sync channel. This first allows to ensure that events in the sync channel (detecting the start photon) should always be recorded earlier than any stop photon events occurring in the channel 1 and 2. This is of special importance for the triple coincidence sorter algorithm that assumes that the first event of a real triple event has to be in sync channel (see *Triple coincidence mode*). The second advantage of setting a positive delay between sync and channels 1/2 is that it makes possible to record the full time spectrum of the positron annihilation lifetime, including the rising part of the time spectrum and some background.

Note: The value of 1 ns in not of importance, only that it should be a positive value and large enough to ensure that events in sync channel are recorded with lower timestamps than events in channel 1 and 2.

3.1.3 Software

PicoQuant provides a dedicated software *TimeHarp* to operate the TimeHarp device and run standard experiments in histogram mode and TTTR mode T2 and T3. It is very useful to learn how to use the device as it is a plug and play instrument control. It can also be used as such for double coincidence PALS experiments, but only the T2 mode allows to do triple coincidence PALS experiments. Unfortunately, the TimeHarp software only allows for writing all raw events to an output file for T2 mode.

In case of short experiments the PicoQuant demo codes, provided in several programming languages, can be used to process those output files. The demo code is normally installed at the installation time. But they can also be found from Github PicoQuant-Time-Tagged-File-Format-Demos repository.

As mentioned earlier, Pals3D allows to run both double and triple coincidence PALS experiments via the T2 mode and processes events on-the-fly. To achieve that Pals3D is built on top of the PicoQuant TH260Lib DLL, so in order to run Pals3D it is required to install the DLL (provided as a separate package by PicoQuant).

The TH260Lib DLL also comes with some demo code that is normally installed at the installation time. But they can also be found from PicoQuant GitHub TH260Lib demos repository.

Note: The Python demos used to build up the Pals3D software were at the time of development only available on Github.

3.2 Detection devices

Three scintillation detectors built for ultra fast timing have been used for the experimental setup.

3.2.1 Devices

Detectors designed for ultra fast timing are required to perform PALS measurements. Usually, plastic scintillators coupled with fast photomultiplier tubes are used, but other scintillator types may be used as well. Shape and size of the scintillators depend on the compromise between the detection solid angle, time resolution, and geometrical set-up requirements. For triple coincidence measurements, the two detectors dedicated to the 511 keV photons must be co-linear and should have a limited solid angle aperture.

3.2.2 Characteristics

Here are the characteristics of the detectors used in the device used to develop and test Pals3D:

Each detector consists of a 25 mm diameter, tapered to 19 mm over 25 mm high EJ-232 Q 0.5 % plastic scintillator. Mounted in aluminum housing, the scintillator is coupled to a fast Hamamatsu R2083 PMT with built-in Voltage Divider with separate anode and dynode outputs. They were supplied by SCIONIX HOLLAND BV under the model number 25/19A25/2M-E1-EJ232Q0.5 %-X-NEG.

Before plugging the detectors, it is necessary to verify that they produce the right output signal shape and amplitude when hit by gammas. To this aim, a radioactive source such as ^{60}Co or ^{22}Na is necessary. The signal coming from detectors is then measured by an oscilloscope. The output pulses must be negative, with a 100-200 mV amplitude (depends on the gamma energy) and with rising/falling time of about 5-10 ns long.

3.3 HV supply

3.3.1 Device

The photomultiplier of a scintillator detector must be under high voltage to work properly. The HV nominal value is usually provided by the supplier of the detectors. Pals3D does not include any functionality to monitor the HV power supply which is then controlled by its own software. Therefore, any model having 3 independent and tunable channels should work. The device used to develop and test Pals3D was a DT5533EN 4 channels Desktop HV Power Supply from CAEN. Please refer to the corresponding documentation for more details.

3.3.2 Software

The HV power supply from CAEN comes with an instrument control software called GECO 2020 that is to be installed with administrator rights. The installation proceeds following the instructions provided by the vendor.

The connection to the device has to be configured to establish communication between the host computer and the power supply device. It requires settings of the COM port to initiate the communication protocol. The *Name* of the power supply can be tailored and the *Port Number* has to be chosen to be a free port of the host computer. The *Power Supply Type* and the *Connection type* as well as the *Baud rate*, *Data bits*, *Stop bits*, and *Parity bit* are set according to the supplier documentation.

CHAPTER 4

Features

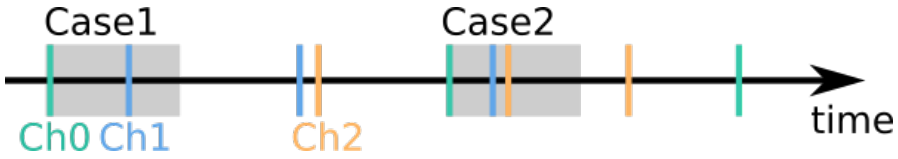
4.1 Standard acquisition mode

This is the standard acquisition mode for PALS measurements. It is provided with two selectable options from the GUI: double and triple coincidence.

4.1.1 Double coincidence mode

In this mode, only events composed of two consecutive photon events recorded within a given time gate will be considered as meaningful events. The time gate (in ps) for the coincidence is set in the GUI application under the *time gate long* field.

For sorting the double coincidence events, the algorithm makes use of a pairwise iterator that go through each pair of successive photon events and check for some conditions to be met.



The scheme above represents a typical event stream as red from the TH260 card's FIFO buffer. The gray box is a representation of the time gate as defined earlier. The sorter will go through each pair of successive photon events and check if the time difference between those two events is lower than the given time gate called *time gate long*. In addition, it makes sure that both events had occurred in different channels. If so, the relative time difference will be stored into an array corresponding to the channels involved.

Note: In the current version of the software, the time difference will be count as positive or negative depending on which channel the first event has occurred in. So all events involving channel 1 and 2 will be recorded in the same array, only changing the sign in case the channel 2 event happened before the one in channel 1. This way, both detectors can be aligned with a zero time delay and we still get the full time spectrum at once. By convention, all events with one of the following time structure: (chn0, chn1), (chn0, chn2) or (chn1, chn2) will be recorded with a

positive time, whereas every event with time structure (chn1, chn0), (chn2, chn0) or (chn2, chn1) will be considered negative.

At the end of each standard acquisition, histogramming of the three arrays is performed and the resulting histograms are saved to file (see *Output files* section).

Important: In the case 2 of the figure example, three events occur in the same time gate and all pair combinations are valid candidates for being considered as double coincidence events. However in the current state of the sorter algorithm only pairs (event 1, event 2) and (event 2, event 3) are examined. The pair (event 1, event 3) is not considered, so in case of a real triple decay event, this pair will be lost. This will not have any consequence for two photon decay physical events (such as ^{60}Co), but can lead to a loss of statistic in case of three-photon decay physical events, such as ^{22}Na. Also, as we have no means to know which pair is a true coincidence event or is not, we decided to keep both events as valid coincidence events. Thus, both time differences will be stored in the corresponding arrays. As we expect that the number of true coincidence events will be very significantly higher than the number of false one, we expect this choice to have little effect, and to not distort the time spectrum.

4.1.2 Triple coincidence mode

The triple coincidence mode is the standard acquisition mode for PALS measurements with radioactive samples producing high gamma ray background. This requires the additional condition that all of the three photons emitted in the physical process to be detected in a selected time gate.

As in the case of double coincidence mode (see *Double coincidence mode*) the algorithm makes use of an iterator to go through each triplet of successive photon events and check for some conditions to be met.

The scheme above represents a typical event stream as read from the TH260 card's FIFO buffer. The gray boxes are a representation of the time gates (bright gray for long time gate and darker gray for the 511 keV photon time gate). The sorter will go through each triplet of successive photon events and check the following conditions by step:

1. The *is3D* condition is fulfilled when:
   - the first event is recorded in the sync channel (chn 0)
   - the second and third events occurred in channel 1 and channel 2 (regardless of the order)
2. The *isInGate* condition requires to have all the three photons detected within the *time gate long* as in the double coincidence mode. Additionally a second time condition is required to ensure that the time difference between the second and third photons is lower than the *time gate 511* (also called *timeRes* in the code as it is related to the time resolution of the detection system)

If those two conditions are fulfilled, the relative time differences between each channel pair will be stored into an array.

Important: In order to meet the first condition of the *is3D* condition, it is important to carefully set the time offsets of the channels so that the sync channel timestamp is always lower than the one of channel 1 and 2 (see *Channel offsets*).

Note: As in the double coincidence mode the time difference between two channels will be count as positive or negative depending on which channel the first event of the pair has occurred in. However, if the time offsets are set

as explained in the *Hardware* section, events in sync channel should always occur before those in channel 1 and 2, leading to positive time differences for Sync- chn1,2 events. As a consequence, only the positive part of the spectrum is histogrammed and saved to file.

4.1.3 Output files

At the end of each standard acquisition, histogramming of the three arrays is performed and the resulting histograms are saved to file. The output file name is based on a name-base supplied by the user in the GUI application. To the name-base is automatically appended an additional suffix in the form *_XXX.hst* with XXX being the number of the current acquisition in a three digit format.

In addition to the histogram data the output files contain information on the hardware settings and acquisition parameters of the corresponding measurement. Below is an example of the header and first rows of a typical output file.

```
#Measurement date : Tue Feb 12 17:01:22 2019
#CFD settings:
#Channel | CFD ZeroCross | CFD level | Offset
#Sync -10 mV -200 mV 0 ps
#Chn1 -10 mV -60 mV 270 ps
#Chn2 -10 mV -60 mV 1184 ps
#Acquisition settings:
#Mode: 2C | long gate: 10000 ps | short gate: None ps
#Acquisition time: 10 min | file #1 out of 3
#
#time sync-1 sync-2 time chn1-chn2
0 0 0 -5000 0
25 0 0 -4975 0
50 0 0 -4950 0
75 0 0 -4925 0
100 0 0 -4900 0
125 0 0 -4875 0
...
```

The time histograms are provided in a 5 column format as shown in the file extract above. The first column represents the bin centers for the first two spectra (sync-chn1 and sync-chn2). Then y-values of those histograms are in the two following columns (2 and 3). The fourth and fifth columns provide the time spectrum (bin centers and values respectively) between channels 1 and 2. Details of the binning and bin edges for each channel pair is given hereafter.

By default, a binning of 25 ps is used and the edges of the histograms are defined as follows:

- Sync channel - channel 1 or 2: lower (left end) edge is always 0, and the upper (right end) edge is set as the value of the selected long time gate (*time gate long*).
- Channel 1 - channel 2: the edges are calculated so that the histogram is centered on the time 0 and its total span is set to the width of the selected long time gate. So it will have edges such as [-time gate / 2, time gate / 2].

As the PicoQuant TimeHarp 260 pico has an internal resolution of 25 ps, having bin edges precisely corresponding to multiples of 25 ps can lead to spectrum distortion. Indeed in that situation counts could be attributed to the wrong bin because of the machine internal round of floating point numbers occurring after decoding of the raw binary data. For this reason it was chosen to center the bins on values that are multiple of 25 ps.

For the triple coincidence mode, an additional output file is produced to allow further filtering of the events. Each triple event is recorded as a list of time differences of the kind: [Δ(sync-chn1); Δ(sync-chn2); Δ(chn1-chn2)]. All

events are then stored in a numpy array that is saved via the *numpy.save* method to an output file with the same file name as the histogram file but with the *.npy* extension.

4.2 Settings mode

Not yet available

Todo: implement this

4.2.1 60Co calibration

4.2.2 Offset optimization

4.2.3 Detector characterization

CHAPTER 5

Software

Pals3D is a graphical application to control the device, perform measurements and sort the data for positron annihilation lifetime measurements using a TCSPC system TimeHarp 260 pico from PicoQuant.

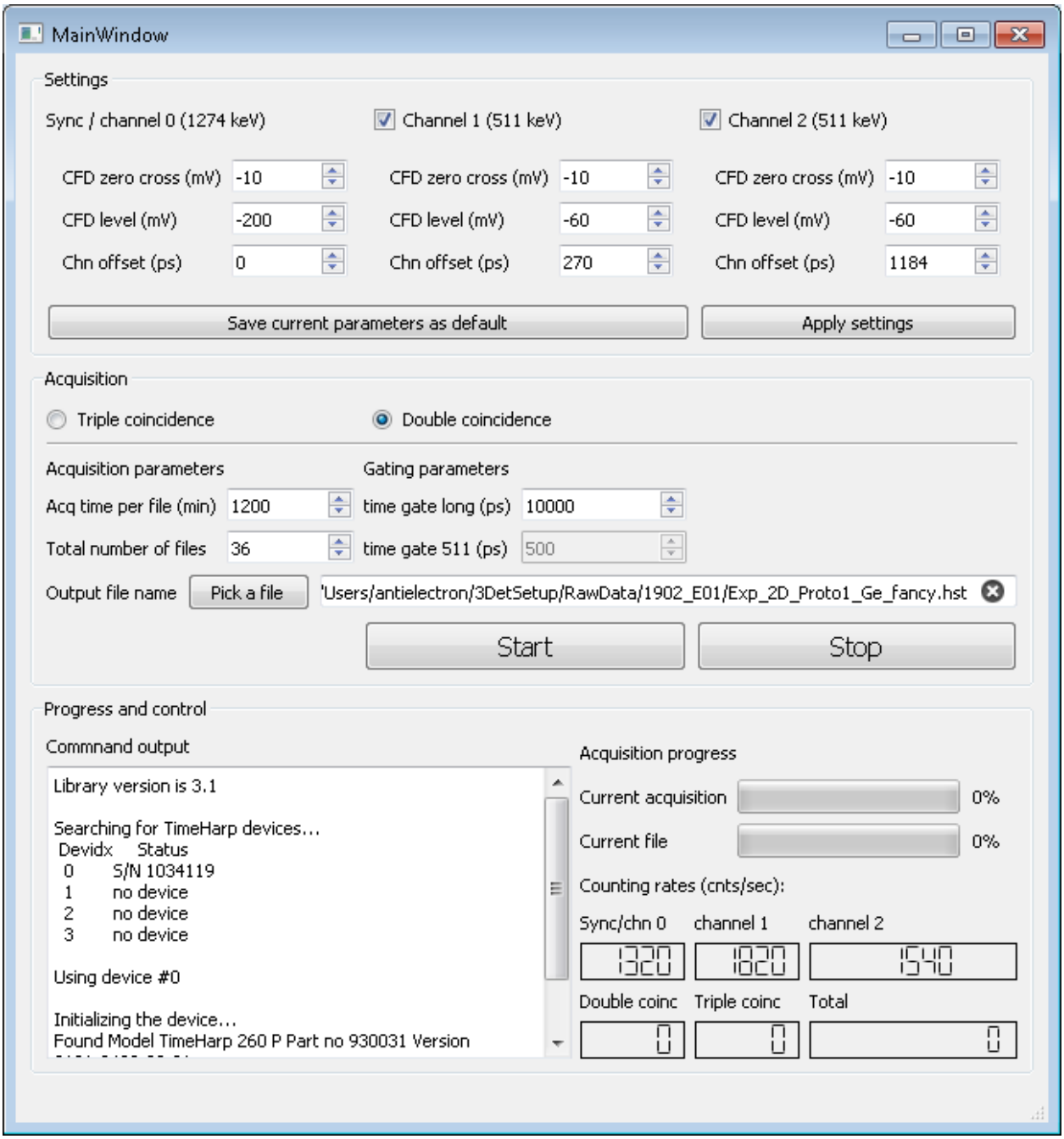
When installed as instructed in the *Installation* section the application is launched in a command prompt from the pals3D directory via the following command:

```
$ python pals3D.py
```

The next section describes the content and usage of the graphical interface, and the following one will go deeper into details of the main software components for future developers to get an overview of the software philosophy.

5.1 Graphical interface

The graphical interface has been designed using QtDesigner and PyQt5. Therefore any modifications to the interface should be done through QtDesigner and then exported to python using the `pyuic5` command.



5.1.1 Settings panel

In this panel of the upper part of the GUI, the user set the device parameters, i.e. the CFD parameters and the channel offsets.

The interface is configured so that it requires that the **Apply settings** button is clicked in order to actually apply the new settings. This is to avoid that the CFD parameters or offsets are changed during an on-going acquisition, which would render the results inconsistent (this should anyway not happen since the fields are disabled when a measurement is running). The **Apply settings** button, as well as the **Start** button, are disabled at application start because it requires time to initialize TH260 acquisition card and it can not be accessed during that time. Once the initialization process is complete, a message appears in the **Command output** and in the status bar, and the buttons are enabled.

A second button **Save current parameters as default** allows the user to save the current settings as default. Those values will then be reloaded at later start of the application. For this purpose it takes advantage of QSettings to store values as default for further use when restarting the application, so that when the settings are optimized there is no need to change those values anymore.

Note: The tick boxes in the settings panel in front of Channel 1 (511 keV) and Channel 2 (511 keV) are currently without any effect on the acquisition. They are meant to allow the usage of only one of the two channels without rising warnings but this functionality is not yet implemented.

5.1.2 Acquisition panel

This panel allows to select between the double and triple coincidence mode. For the double coincidence mode, only the *time gate long* is available as only one time gate is used in this mode (see *Double coincidence mode*). When the **Triple coincidence** box is checked the *time gate 511* field is enabled and the time gate for the short gate can be added (see *Triple coincidence mode*).

The details of the acquisition is also set in this panel. The user chooses an *acquisition time per file* value as well as a *total number of files* to be recorded. The application will then run the corresponding number of independent measurements of the given time.

The name of the output file is entered in the *Output file name* field either using the **Pick a file** button or manually. The default extension is *.hst* but at this point it does not matter since the filename entered here will be stripped from its extension if any and a *.hst* or *.npy* extension will be added automatically later on (together with the acquisition number). The name selected here should then be considered as a filename base that will be used as the common basis for all output files of the current acquisition.

The acquisition parameters will be saved as default through QSettings at each starting of a new measurement series. When later starting a new instance of the application the acquisition parameters will be loaded from the last measurement performed.

5.1.3 Progress and control panel

This part of the interface is meant for information to the user and as such allows no interaction.

Command output

The left-hand part contains a command output text box where all messages are sent that come from the various commands used in the application. For example, information about the device is written here when the device is first initialized at application start.

Each time the **Apply settings** button is clicked, the new parameters will be printed together with the new count rates obtained after applying the new settings.

Acquisition progress

Two progress bars show the advancement of the current measurement. The *current acquisition* refers to the number of individual measurement completed so far relatively to the *total number of files*, while the *current file* bar refers to the on-going file itself.

In addition, a more quantitative information can be found in the status bar, located at the bottom of the window, while the acquisition runs.

Counting rates

Six numerical displays allow to follow diverse count rates. The three upper ones, labeled with the channel names are live estimation of the counting rates in each channels.

The three lower ones labeled *double coinc*, *triple coinc* and *total* are only relevant during a measurement. They display the total number of events for one, and the number of coincidence events for the others, recorded so far for the whole measurement.

5.2 Software components

Pals3D makes a great use of PyQt signal and slot mechanism to communicate between different threads and keep the GUI responsive. Three main thread components can be identified as follow:

- The **GUI main window** takes care of all the interaction between the user and the application.
- The **TH260 controller** deals with every instruction that actually interacts with the PicoQuant TH260 Pico acquisition card.
- The **TH260 sorter** is in charge of all the processing of the data stream coming from the TH260 card.

5.2.1 Main window application

Interface design

The graphical user interface has been built using PyQt5 and uses signals and slots to implement the GUI logic. When possible the connecting slot by name convention has been used whenever possible. The software makes also extensive use of the *@PyQt.Slot()* decorator.

The GUI design has been done using Qt Designer and the `pyuic5` command to generate the corresponding python script. It is then highly recommended to make further change to the interface design using the same way. Note also that any change made inside the .py script of the GUI design will be overwritten whenever the `pyuic5` command is launched.

Threading

To keep the application responsive, threading is used in various ways. However, all kinds of threading make use of PyQt5 threading tools depending on the thread required:

- The main thread runs the GUI application itself and is started when the application is launched
- A timer is used for fetching the counting rates when no measurement is running
- For short and punctual actions, such as initialization of the device, a threadpool and a pool of workers are used to allow the user to interact with the software while those operations are on-going
- One thread is dedicated to running the measurement itself and its worker is defined in the TH260controller class (see *th260.th260controller module*). It takes care of starting the acquisition, fetching counting rates and checking for warnings during the whole duration of a measurement. Data buffers generated by the card are then sent over a signal to an other thread dedicated to the data processing
- The data processing is entirely done in an other thread so that the sorting time would not impact the acquisition and reduces the risk of overrunning the FIFO buffer of the card. The raw data buffer is received from the controller thread and will take care of unpacking the data, and sorting and filtering the events. At the end of each individual measurement, relevant events are processed into a histogram and then saved to an output file. See the section *Output files* for the detail about output file formats.

5.2.2 TH260 controller

As stated above, the TH260controller class takes care of all the interaction dealing with the acquisition card. It is built on top of the PicoQuant TH260Lib DLL which supplies all the necessary methods to control the device. The present version of the TH260controller is strongly based on the demo codes developed by PicoQuant available on GitHub PicoQuant GitHub demos.

Note: Please note that the TH260Lib is not supplied as a part of the present software and must be ordered from PicoQuant directly.

The ctypes foreign function library for Python is used to allow calling functions of the C DLL supplied by PicoQuant.

The TH260 controller defines a number of signals that allow the smooth delivering of information to the end-user through the GUI. The TH260controller has been made inheriting from the QObject class in order to use the signals and slot logic. Whereas this has been mainly designed to be used jointly with a GUI, the signals can as well be caught by other slots. For example, the *printOutput(self, text)* method allows for console output of text messages, and similarly writing data buffers to file instead of sending it to the sorter worker can be done in a very simple way.

5.2.3 TH260 sorter

We will not in this section discuss the way the sorting of the data stream is done, this is explained in the *Features* section, but we will discuss more technically how the task is performed.

The sorter worker receives the raw data buffer from the controller thread through a signal. It is received by the *sortBuffer()* method of the sorter class and the data stream is unpacked event by event following the data structure provided by the PicoQuant demo codes. The event is then handled depending of is type (real photon, overflow tag, markers). In standard PALS measurements only real photons and overflow events are expected so the case of marker events as been discarded.

Instead of being written to files, as in the demo codes, the events are then filled into a deque of fixed length for later sorting. When the deque collection reaches its length limit, it is sorted in double/triple coincidence events and after that, only time differences between channels are kept into an array. A signal is at the same time emitted to update the display of the coincidence event numbers of the GUI.

At the end of an individual acquisition, the remaining events in the deque (if not full) are forced to the sorting algorithms to not loose any events. Then the whole time difference array is histogrammed and save to an output file. Before a new measurement is started, all relevant class attributes are reinitialized.

CHAPTER 6

Pals3D

6.1 th260 package

6.1.1 th260.th260controller module

class th260.th260controller.**TH260Controller**

Bases: sphinx.ext.autodoc.importer._MockObject

TH260 controller class to configure and monitor a TH260 P card

TH260Controller is meant to access a PicoQuant TimeHarp 260 Pico via TH260LIB.DLL v 3.1. for application to positron annihilation lifetime spectroscopy. It is derived from QtCore.QObject to allow the use of signals and slots logic to communicate between thread workers (usually a GUI application and a sorter worker).

ACQTMAX = 360000000

ACQTMIN = 1

ACQ_ENDED

pyqtsignal() Signal send to the sorter worker to force the sorting of last events.

Type obj

CFDLVLMAX = 0

CFDLVLMIN = -1200

CFDZCMAX = 0

CFDZCMIN = -40

CHANOFFSMAX = 99999

CHANOFFSMIN = -99999

DATA

pyqtsignal(obj, int) Signal to share data object with the sorter worker of TH260 module. The first argument

is the data object itself, here a c_type array buffer. The second argument is the number of records contained in the data buffer.

Type obj

DEVINIT
pyqtsignal() Sent by the initialization method when the device is successfuly initialised.

Type obj

ERROR
pyqtsignal(tuple) Not yet in use.

Type obj

FLAG_FIFOFULL = 2

FLAG_OVERFLOW = 1

LIB_VERSION = '3.1'

MAXDEVNUM = 4

MAXINPCHAN = 2

MAXLENCODE = 5

MODE_T2 = 2

MODE_T3 = 3

NEW_OUTPUT
pyqtsignal(str) message to be printed in a console or GUI output

Type obj

PROGRESS
pyqtsignal(str, int) Signal to share the progress status of the on-going acquisition. The first argument should always be 'file' when use together with the Pals3D GUI application. 'file' refer to the status of a single acquisition file in opposition to 'acq' that relate to the totale acquisition programm, and that is sent by the acquisition worker of the GUI application. This can be changed when used with an external application.

Type obj

TH260LIB = <MagicMock name='mock()' id='140320117037544'>

TTREADMAX = 131072

UPDATECountRate
pyqtsignal() Sent to the GUI application to force the update of these values in the GUI application.

Type obj

WARNING
pyqtsignal(str) Warming messages for message box in GUI

Type obj

closeDevices()
Close the currently opened devices

configureSetting()
Set the card paramaters before starting a measurement

Parameters are either changed in the init function, or acquired through an external script or GUI. Here, we only set CFD parameters and channel offsets and the device resolution should always be 25 ns as we are running in T2 mode only.

getCountRates()
Get the count rates for each channels and store them

initialization()
Initialize communication with TH260 pico card

When initializationg is successfuly done, emit a DEVINIT signal.

printOutput(*text*)
Print a message to console output.

Meant to be used as slot for the NEW_OUTPUT and WARNING signals when no other slot (e.g. GUI application) is defined.

Parameters **text** (*str*) – Message to be printed to console output

searchDevices()
Search and list available devices on the host computer

startAcquisition()
Start data collection with current settings

Launch measurement according to current settings. Emit signals to communicate with other workers. Data buffers are emitted through a signal for being sorted by the TH260sorter module. Output messages and countrates are also sent over signals.

stoptttr()
Stop the ongoing TTTR measurement

tacq = None
Measurement time in millisec,

tryfunc(*retcode*, *funcName*, *measRunning=False*)
Check for errors when executing a function

If an error is raised, print the corresponding error message, stop the TTTR measurement if needed and close the device.

Parameters

- **retcode** (*int*) – code return by the function funcName (0 = success, <0 = failed)
- **funcName** (*str*) – Name of the function being executed
- **measRunning** (*bool, default False*) – specify if TTTR measurement is running. If True, the measurement will be stopped when an error occured, and then the device is closed

6.1.2 th260.th260sorter module

class th260.th260sorter.**SortingWorker**(***kwargs*)
Bases: sphinx.ext.autodoc.importer._MockObject

Data sorting class for data received from a TH260 PicoQuant card

Inherit from QObect to use Qt signal and slot logic for receiving and sorting data. Can then be easily used in a GUI application

Parameters **kwargs** (*kwargs*) – Keyword arguments passed from the main thread

dataArray

Type dict

dataDeck

Type deque

islastEvent

Type bool

globRes

Type double

resultArray_01

Type list

resultArray_02

Type list

resultArray_12

Type list

Keyword Arguments

- **sortingType** (*str*) – '2C' or '3C'
- **timeGate** (*int*) – Long time gate for positron lifetime (in ps)
- **timeRes** (*int*) – Short time gate for 511 MeV photons (in ps)
- **filename** (*str*) – Filename base for output file
- **CFDset** (*dict*) – Dictionnary of all the CFD settings for all channels
- **acqTime** (*int*) – Acquisition time (in min)
- **nftot** (*int*) – Total number of files during current measurement

COINCRATE
pyqtSignal(int)

Type obj

NEW_OUTPUT
pyqtSignal(str)

Type obj

T2WRAPAROUND_V1 = 33552000
Wraparound for version 1

Type int

T2WRAPAROUND_V2 = 33554432
Wraparound for version 2

Type int

VERSION = 2
Version ==> remove?

Type int

newMeasurement(*noFile*)
Initialise class attributes for new measurement

Parameters **noFile** (*int*) – File numero in case of multiple file acquisition

processLastEvents()
Force the processing of the last events to empty dataDeck

saveData(*noFile*, ***kwargs*)
Do histogramming of the whole data set and save to files

Parameters **noFile** (*int*) – Numero of the current acquisition file to be appended to the filename base.

sortBuffer(*buffer*, *nrecords*)
Decode buffer individual events to produce a time tagged event

Parameters

- **buffer** (*object - ctype array buffer*) – Raw data buffer received over a signal from the acquisition thread.
- **nrecords** (*int*) – Total number of records in the buffer

sortDeck()
Check if the data deque is full to start the sorting according to the sortingType ('2C' or '3C')

6.2 toolbox package

6.2.1 toolbox.plotting module

toolbox.plotting.**gaussian**(*x*, *x0*, *sig*, *amp*)
Formula for a Gaussian

toolbox.plotting.**lorentz**(*x*, *x0*, *sig*, *amp*)
Formula for a Lorentzian

toolbox.plotting.**plotFitHist**(*data*, *figname='fig.pdf'*, *bins=161*, *rmin=2000*, *rmax=6000*)
Plot an histogramm and fit with Gaussian and Lorentzian functions

Parameters

- **data** (*np.array*) – Input data
- **figname** (*str*) – Output filename of the file to save the figure
- **bins** (*int*) – Number of equal-width bins in the given range
- **rmin** (*int*) – max (most right-end bin edge) of the histogramm
- **rmax** (*int*) – min (most left-end bin edge) of the histogramm

See also:

plotHist(), *plotHists()*, *saveHists()*

toolbox.plotting.**plotHist**(*data*, *figname='fig.pdf'*, *bins=161*, *rmin=2000*, *rmax=6000*, *logY=False*)
Plot an histogramm and save the output spectrum as a figure

Parameters

- **data** (*np.array_like*) – Input data

- **figname** (*str*) – Output filename of the file to save the figure
- **bins** (*int*) – Number of equal-width bins in the given range
- **rmin** (*int*) – max (most right-end bin edge) of the histogramm
- **rmax** (*int*) – min (most left-end bin edge) of the histogramm
- **logY** (*bool*) – Allow to plot the histogram in a logarithmique scale

See also:

plotHist(), *plotHists()*, *saveHists()*

toolbox.plotting.**plotHists**(*data*, *figname='fig.pdf'*, *bins=161*, *rmin=2000*, *rmax=6000*, *logY=False*, *style='-'*)

Plot multiple histograms and save the output spectrum as a figure

Parameters

- **data** (*np.array_like*) – Input data as multiple numpy arrays
- **figname** (*str*) – Output filename of the file to save the figure
- **bins** (*int*) – Number of equal-width bins in the given range
- **rmin** (*int*) – max (most right-end bin edge) of the histogramm
- **rmax** (*int*) – min (most left-end bin edge) of the histogramm
- **logY** (*bool*) – Allow to plot the histogram in a logarithmique scale
- **style** (*str*) – Plotting style, see matplotlib for more info

See also:

plotHist(), *plotHists()*, *saveHists()*

toolbox.plotting.**saveHists**(*data*, *filebase='histo'*, *bins=161*, *rmin=2000*, *rmax=6000*, *useChnEnding=False*)

Sort each data-set in data into histogram and save as text files

Parameters

- **data** (*list of np.array_like*) – Input data asa list of several numpy arrays
- **filebase** (*str*) – Output filename base that will serve to form the output filename of each text file generated. Output file format is ASCII
- **bins** (*int*) – Number of equal-width bins in the given range
- **rmin** (*int*) – max (most right-end bin edge) of the histogramm
- **rmax** (*int*) – min (most left-end bin edge) of the histogramm
- **useChnEnding** (*bool*) – If Ture, channel ending type are inserted in the filename before the file extension. In False, an integer reprenseting the position in the input dataset is inserted instead

See also:

plotHist(), *plotHists()*, *saveHists()*

6.2.2 toolbox.utils module

toolbox.utils.**disableChildOf**(*widget*, *exceptThis=None*)
Disable all the interactive widgets of a group/parent

Parameters

- **widget** (*QtWidget name, e.g. self.widgetName*) – parent/group widget name for which all members will be disabled
- **exceptThis** (*QtWidget name, e.g. self.widgetName*) – name of the widget that remains enabled

toolbox.utils.**enableChildOf**(*widget*, *exceptThis=None*)
Enable all the interactive widgets of a group/parent

Parameters

- **widget** (*QtWidget name, e.g. self.widgetName*) – parent/group widget name for which all members will be enabled
- **exceptThis** (*QtWidget name, e.g. self.widgetName*) – name of the widget that remains disabled

toolbox.utils.**pairwise**(*iterable*)
s -> (s0,s1), (s1,s2), (s2, s3), ...

toolbox.utils.**tripletwise**(*iterable*)
s -> (s0,s1,s2), (s1,s2,s3), (s2,s3,s4), ...

toolbox.utils.**unique_everseen**(*iterable*, *key=None*)
List unique elements, preserving order. Remember all elements ever seen.

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

Python Module Index

t

Index