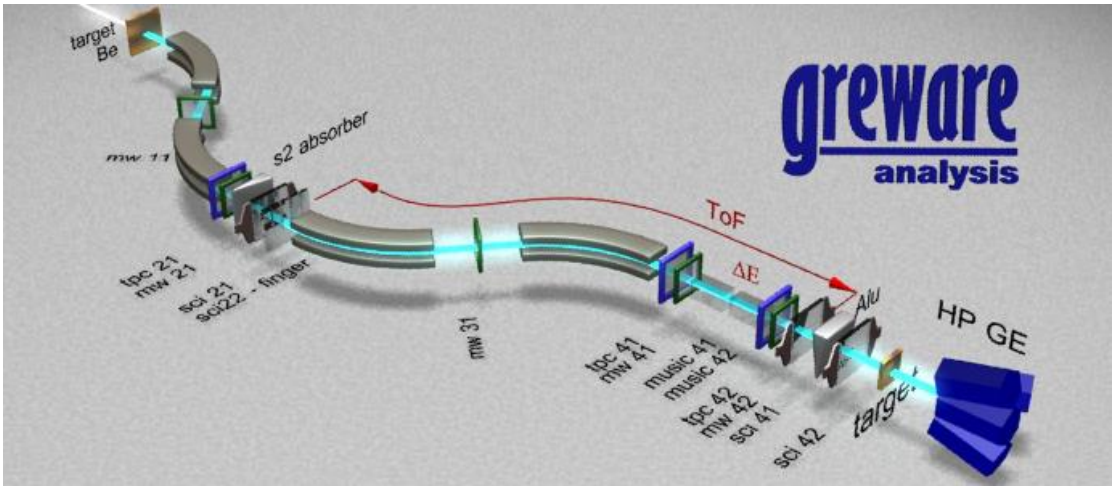


Jerzy Grębosz

GREWARE manual

How to make your own conditional spectra

Institute of Nuclear Physics, Polish Academy of Sciences
Kraków, April 2020

Table of contents

1. Introduction – specificity of online analysis

1. 1. Event-by-event data taking

GREWARE is a software for making a data analysis of the nuclear spectroscopy experiment data. The data which is collected in an "event-by-event" method.

An **event** is a set of numbers (delivered by the acquisition system) after each individual nuclear reaction. These numbers represent the results of measurements made by detectors that worked in a given reaction.

In this type of experiments a data analysis system works in a following order:

1) It receives subsequent data block from the data acquisition system (DAQ).
2) Unpacks the block – so the events are taken out of them: one by one.
3) Each event is subject to analysis. Raw data from ADC analog-to-digital converters are processed according to algorithms appropriate to the type of detector they come from. As a result of this processing, graphs in the form of spectra (one- and two-dimensional) are accumulated. By looking at them, the experimenter can monitor the correctness of the measurement system.

The core of the analysis program is a *program loop* in which subsequent events are analyzed. This analysis may be relatively simple (therefore fast), but usually an experimenter would like to perform more sophisticated operations on data.

online, offline, near-line

When events arrive from the acquisition system faster than they can be analyzed, the **online** analysis skips a part data blocks offered to it. Therefore, only a certain percentage of collected events are subject to online analysis. However, since 100% of events are saved to disk, you can analyze the entire data **offline** – at any time by downloading them not from the acquisition system, but by reading from a disk file.

In the online analysis, events are coming from the data acquisition system (DAQ). This type of analysis is useful for monitoring the correctness of detectors and associated electronic devices. However, if the experimenter would like to assess the physical value of the data, the physical result of the experiment, then he usually runs the data analysis program in the mode of downloading events previously saved in the disk file, which means that 100% of the data is analyzed. This mode of operation, when disk events are analyzed still during the experiment time, is often called **"near-line" analysis**. All of these modes of operation are provided by the event analysis system GREWARE.

1. 2. Where the eagles dare

Nowadays a complex physical experiment – requires a complex system of analysis. This means that a large group of experimenters depends on one or several people, who are able to modify the analysis, which should answer questions arising from the current situation of the experiment. It may sound like a bitter anecdote, but sometimes, during a night shift, when everything goes wrong, a professor might ask:

– Can we see this spectrum – gating with this isotope arriving from the fragment separator and gated with the position on this scintillator ? Very often the answer is: –No, because a PhD student, who knows how to do it, has just left.

Someone once jokingly said that humanists are people who know *"what"*, but don't know *"how"*. Technicians are people who would know *"how"*, but don't know *"what for"*. There is something wrong with this approach. Why do these two viewpoints have to be such an antinomy? Why can't these two viewpoints be combined? Create a system in which people who know *"what" (would like to study)* will also easily know *"how" (to get it)*; without having to read several hundred pages of documentation.

This became the starting point for me to come up with a system GREWARE that is described in this article.

What is a main obstacle in creation of such online analysis system, which would be capable of fulfilling the searcher's demanded requirements?

Actually, what demands are we talking about?

The "spectroscopy" experimenter wants to see spectra. The online analysis program creates many spectra, because it is known in advance that they will be needed – so they are available by default.

However, this is not enough. Soon comes a moment, when the experimenter would like to see some spectrum collected under some specific condition he invented. Just to see the first signs of his experiment's success, or to understand "why this doesn't work at all."

> So we need a possibility of creation of new conditional spectrum – created ad hoc, in the middle of an ongoing experiment.

Let's summarize. What features should have an online analysis system capable of easily meeting the experimenter's demands?

- The analysis system should be easy to modify or to create (invented ad hoc) conditional spectra.
- Creating new spectra and conditions should not require recompilation of the analysis program. In other words: it should be possible to satisfy the experimenter's demands while the system is running.
- The system should talk to the experimenter in a language that the physicist can easily understand (not in a language of a programmer).

This is how Spy-GREWARE was born

Motivated by the issues presented here, I invented online analysis system called GREWARE.

Its early version was named: CRACOW.

Starting from 2003, it was successfully used and developed during dozens of specific experiments of the RISING project (and its continuation – the PreSPEC project) at the GSI-Darmstadt Institute. Subsequent groups coming to GSI to make their experiments (after using this system) emphasized the ease of carrying out even a very sophisticated analysis. This inspired me to equip this system with suggested new features.

GREWARE were also used in experiments outside of GSI. These were: EXOTIC, PISOLO, PRISMA, AGATA and GALILEO experiments at LNL Legnaro (Italy) and PARIS-LaBr3-KRATTA experiments at CCB in IFJ PAN Krakow.

The user – it is you

Creating an analysis system that should be used not only the creator himself, but also the entire group of experimenters, requires an anthropocentric approach. Other members of the group will be system users. Thus the word "user" will appear repeatedly throughout this text. It should be remembered, that it is not about an engineer, not a computer scientist, but a physicist working on an experiment. It is about the part of his knowledge that is necessary for making decisions.

ROOT TTree

It is worth noting that the analysis system described here is really "***online*** analysis". That means a system, where the experimenter holds one hand on the potentiometer and his other hands operates the monitor, on which he watches a spectrum. The spectrum, which should immediately react to the changing setting of the potentiometer.

Without this demand – the user could simply use the so-called ROOT TTree – so popular in ***offline*** analysis.

By the way: the analysis system described here (Spy) also has the ability to produce Root TTree, in which it places the results of the analysis, but this issue will not be discussed in this work.

2. GREWARE analysis

2. 1. Two cooperating programs

The GREWARE analysis system consists of two cooperating programs:

4) The program analyzing events – called the **Spy**,
5) the **G**raphic **U**ser **I**nterface program (GREWARE **GUI)** responsible for:
   - *visualization of spectra,*
   - *changing analysis parameters,*
   - *creating new spectra and new conditions.*

Why two programs?

Separation of analysis and visualization

It is obvious, that an analysis program (the Spy) requires to set many parameters of its work. So it seems natural that just after launch of the Spy, this program should open some menu where the experimenter could provide the value of the desired parameters (gates, thresholds, etc.). In order to set these parameters in a user-friendly way – the Graphic User Interface (GUI) could be used. So far so good. No problem.

Dialog with the user should not interfere the analysis

However, sometimes – during the analysis – the experimenter would like to change a parameter; for example to change a gate in a condition of one of spectra. So the Spy program must stop the analysis loop and begin to talk with the experimenter. While the user is entering new values – the analysis of events is suspended.

Even if we solve this problem by using multi-threaded programming – we still have to deal with the situation where the experimenter providing some incorrect data – will cause a serious violation of the segment of memory (called segment violation) – and as a result the operating system will shut down ("crash") the analysis program. The results collected so far – will be lost.

Impasse? No!

Such eventualities can be avoided by applying a "Non-blocking GUI" technique. In this technique a usual work of graphical user interface (or even its “crash”) will not stop the analysis of events.

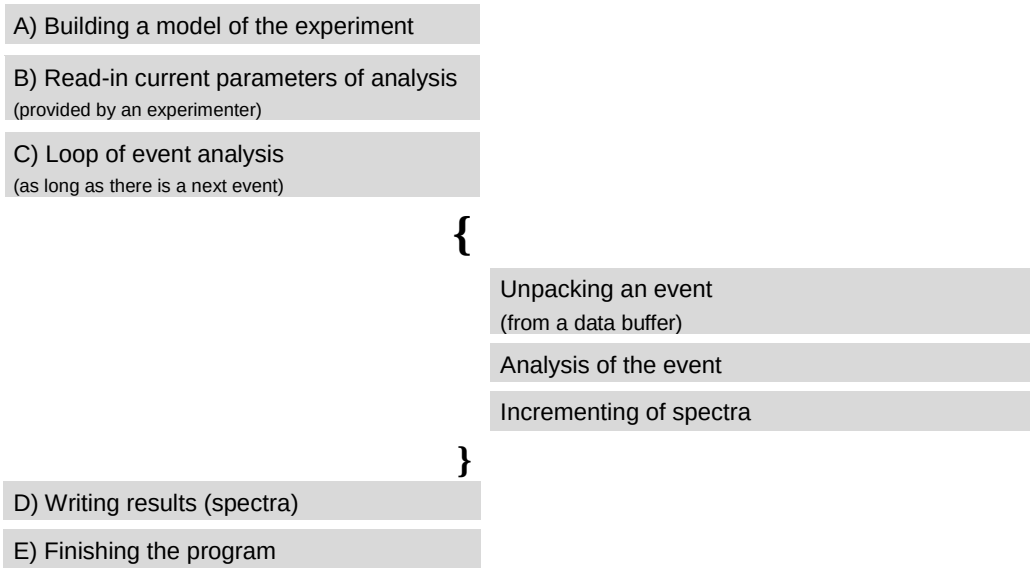
How it can be done? The solution is to divide work between two cooperating programs.

- The first of them (usually called **spy**) is just an event analysis program; the analysis is carried out according to the parameters prepared on the disk file.
- The second program (GUI) is responsible for all conversation with the experimenter:
  a) it can set parameters of the analysis;
  b) it can also show the results of current analysis (produced spectra);
  c) it can even allow to formulate some wishes (tasks) for the Spy.

The “GREWARE approach” is even more flexible: at any time during the analysis, you can change the value of parameters of analysis and send this change to the currently working analyzing program (spy). Seeing this, the spy will load new parameters and the next events will continue analysis according to the new settings.

2. 2. Structure of the event analysis program (Spy)

In the "event by event" data collection method, the analysis program itself can be divided into several basic parts. The most important of course is the analysis loop of subsequent events. In the diagram below you can see the others.



A) Constructing the experiment model

After starting the analysis system – objects representing experiment detectors and objects representing some algorithms are built.

It means that at this stage the constructors of these objects start working. The spectra are usually components of objects representing detectors – so every detector object is creating spectra for his own use.

During their construction, detectors objects (and algorithm-objects) – learn which other objects are their collaborators. In this way a set of relations between objects is defined. By this network of collaborators a model of whole experiment is build.

B) "Pre-Loop" phase

After building the model, the Spy program begins a phase in which all individual detectors (and algorithms) load (read-in) the parameters used for this particular analysis. These parameters can be very different type of data. For example:

- lookup tables (containing information: how the signals coming from specific electronic blocks – should be assigned to particular detector objects);
- calibration factors for individual signals;
- thresholds and gates values.

C) Event analysis loop

After loading all parameters of analysis, the Spy starts the event analyzing loop. What means to *analyze* an event?

1) In the simplest case it is just **calibrating** raw data.
2) In a more complicated case, several raw data can be used to **calculate** some physical quantity.
3) After calculations related to a given event for particular detector are done – the obtained values are used to **increment the system (build-in) spectra** showing the given quantity. Then the next event is going to be analyzed.

This is how the event loop works.

D) E) "Post-Loop" phase

The event-analyzing loop can be interrupted,

- when the set of data analyzed runs out,
- or at the request of the experimenter.

After this the "post-loop" procedures are called, (usually they are dealing with saving the results of the entire analysis). This is the action of saving the created spectra to disk. At this time build-in spectra (including elaborated conditional spectra) are saved. Usually in such spectra we look for the first signs of the success of our experiment.

3. Incrementers

3. 1. Interesting variables from an event analysis program

Let's assume that an experiment is ongoing already for a long time and the Spy program is analyzing events. By default the Spy creates build-in spectra that the experimenter can watch (using the GUI program). This is a normal monitoring of the experiment.

However, soon comes a moment when the experimenter would like to create an entirely new spectrum, which is not a build-in spectrum. Let's assume that it should be a spectrum of one of the variables defined in the analysis program

for example a spectrum of a variable called:
detector_german_E_4_energy_calibrated

Let's make a short day-dreaming:

The experimenter could express his wish using the GUI program. Then the GREWARE GUI program should send this wish to the event analysis program (Spy), who should create a desired new spectrum.

After compilation – names of the Spy variables are not known anymore

Unfortunately the GUI program cannot inform the Spy that the spectrum of the variable is called the detector_german_E_4_energy_calibrated. The *names* of the variables used in the source text of the Spy program – are "forgotten" during compilation and consolidation (linking) of the Spy program. At the time that the Spy is running – these variables are identified by their addresses.

So: the Greware GUI program cannot tell to the Spy program: "please make a new spectrum incremented by a program variable called… <*desired name*>.

What to do?

Here is a solution – a concept called an "incrementer"

Let's look at this problem more general. The Spy has to deal with thousands of his program variables. No all of them are equally important for a physicist making a data analysis. Some of them are specially interesting – for example:

- a value of energy measured by a particular detector,
- value of time measured by a particular detector.

If such variables may be interesting for the physicist, he may want to see spectra of this data, or even he may want to set a gate (condition) on a value of this variable.

Suspecting that some variable is so important to the users …

the author of the Spy makes a special object. In this object he assigns some text name to an important variable.

variable (its address) ← → human_readable_name

This concept (variable which has its own text name) is called: an '***incrementer***'. *(Because such a variables are mostly used **to increment** some spectra).*

A dictionary of incrementers' names

There can be plenty of assignment like this in the program (plenty of incrementers). All of them create a kind of dictionary. The Spy program saves this dictionary on the disk, so the GREWARE GUI program can read it and show these names of available incrementers to the users.

Thanks to this:

1) the user can ask GUI that he would like to have a new spectrum with an incrementer having a particular name *xxx*.

2) Then the GUI sends this wish to the spy, that we are interested in a incrementer/variable called *xxx.*
3) Getting this wish, the Spy creates a new spectrum knowing that this spectrum should be incremented by the variable *xxx* (located under a particular address).

✧

The variables that interest the experimenter (from now on, we call them: incrementers) are not only the data measured directly by the detectors. These are also variables representing the data:

- calculated according to some algorithms
- or even logical values resulting from the fact that something occurred or not.

3. 2. Incrementer – is a key concept in the system described here

The incrementer specifies any variable in the analysis program that has been made available to the experimenter – so that (from the GUI program position) he can use it to interactively create his spectra.

The incrementer has some important features. Among others there are:

- Name of the incrementer;
- Address of the variable, which is represented by this incrementer;
- Validator of this incrementer;
- Pointer to the owner object (a detector) which created this incrementer.

The first two of these features are clear. Now we will talk about the last two.

3.2. 1. Validator of an incrementer

The incrementer representing the **multiplicity** of γ quanta registered in a given event – **always has a defined value**. Even if it were to be zero. Zero means: zero gamma detectors fired.

> However – not always it is so simple. There are variables (incrementers) – which sometimes cannot always have a precisely defined numeric value. In some events they may be 'unspecified'.

This can happen in case of a variable representing data which is a result of some calculation.

For example, the value of the variable representing the position of the nucleus passing through a multiwire chamber – is calculated using two data provided by the left and right electrodes. It may happen that one of these electrodes did not fired in a given event. So the position cannot be calculated.

In such a situation there is a problem, **what value** should be placed in the indicated variable.

Even if no value is assigned to the variable, the pre-stored zero will remain there.
Unfortunately such zero value, (in the case of a wire chamber), means that the ion flew exactly in the beam axis.

So, if someone will use this (zero) value of the incrementer (in such an undefined situation) – this will lead to incorrect analysis results.

To avoid such errors, the incrementer has been equipped with an additional component. This is a pointer to a logical variable called: a ***validator***.

> A validator is another local variable, usually of a logical type (bool), which informs whether the calculation of a value for a incrementer variable was successful/possible or not.
> *Such variable-validators are very often used in data development procedures.*

The incrementers for which a validator is essential –in their names usually have some suffix as:

...when_fired (when its detector fired)
...when_good (when its detector fired and we consider value as approved)
...when_doppler_correction_successful

3.2. 2. Address of the incrementer owner

Another useful component of the incrementer is the address his owner. That means the address of this detector-object where the given incrementer was created.

Usually this incrementers are "born" in an object representing some detector or some algorithm.

The system automatically remembers this address.

When it can be useful?

Sometimes we want to check if two incrementers are coming from the *same* detector or from the *different* detectors.

For example we have an incrementer representing the energy registered by a detector and another incrementer representing a time registered by a detector.

We would like to check if they represent the energy and time from the **same** detector. The Spy can check this just by looking at this *addresses of the owner* detector in both incrementers.

Sometimes we want to check if two incrementers are coming from the different detector

For example we have an two incrementers representing the energy registered by gamma detector. We would like to make γ–γ energy coincidence matrix, but we want to increment it only when these two coincident energies are really from ***different*** *gamma detectors.*

The Spy can check this by comparing this owner-addresses of both gamma energy incrementers.

4. Spectra defined by the experimenter

Incrementers are available to the user, because their list, is saved by the Spy program to a disk and becomes available in the GREWARE GUI program. Thanks to this, the experimenter can see such a list on a screen. From this list he can select the name of the incrementer he is interested in. He can use it while defining a new user-defined spectrum. How to make this, we will learn in this chapter.

4. 1. Spectra in the online analysis program

Spectra built-in into the analysis program by default

In the code of the analysis program – there are definitions of a huge amount of standard spectra placed there by the developers of the program. They are 'build-in' in the program by default, because it is obvious that they are needed. Like the rest of the program, these definitions are compiled and linked. Shortly after the start of the analysis program Spy – the spy creates them and they are ready for work.

User spectra

In the case of spectra created ad hoc by the experimenter – the situation is different. At the time when the Spy is compiled, or even when it is launched – it does not know yet anything about any new spectra invented by the user. In the code of Spy, there are neither user-spectra definitions nor any instructions incrementing these spectra.

No problem, because the Spy program (while is already running):

- has a mechanism for creating any new spectra, which we will ask him now,
- has a mechanism for incrementing these spectra with values coming currently the experiment.

The creation of new objects during the program operation is ensured by dynamic memory allocation procedures. The spectrum can therefore be brought into existence during the Spy run.

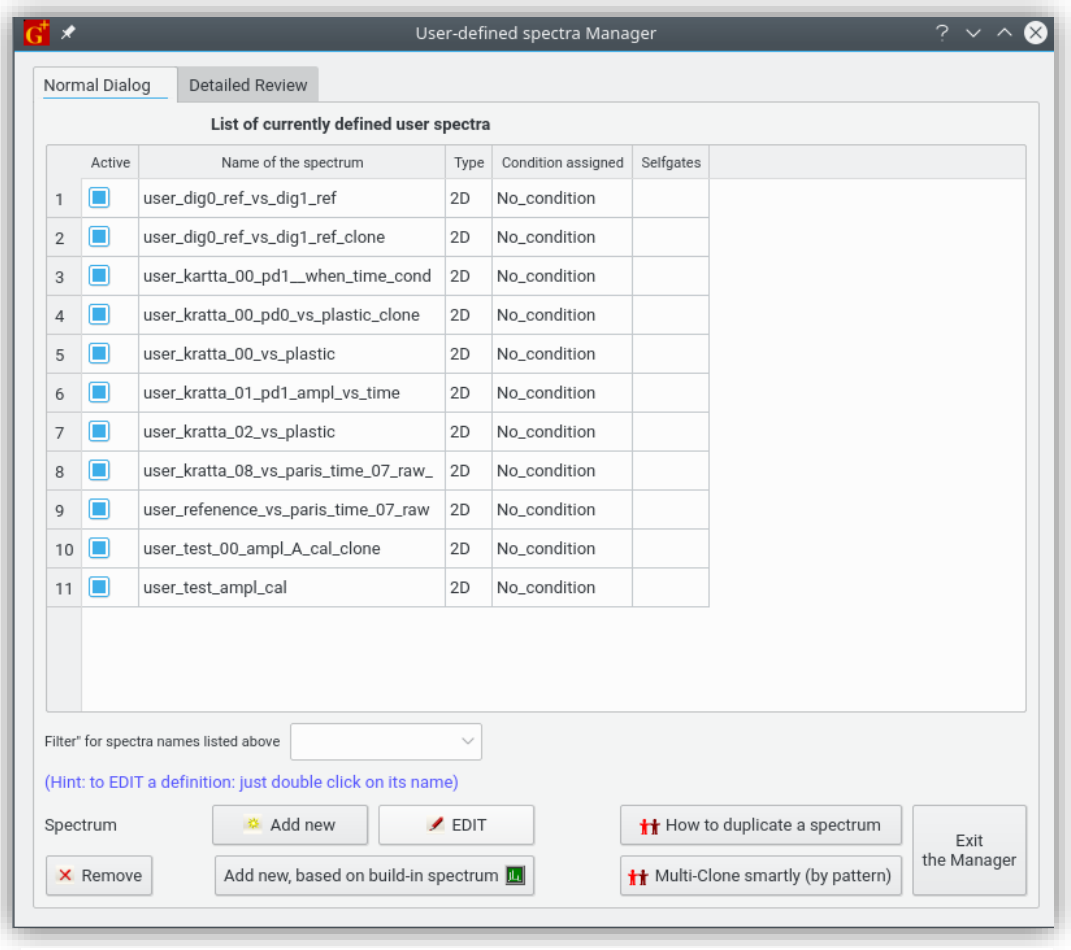
4. 2. Spectrum manager and a user spectrum definition creator

The Spy can create a new spectrum required by the experimenter, but the user must provide a detailed description of a desired spectrum.

> *This description is in the form of a text file with specific syntax. This description should contain information on the size (range) of the spectrum, as well as on variables that should increment this spectrum (the names of incrementers).*

Lucky the user does not have to prepare such a description himself. This work is done with a help of a so-called "spectrum wizard" (which is part of the GREWARE GUI).

The spectrum wizard can be started in the "User-defined Spectra Manager" dialog box. (**Błąd! Nie można odnaleźć źródła odwołania.**)



Błąd! Nie można odnaleźć źródła odwołania.. General look of the User-defined Spectra Manager.

What the spectrum manager can do for us? It can:

- create definition for a brand new spectrum,
- create definition which is based on some build-in spectrum *[NEW!]*,
- modify (edit) existing definition of a spectrum,
- clone an existing user-defined spectrum,
- delete a spectrum definition which is no longer needed,
- review a full detailed list of parameters of all spectra definitions (in the form of a large table).
- smart multi-clone of chosen spectrum *[NEW!]*

4.2. 1. Spectrum creator

If the experimenter press the button "Add new spectrum", the spectrum definition wizard appears. A wizard is a dialog which has several pages. On individual pages there are questions about the features of the new spectrum which we want to create.

> Note, that **the wizard does not create a spectrum, but only its description** in a language understood by the analysis program (Spy). As a result of the wizard's work, a small text file is created. This file will be read by the Spy program. Based on the information contained in this file, the analysis program will create the desired spectrum.

The wizard is like a list of questions to be answered by an experimenter. They are placed on many pages of the wizard dialog.

The next questions which will be asked – depend on the answers already given. For example, if the experimenter decided that he wants to have a two-dimensional spectrum, then on the following pages appear only questions specific to the description of the two-dimensional spectrum.

Spectrum wizard – step by step

Let's follow the simplest case – that is, let's define a one-dimensional spectrum in a simplified form.

After starting the spectrum definition procedure – a wizard dialog box appears. On its first page we must provide the name of the new spectrum. The name of the spectrum must always have a prefix "user_. (The wizard will add this prefix if you forget).

After entering the name of desired spectrum, we go to a next page of the wizard. Here we decide on the dimension of the spectrum; whether it is a one-dimensional (1D) or two-dimensional (2D) spectrum (matrix). Suppose the answer is: "one-dimensional".

Resolution of the spectrum

On the next page of the wizard we must define a resolution and range of the 1D spectrum (binning). (**Błąd! Nie można odnaleźć źródła odwołania.**)

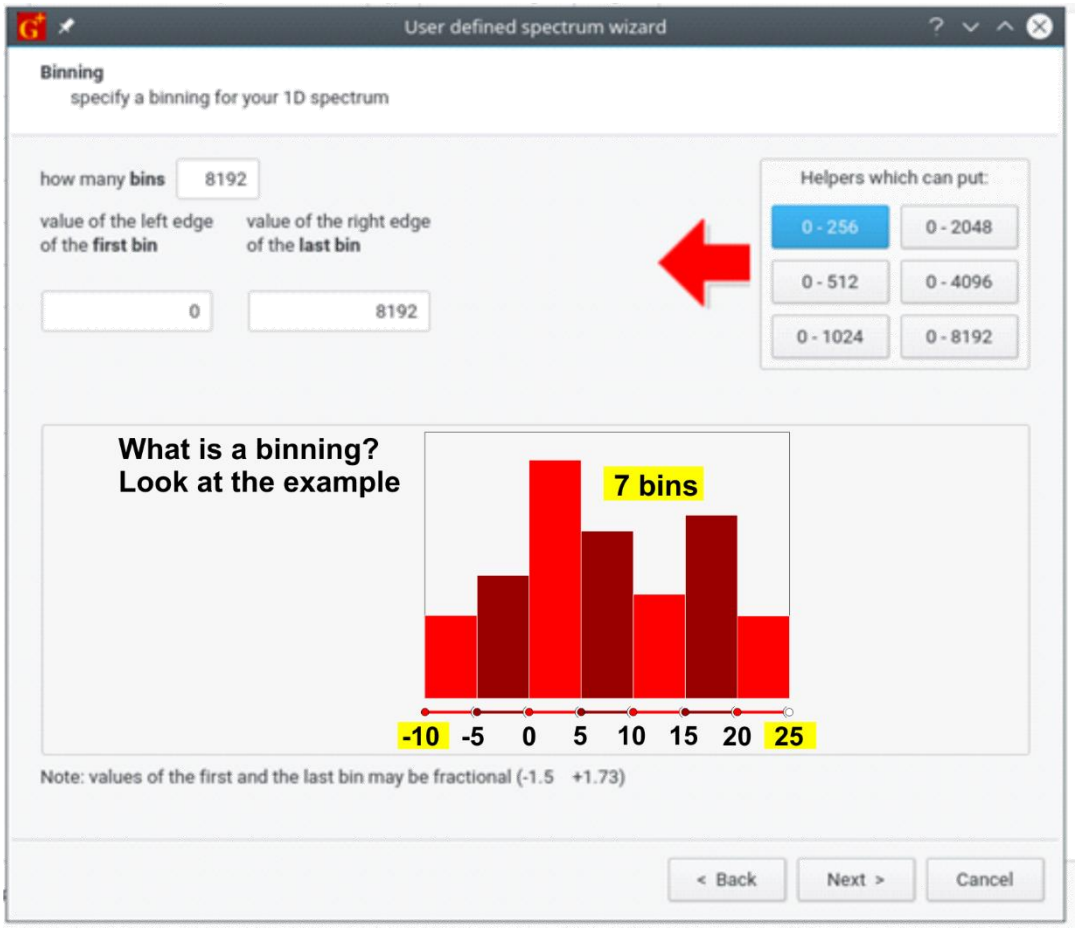


Figure 2. The "binning page" of a 1D spectrum – in the spectrum wizard

So we must write:

- how many channels the spectrum area is divided into, (bins)
- value representing the left edge of the first channel (min),
- value representing the right edge of the last spectrum channel (max).

  For example: a given spectrum should have a range (–150, +150) divided into 4096 channels.

The spectrum range can be described even by real numbers.

For example at GSI RISING experiment we used to have the spectrum of "A/q" which was usually defined in the range of 1.5 – 4.0

In this way, the basic 1D spectrum parameters were determined.

- If it is a one-dimensional spectrum – these numbers determine the parameters of the spectrum X axis.
- If you define a two-dimensional spectrum (matrix), the wizard also asks for similar parameters describing the Y axis.

The look of the 2D binning page will be shown in one of next paragraphs.

A new, nice option is…

…a helper which can automatically insert most popular 'binning data' into the 'binning fields'.

You can use this helper option, which is closest to this, what you need, and then you can edit the answers to this what you really want.

Who will increment our spectrum

The parameters described so far determine the size of the spectrum, but they say nothing about **what** is to be presented on the spectrum. These decisions we must make now. On the next page of the wizard we see a table in which we can place the name of the incrementer which we want to increment our spectrum.

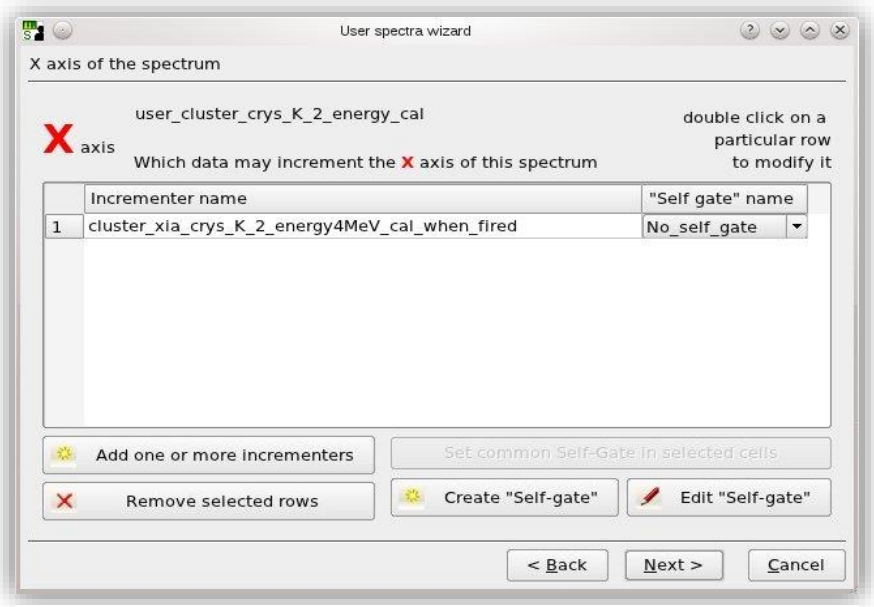


Figure 1. A page of wizard where we need to specify incrementers which will increment our spectrum. In this case the table has only one row, because we decided to use just one incrementer.

ANCHOR PODPISU

There are several thousand incrementers available. How to find a proper one?

The GREWARE GUI does not expect that you remember the names of all incrementers. GUI gives you a help: a tool which displays all available incrementers. From the hundreds of incrementer's names you can easily filter out a desired one.

How to do it? The "Add one or more incrementer" button opens a small dialog box, which is listing all available incrementers. This is the "dictionary" list that Spy makes and writes to disk every time it starts. The spectrum wizard opens this list and displays it on the screen. (See Figure 2).

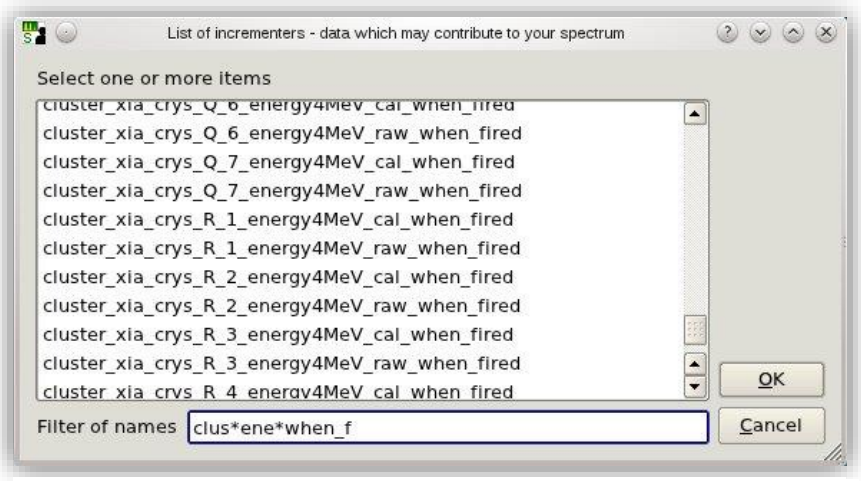


Figure 2. A dialog window containing a long list of all available incrementers.

Selecting the correct incrementer from this list (which has several thousand items) is not difficult because there is a convention that the incrementer's name starts with the name of the detector in which it was defined. Here, very useful is a filter field, where we can type a name of the selected detector. Thanks to this filter only these incrementers will be displayed, which apply to the selected detector. Usually we will see several items – names of incrementers. From this short list we can select the right incrementer (by clicking on it).

After making the selection and pressing OK, the incrementer's name will be automatically copied into the table.

Sometimes a simple filter string is not enough.

There are some multi-segment detectors, which have many incrementers. In case of such detectors a "simple " filter is not enough – because still we see too many incrementers on the list. No problem.

> You can make more elaborated filter by using the * (wildcard) characters inside a filter string.

This should narrow down the list of incrementers presented for your selection.

Mysterious 'Self gate' (Local gate)

In the table of incrementers' names – there is also a column called "Self-gate name". We will explain its meaning later (chapter 8, page 60). We will also deal later with the wizard's last question – the condition in this spectrum (chapter 7 page 44).

Finishing the definition

If we give all the required elements of the spectrum definition – we can press the "Finish" button. In this way, the process of defining the description of a desired spectrum has basically ended. GREWARE GUI saves this information to disk as a text file. This file is like a letter written by the GUI to the Spy program. A letter in which the experimenter's requests are written in a language that the Spy program understands. If the Spy is currently running, it can temporarily stop analyzing events and generate the desired spectrum. Then the spy returns to analyzing further events. From now on, the new spectrum will be slowly filled with data coming from subsequent events.

The user spectrum is permanent

It is obvious that sooner or later the Spy program will end its work and stop.

> However, the definition of the new spectrum is stored on disk. So the next time you run the analysis program, it will create a spectrum again (according to this description). This is a very important feature, because thanks to that – the spectra definitions are "permanent".

Experimenters can gradually build entire sets of spectra which are useful for their experiment. This resource can constantly grow.

Result – for example…

We just saw how simple (using the wizard) – you can create a definition of the spectrum you need. For example, in the RISING experiment there are 105 germanium crystals working as gamma quantum detectors. In the above way you can create a spectrum showing the energy of quanta γ registered by one of them. The following picture shows the spectrum that,

by using the incrementer called clus_crys_K_2_energy4MeV_cal_when_fired

shows the energy spectrum of γ quanta detected in a germanium detector (cluster type) called "K" by its crystal No. 2.

ANCHOR

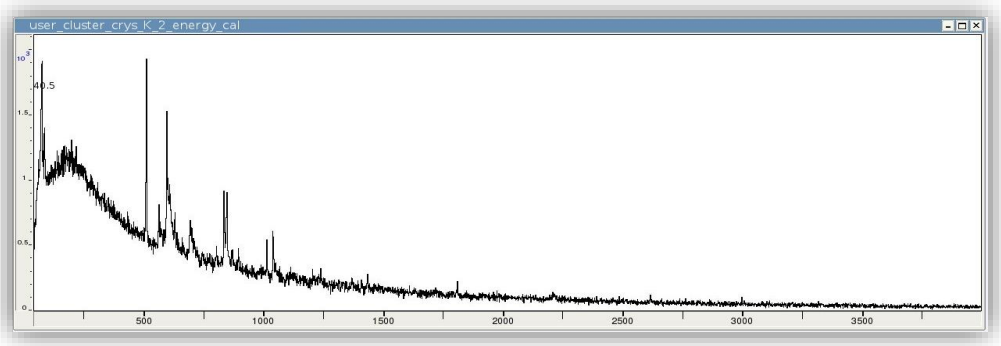


Figure 3. Energy spectrum of quantum γ recorded by one germanium crystal RISING detector, prepared using one incrementer called: clus_crys_K_2_energy4MeV_cal_when_fired.

4.2. 2. Spectrum can use more than one incrementer

'Sum' or 'total' spectrum ***(a poor solution)***

In the practice of the RISING experiment, the spectrum most watched by the experimenters was a spectrum of energy of γ quanta detected by **all** 105 crystals ("total energy spectrum"). Such a spectrum could by:

1) creating 105 elemental spectra of individual crystals
2) and adding this spectra up.

There is a simple procedure which allow this summing.

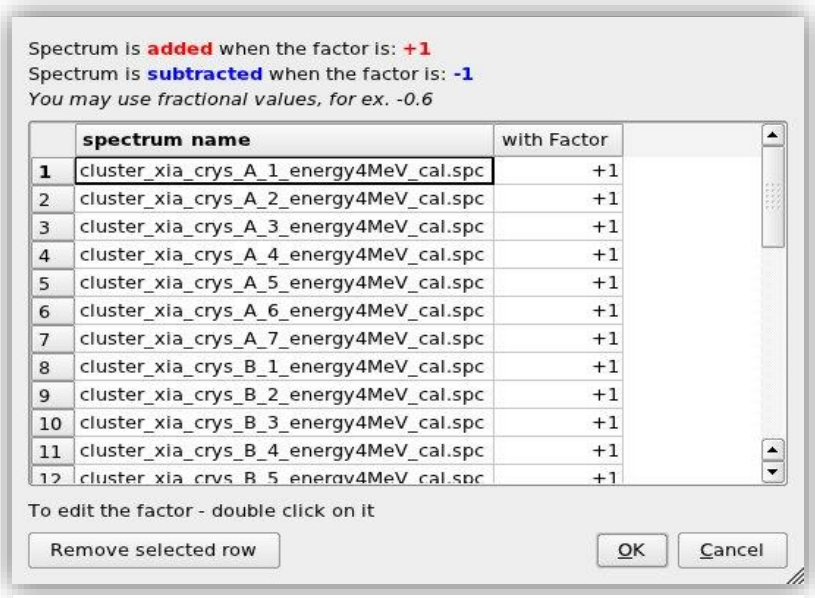


Figure 4. The tool for automatic (and cyclical) spectral summation available in the GREWARE GUI – allows creating the "total" summary spectrum. This spectrum is updated every few dozen seconds.

As such adding up is repeated by the GREWARE GUI every few dozen seconds – this total spectrum "grows" on its screen during the analysis.

However, such a solution is not convenient. Specially that the experimentalist needs a many other variants of such "total" spectra.

So, there is a better way to create the "total" spectrum – without having to create 105 intermediate spectra

> We just need to ask the spectrum wizard to create one spectrum that has many (105) incrementers on its list of incrementers.

These should be the same incrementers which we would use to create elementary spectra.

During the analysis of every single event, such spectrum has a chance to be incremented up to 105 times.

> *Of course, usually in a single event only a small part of germanium detectors register some γ quanta. Incrementers of the others carry the value zero.*

Look at the corresponding wizard page (Figure 5), where you can see a fragment of the list of incrementers of such a *total spectrum*.

ANCHOR 4.6

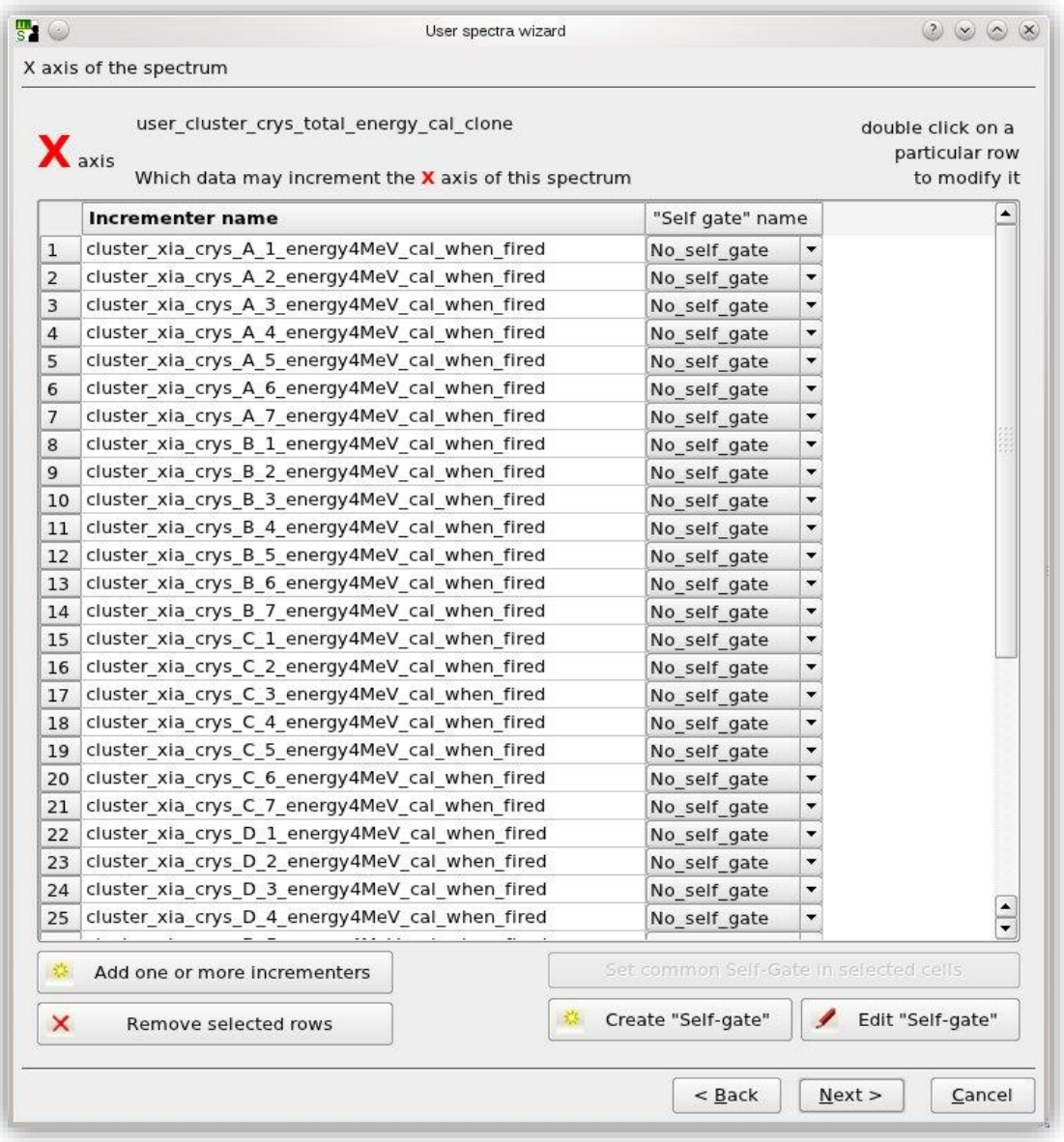


Figure 5. Page of the spectrum wizard which shows a list containing 105 incrementers of the spectrum "total gamma energy calibrated".

The next Figure 6 shows the result "Total" spectrum crated by such definition. This spectrum was created by the Spy – using incrementers presented above.

ANCHOR 8

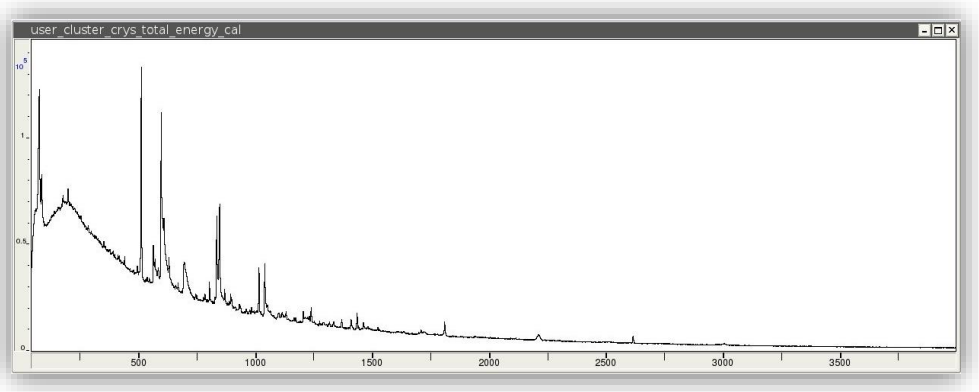


Figure 6. Spectrum of Energy of gamma quanta registered by all 105 germaniums crystals.

Soon we will see an even better way of constructing such “total” spectra (page 42, §6).

4.2. 3. Creating a two-dimensional spectrum definition (creating a matrix)

The wizard can also help us to create a two-dimensional spectrum definition. If we decided so (on the first page of the wizard), the wizard show the next page where we can choose the resolution and range of both matrix axes (X and Y).

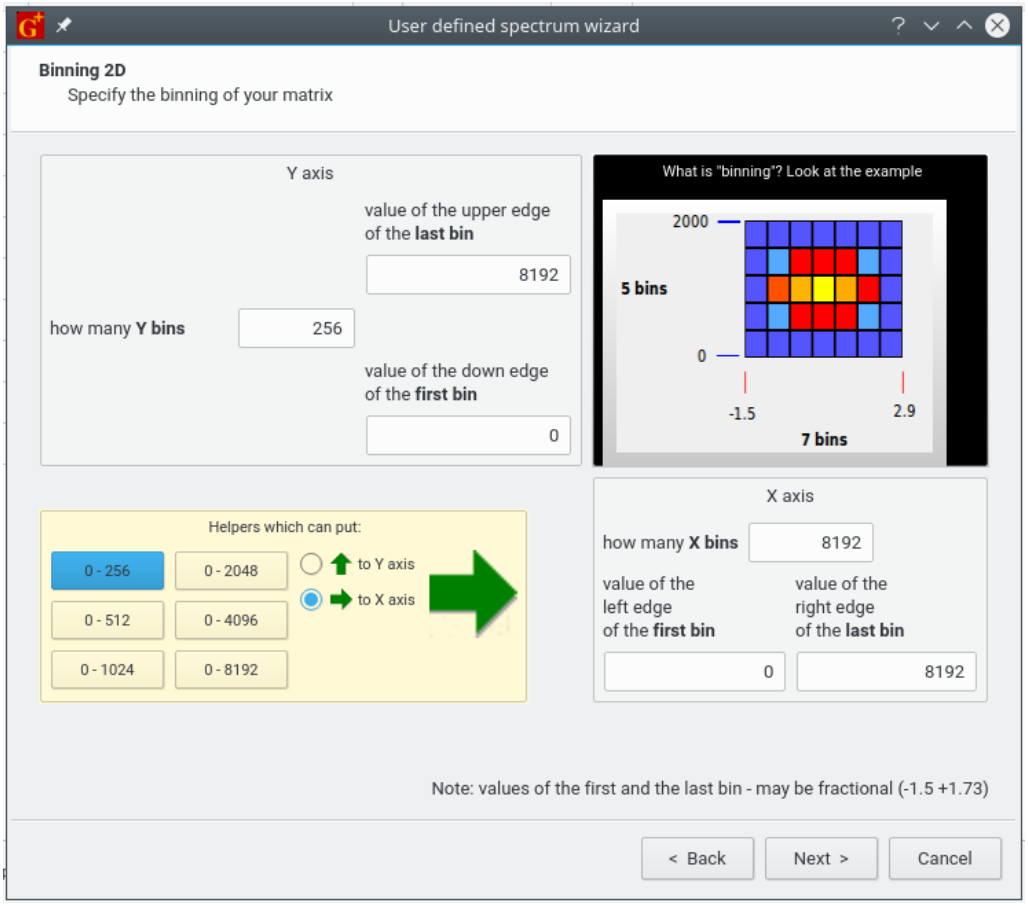


Figure 7

ANCHOR 4.8

What is an idea of incrementing of two-dimensional 2D spectrum?

The 2D spectrum (matrix) allows you to increment in the matrix points P (x, y), whose coordinates are provided by a pair of incrementers. One of them provides the value that becomes the x-coordinate, and the other y-coordinate of the P point. Therefore, when defining a two-dimensional spectrum, you should specify not only the incrementer for the X axis, but also incrementer for the Y axis. To make this possible, the wizard displays additional page, where we should place one (or several) incrementers providing data for the Y axis.

For example,

imagine that we want to create the matrix energy (E) versus time (T), of registered in particular detector.

It is simple to do:

- in the table with the list of Y-axis incrementers, we should put the **energy** incrementer
- on the list of X-axis incrementers we need to put the incrementer representing the **time**

As a result the Spy will produce a matrix for particular detector. We will see this procedure in the next paragraph .

4.2. 4. When we want a “total” 2D spectrum

Sooner or later, when the experimenter is sure that all detectors (of one particular type) are working correctly. So now he would like to see some ‘total spectrum’ (`total matrix`) based on data coming from all of them. Creating of total matrix definition is very simple. We just need to make a matrix, which has many incrementers working on X-axis, an many other working on Y-axis.

So there is a situation, there are *many entries* in both tables (X incrementers and Y incrementers)

What are the consequences of placing multiple incrementers on two-dimensional spectrum axes?

If there was just one incrementer on every lists – the matter is clear: after analyzing each event, the procedure will increment in this matrix the point P (x, y), where

x – is the current value of the X incrementer,
y – is the current value of the Y incrementer

In our case we have many incrementers on both axes.

There is an important issue here…

To makes things simpler, let us assume that there are not 105, but only 7 detectors. With this assumption, both lists of the spectrum incrementers schematically look like this:

List of incrementers *X*		List of incrementers *Y*
x_1		y_1
x_2		y_2
x_3		y_3
x_4		y_4
x_5		y_5
x_6		y_6
x_7		y_7

Which matrix points should be incremented in case of such lists?

There are three sensible interpretations – and they will be discussed below

You can choose one of these 3 option by marking your decision in the radio-button selection box (it is below the table with Y incrementers). See Figure 8 below.

ANCHOR 4.9

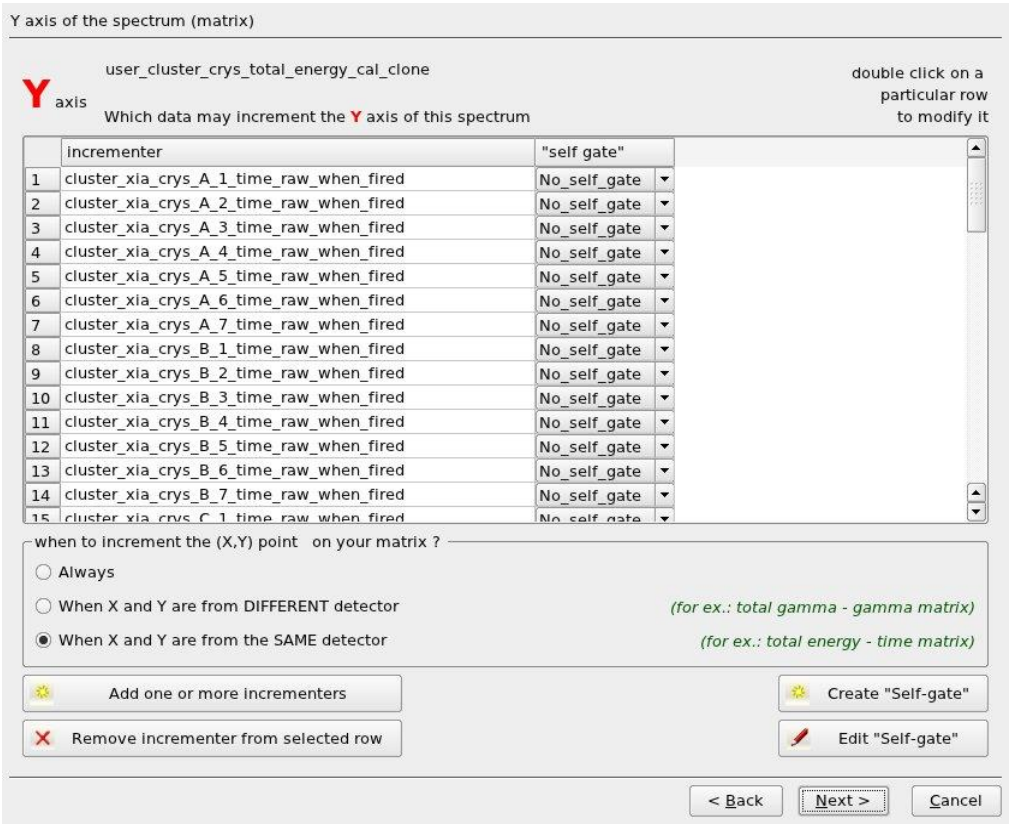


Figure 8. A page of wizard where we put our decision about Y-incrementers. Below the table, there is a set or "radio-buttons", which allow us to make a choice about: "When to increment the (X, Y) point on your matrix?"

The meaning of this triple choice is discussed in following paragraphs.

4.2. 5. All possible combinations of incrementers

This is an "everybody with everybody" variant. Points that have a chance to be incremented are:

$$P(x_1, y_1) \quad P(x_1, y_2) \quad \ldots \quad P(x_1, y_7)$$
$$P(x_2, y_1) \quad P(x_2, y_2) \quad \ldots \quad P(x_2, y_7)$$
$$\ldots$$
$$P(x_7, y_1) \quad P(x_7, y_2) \quad \ldots \quad P(x_7, y_7)$$

This is the most general approach, although rarely useful from the perspective of the physics of the experiment. But if you want this case, you should select the "Always" option.

4.2. 6. Combinations of incrementers belonging to the same detectors

This option is suitable, for example, for creating the E_γ versus T_γ matrix.

- On the list of incrementers X there are incrementers representing the **time** of recording γ quantum by crystals 1 – 7,
- while on the list Y there are incrementers representing their **energies** in these crystals 1 – 7.

> When preparing such matrix, the experimenter would like to increment the points whose coordinates relate to energy and time of the particular γ quantum registered by a some crystal.

There is no physical sense of incrementing a point whose coordinates are, for example, time recorded by the first crystal nr 2 and energy recorded by the crystal nr 5.

So, the desired combinations are:

$P(x_k, y_j)$ *where* x_k, y_j *are incrementers belonging to the same detector (i.e.* $k \equiv j$*).*

Note that does not necessarily mean combinations: “the **first row** of table X with the **first row** of table Y”. This approach would not be user-friendly, because it would require that the incrementers be placed in the tables in a strictly defined order. The system should not absorb experimenters with its syntax. Good news:

> The Spy can judge himself which combinations of incrementers form pairs belonging to the same detector (e.g. a germanium crystal) and will use such combinations.
> *This is possible due to the fact that the incrementer remembers the "address of the detector object where he was born and belongs" (“owner address”).*
> *It was described on page 13 (§3.2. 2).*

If you want such combinations of incrementers from both tables – you should choose the option:
"when X and Y are from the SAME detector".

Next to this option there is also a hint telling that this option is suitable for the energy vs time matrices.

The following Figure 9 shows an example of energy vs time spectra prepared in this way.

ANCHOR

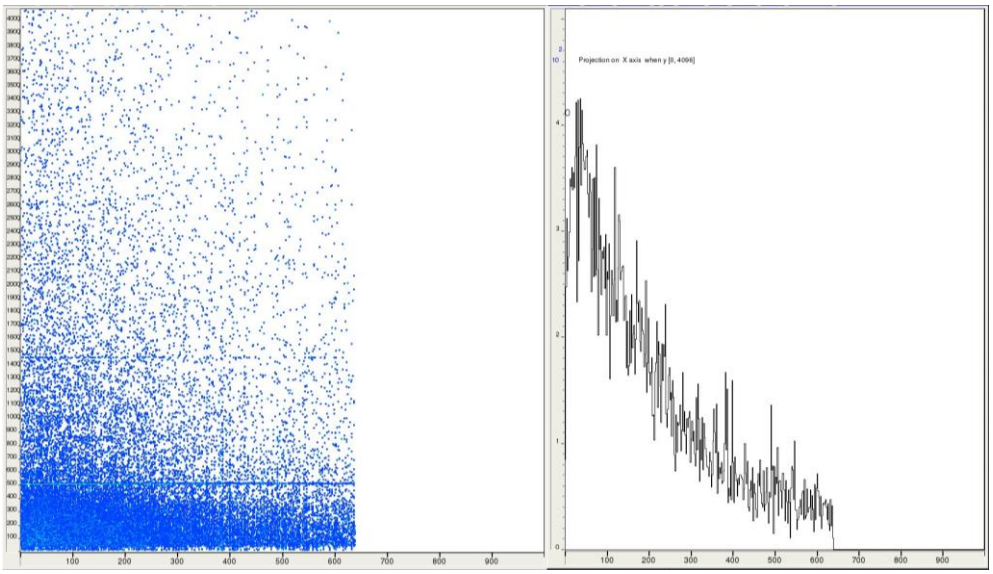


Figure 9. Left: – spectrum gamma energy versus gamma time. Having such a 2D spectrum we can make its projection on X-axis (seen on right hand side).

4.2. 7. Combinations of X and Y incrementers belonging to different detectors

In a situation when we want to create a γ–γ energy coincidence matrix we place the gamma energy incrementers on the X list of incrementers, and the same set of energy incrementers on the list Y.

Note, that we are interested in combinations of the energy of one γ quantum with the energy of **another** γ quantum.

There is no point in incrementing the point when both axes have the same incrementer. The coincidence of gamma quantum with the same gamma quantum – is a nonsense (“you cannot be in coincidence with yourself”).
Such “fake coincidence” would create a set of points on the diagonal in the spectrum – and we do not want this.

Thus, the desired combinations are:

P (x_k, y_j) where x_k, y_j are incrementers from different detectors (i.e. $k \neq j$).

As before – we do not have to take care of the order of incrementers in the tables.

The system itself can check, which incrementers come from the same detector (here: germanium crystal), – and can exclude such a combination.

If the experimenter is concerned with just such a selection of increment combinations from both tables – he should choose the option:

"when X and Y are from the DIFFERENT detector".

Next to this option you can also see a hint that it is suitable for coincidental Eγ – Eγ matrices.

Figure 10 below shows an example of a two-dimensional spectrum created in this way.

ANCHOR

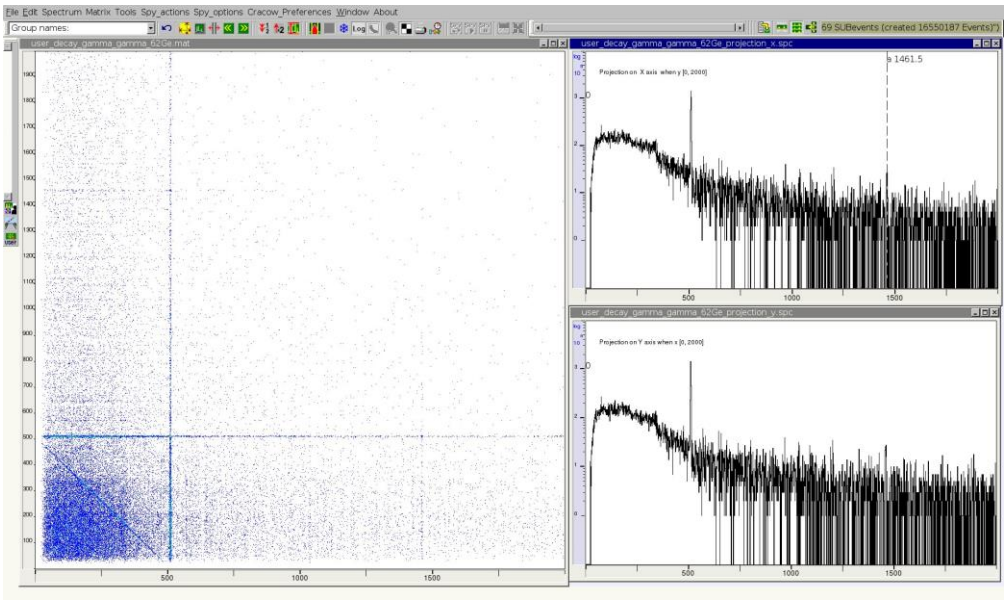


Figure 10. Left: – Created by the wizard – a symmetric matrix of γ–γ energies.
Right: two spectra which are projection of this matrix on X-axis and on Y-axis.

4. 3. What the Spy does with our spectrum description

In the previous paragraphs the spectrum wizard was presented. It allows the experimenter to create a definition of the spectrum. This definition is finally saved to disk.

Now the Spy needs to read it. Let's assume that the Spy is already working for some time. How will the Spy react?

During his normal work, the Spy program processes the events coming from the data acquisition system. However, from time to time it also checks (every few seconds) whether any commands from the GUI program have come to him. If so, the Spy suspends the event analysis loop, saves current results (i.e. the spectra produced and populated) and loads all the analysis parameters once again.

The new values loaded replace the existing ones.

Then the Spy program loads the wishes of the experimenter spectra (user spectra definitions) which are stored on a disk. Among them can be new spectra defined just now, but also these which we defined some days ago. The program loads all these definitions.

When the spy reads-in our definition of a spectrum, at first it checks whether such a spectrum is brand-new, or it is already exists in the program's memory.

- Of course – if the Spy program has been started just now, it is a first reading of the user spectrum list. So none of these user-spectra exists in the program yet. The Spy begins to **create** them one by one.
- If the Spy analysis program is running already for some time, it is possible that particular "definition" of the spectrum is not brand-new, but it was already read by the Spy. So such

spectrum exists in the memory of the Spy. Even more: this spectrum can be already partially filled with sensible data. In this case the Spy **does not need to create** this spectrum once more, because it is ready for use.

Severe modification of spectrum definition – is zeroing an existing spectrum

Sometimes the spy reads a definition of the spectrum called nnnn, which is already known to him, but – even so – the Spy decides to reset the old definition and to create this spectrum again.

This happens when there are severe differences in the new version of spectrum definition:

- the dimension of the spectrum has changed (for example it was 1D and now it is 2D spectrum) or…
- there are differences in binning or range of X or Y axis.

In such case the old spectrum is deleted and new ones are created, according to new parameters. The old content is therefore lost.

Change in incrementers list – will not zero a spectrum

Note that the identity of the spectrum definition applies only to the name, dimension, resolution and range of the spectrum, but…

> The identity of the increment lists is **not** checked. So when you change only the list of incrementers – the spectrum will not be automatically zeroed.
>
> It will work now using new incrementers.

This is not a big problem. In this situation you can always make manually a command “zeroing spectrum”.

5. How to clone the existing spectrum

We have 3 options of making a clone of existing spectrum:

a) simple cloning the user-defined spectrum,

b) making a new user-defined spectrum which is based on some chosen build-in spectrum *[NEW]*

c) smart multi-cloning of user defined spectrum – to create many variants of particular user-defined spectrum

In the following paragraphs we will talk about these 3 possibilities.

This is an interesting chapter, but if you are reading this manual for a first time, perhaps it is better to skip this chapter and go faster to the conditional spectra.
You can always return here in case if you really need to clone your spectra.

5. 1. Simple cloning of the user-defined spectrum

Simple cloning it is just making the new user-defined spectrum, which is exactly the same like another existing user-defined spectrum (cloning 1:1). The only difference is the new name.

To make such a clone spectrum is very simple: we open some existing definition of a user-defined spectrum and on the first page of the spectrum wizard we change name of this spectrum. Then we finish editing the spectrum.

By doing this the old definition will not be affected, the new definition will be saved under the new name.

> Note: if you press a button 'cloning', the spectrum editor wizard will allow you to change the name, but this procedure will also add a postfix '_clone' to the current name.
> *It is just to be sure, that you will not save this definition under the old name.*
>
> You may accept this postfix, but usually you delete it, because you want to have some more meaningful change of the name.

Producing two identic user-spectra is not a big thing. So, **when such cloning can be useful?**

In practice we do this cloning as a first step. Then immediately we go to the editing this new definition. During this editing process we modify something – like applying another condition, or using some other set of incrementers, etc.

5. 2. Cloning of a build-in spectrum *[NEW option!]*

Sometimes you would like to create a user-defined spectrum, which is just like some build-in spectrum (existing by default). There is a tool for making this.

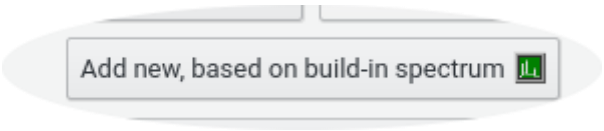
The newly created user-defined spectrum will have:

- a *similar* name (which you may change later),
- a same binning data,
- a same dimension,
- a same set of incrementers (if proper description of incrementer was provided)

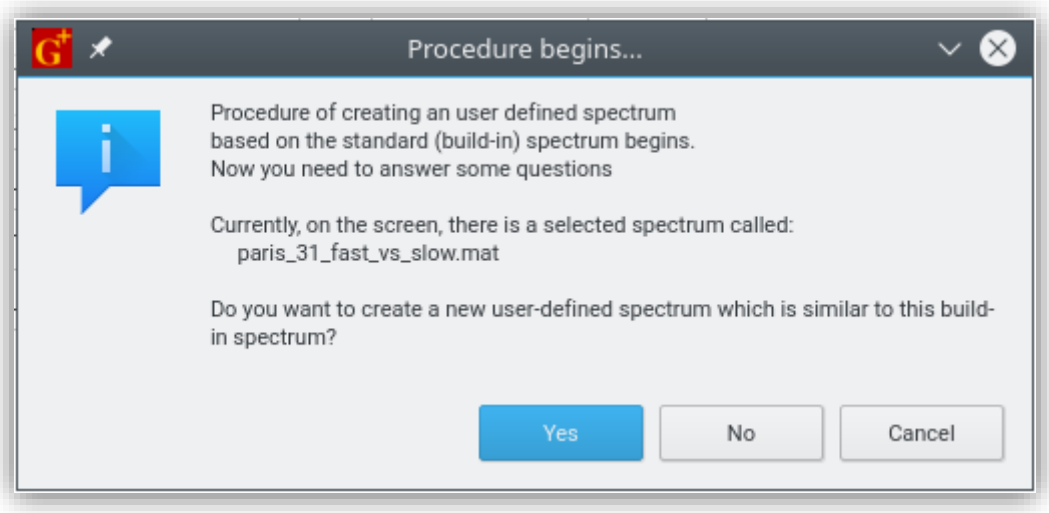
This last point is not always possible.

To make a clone of existing build-in spectrum make following steps:

1) Display this build-in spectrum graph on the screen,
   if there are also other spectra displayed on the screen – click on the chosen one to make it 'selected'.
2) Open the User Spectra Manager dialog box.
3) At the bottom of this dialog you will see a button called "New, based on build-in spectrum". Click this button.



Then a following information will appear



After doing this, the procedure of cloning will start.

- It will make some obvious things, just like recognizing the binning and dimension of the build-in spectrum.
- Then it will try to recognize which incrementers were used for this spectrum.

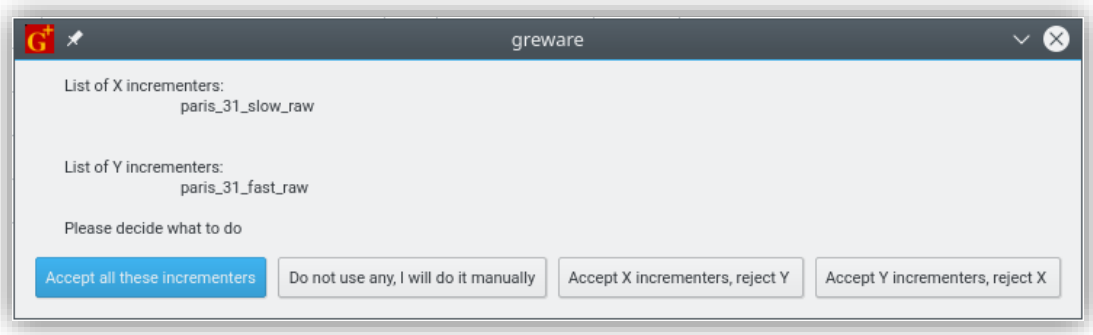
This recognizing may not always work. Why? Because information about the names of the incrementers is coming from some comment (description) which should be added to the build-in spectrum by the person, who wrote this section of the Spy.

This is the information which you can see if you press right mouse button while being on the graph of the spectrum.

The problem is, that so far providing this information (description of incrementers) was not obligatory for the Spy programmer. If he neglected to write this information – there will nothing to offer you.

Let's assume that description exists and is correct.

The cloning procedure will show you what was (in this 'description') the name of incrementer. It will also ask you if you accept this as an incrementer or not.



You may accept the incrementers or not. Whatever you do, remember that just after finishing this cloning procedure you will be transferred to the spectrum wizard. The wizard will give you a chance to edit this newly created (cloned) user-defined spectrum, so you will be able to make the final decisions manually.

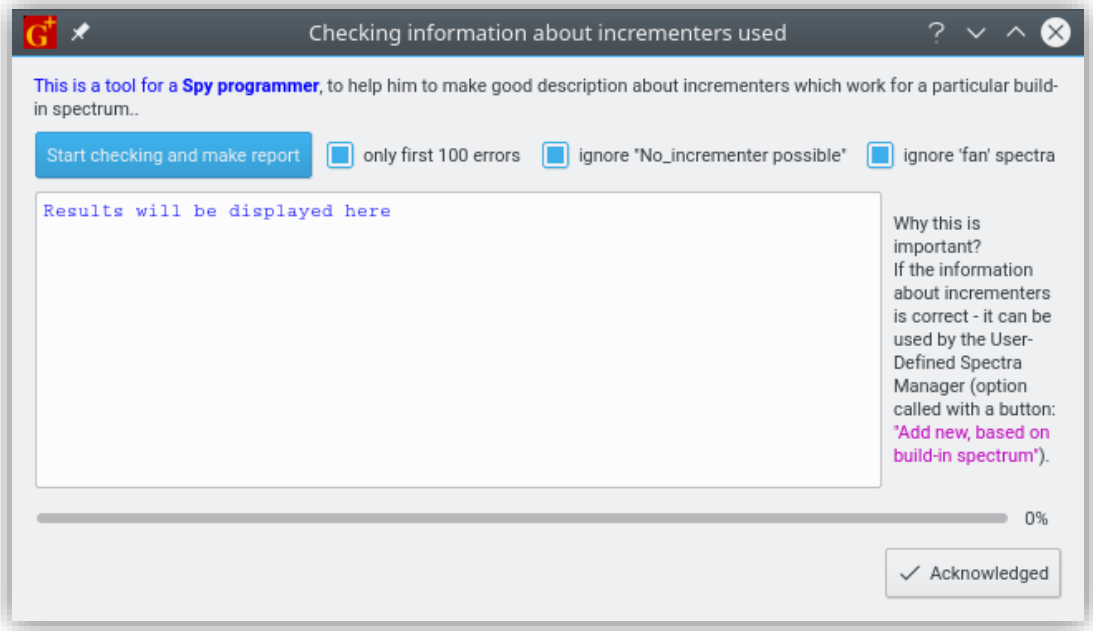
✧

A new tool for the Spy programmers

In most cases the information about the incrementer's names should be correct. If it is not correct, at first tell this to the person who programmed this part of the Spy program (Jurek?). He will be able to correct this.

> There is a special tool for the spy-programmers. It helps to look at all the build-in spectra and to check if their incrementer descriptions are correct.
> *If a incrementers description (of some build-in spectrum) is not correct, the helper program will suggest what could be most probable names of incrementers used in this case.*
> *The programmer can follow this suggestion and put the right information in the source code of the Spy.*

To run this tool from GREWARE GUI, go to Menu Bar → Tools → Make check of incrementers



5. 3. Smart multi-cloning

Here we are going to talk about the third option of cloning spectra. This will be a smart cloning procedure, which can produce many spectra from one (user-defined) spectrum.

✧

Imagine that you have an array of many similar detectors working together. For example the Galileo array, which consists of 40 germanium detectors. You just created the user defined spectrum dedicated to particular detector of this array. The name of this spectrum is:

user_ge_**01**_energy_raw

Inside the definition of this spectrum you asked that this spectrum is going to be incremented by an incrementer called

ge_**01**_energy_raw_when_fired.

You tried to analyze some data with and you are happy to see that spectrum collects he proper data.

Now you need to make the same spectra for other 39 detectors of Galileo array.

Of course you can start cloning this spectrum manually. At first you clone it to have the spectrum with a name:

user_ge_**02**_energy_raw

and inside the definition of this spectrum you replace the incrementer to the new one:

ge_**02**_energy_raw_when_fired

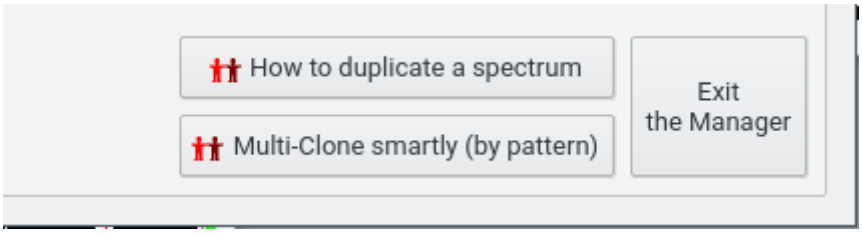
Then you must repeat the same procedure (of cloning and replacing) for detector 03, 04, 05, … , 40. Plenty of work.

You may ask: why the computer cannot make this multi-cloning for me?

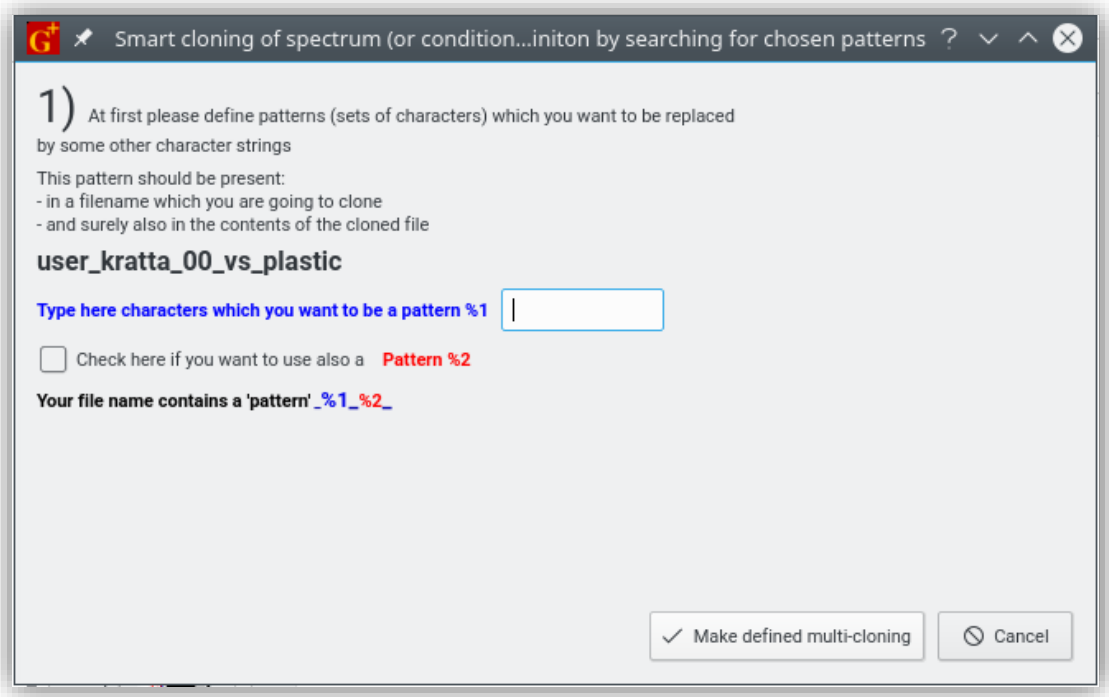
Yes, the GREWARE GUI can do this for you. For this, there is a special button in the User-defined Spectra manager.

What is an idea of smart multi-cloning?

At first select on the list of user-defined spectra (above) the spectrum definition which you would like to clone. Then press the button called “Multi-clone smartly by pattern”.



Then the procedure will start. The following dialog window will be open:



As you see, this procedure gives you the chance of choosing some pattern (some group of characters) inside the name of the spectrum definition, which you are going to clone. During the cloning process **your pattern will be replaced with some other string** of characters. By this a new spectrum definition will be created.

There is something even nicer. If you provide many version of "replacement" patterns, many variants of clone will be produced.

> What is important – the replacing of a pattern will occur not only inside the name of a spectrum. The replacement will also occur *inside* a definition of the spectrum, so also in a list of used incrementers.

This is what we were dreaming about.

To understand how this cloning works and how it is possible, let's look inside a text file with a spectrum description.

Below you will see an example of the spectrum description file. The interesting places are marked with numbers in black circles. They will be explained below.

```
// This is a definition of the conditional spectrum
// enables are 0 or 1 which means false or true
// comments are marked using two slashes: // comment

spectrum_name     user_zet_vs_aoq          ❶
enabled           1                        ❷
dimension         2                        ❸
//-------------------------------------------------------------------------
bins_x            500                      ❹
beg_x             2.2                      ❺ // left edge of the first bin
end_x             3                        ❻ // right edge of the last bin
bins_y            500                      ❼
beg_y             42                       ❽ // left edge of the first bin
end_y             57                       ❾ // right edge of the last bin
incrementers_x                             ❿ // X incrementers and their self_gates
```

```
{
    aoq_A_over_Q_when_ok        No_self_gate                ⓫
}                                                           ⓬
incrementers_y                          // Y incrementers and their self_gates
{
    zet__Z_when_ok              No_self_gate                ⓭
}
policy_when_increment   0               // 0 = always,      ⓮
                                        // 1 = only when from DIFFERENT detector,
                                        // 2 = only when from SAME detector
condition_name          104Sn.cnd                           ⓯
```

As you can see, there are keywords in the file. These keywords specify what is a meaning of the value which is directly following it.

❶ – name chosen for the spectrum

❷ – spectrum is enabled (so the spy will really create it and increment) [1– yes, 0 – no]

❸ – this is 2-dimensional spectrum (a matrix) [1 – 1D, 2 – 2D]

❹ – nr of bins (channels) on X axis is 500

❺ – left edge of a first channel is 2.2

❻ – right edge of the last channel is 3

❼ – nr of bins (channels) on Y axis is 500

❽ – left edge of a first channel is 42

❾ – right edge of the last channel is 57

❿ – a keyword announcing the block of X incrementers

Since it is possible that the spectrum is incremented by more than one incrementer (both in terms of incrementers x and incrementers y), so after the keywords incrementer_x and incrementer_y you can see curly brackets. These brackets open a list in which you see the names of the incrementers which you have chosen.

⓫ – an incrementer chosen for working for X-axis, here: zet__Z_when_ok

As it was mentioned – each name of the incrementer can be accompanied by the name of his local condition, the so-called: self-gate. The name of such a personal condition may be next to the name of the increment. If this condition is not provided – the text "No_self_gate" is placed there.

⓬ – a curly bracket closing the list of X-incrementers

⓭ – an incrementer chosen for working for Y-axis

⓮ – a choice for **policy** of allowed combinations of X and Y incrementers

(in case of one incrementer on each list – it has value "always"),

⓯ – name of the condition under which the incrementing will occur or not.

✧

Note: this description is saved on the disk, but it is not a spectrum. So far, this is only a recipe. According to this recipe, the Spy can produce the desired spectrum.

Now we can return to a…

5.3. 1. "Smart multi-cloning" of a spectrum definition – step by step

Imagine, that we have a spectrum:

user_ge_**01**_energy_raw

This spectrum is using the incrementer called:

ge_**01**_energy_raw_when_fired.

Let's look at the smart multi-cloning of this spectrum step by step.

1) Open the User-defined Spectra Manager.

2) Find a name of this spectrum (user_ge_01_energy_raw) on the list of available user-defined spectra, and select it (by clicking on it).
3) Click a button called "Multi-clone smartly by pattern".

The following dialog will appear.

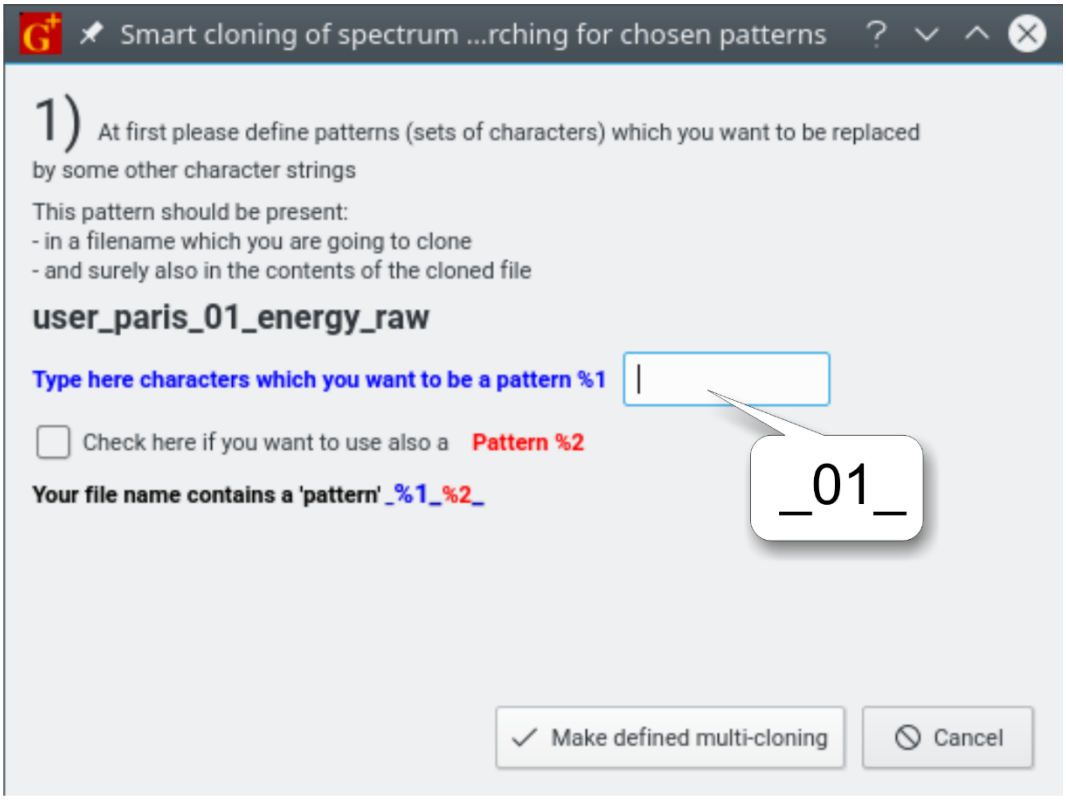


Figure 11 . Part of a multi-cloning dialog – just before we choose a patterns to be replaced.

4) In the left side of this dialog you can see the name of selected spectrum. Now we have to decide which characters in the name should be a pattern, which should be smartly replaced. In the edit box pattern1 you type: _01_
   Note these underscore characters. They will be very useful. (Later you will see why).
5) After typing : _01_ you will see the information, confirming that such a pattern was found in the name of the selected file user_ge_**01**_energy_raw

The dialog allows us also a chance of choosing another pattern in this name (pattern2), but to make thing simpler, we do not use this option now.

Look what was result of your typing pattern1 so far. The following new parts of the dialog appeared:

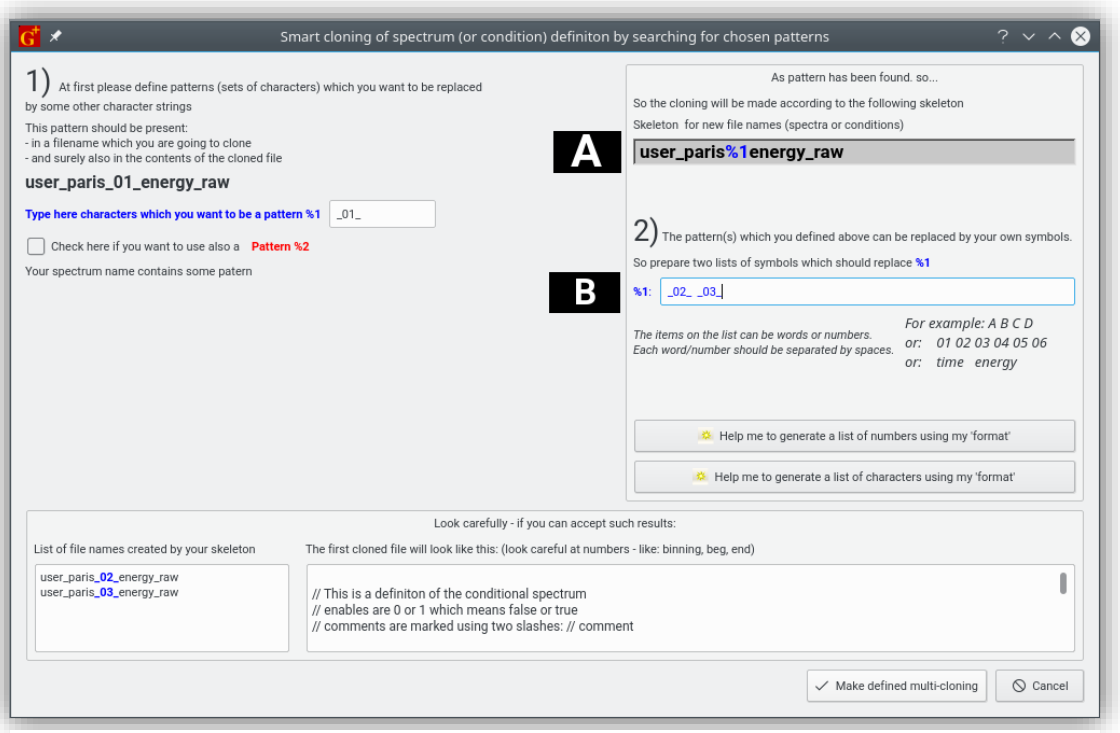


Figure 12. Smart multi-cloning dialog window

Let's look at the details.

Here, on the top A, you can see the skeleton of the name of the spectrum. The places where the pattern1 was found are marked as %1 and are printed in blue:

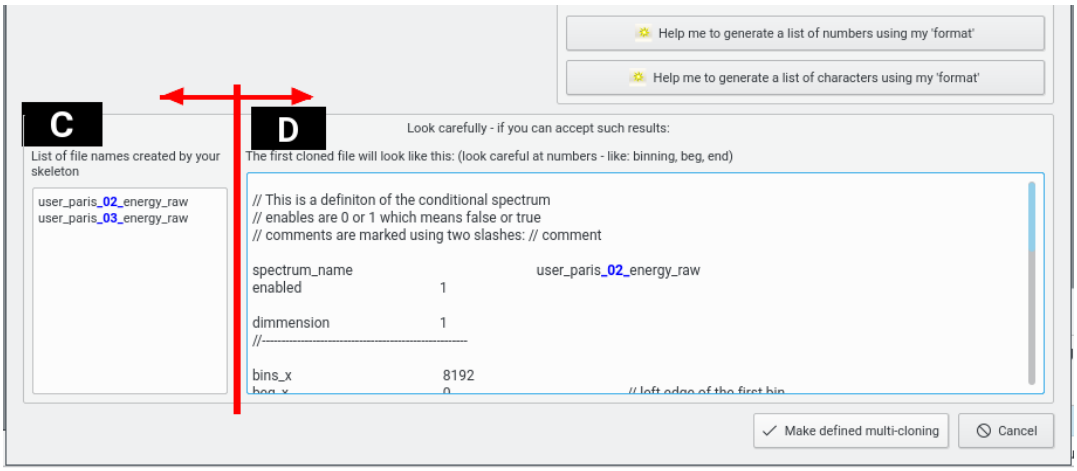
user_ge%1energy_raw

Below this text you can see a field B, where you can put one (or many) strings which will be used for making different variants of clones.

4) Please type (in this field) two following strings (separated by spaces): _02_ _03_

The smart-clone procedure will inform us that it is ready to create two clones of spectra definition. The names of these suggested new clones are displayed at the bottom of the dialog box C, on the left side.

Figure 13. Part of the multi-cloning dialog



Hint: if the frame with names of cloned spectra is too narrow (so names are wrapped)– you may resize the frame. To do this just click at the gap (between this frame and a frame on the right) and drag it to the right. The frame will be resized so you will read the names easier.

In the frame with filenames C ,you should see the following spectra names:

user_ge_**02**_energy_raw
user_ge_**03**_energy_raw

In the frame on the right D you can see the example of the first cloned file. Do you recognize this text? Sure, we were discussing the syntax of such a file in the previous paragraph. In color you can see the places where the pattern1 was found in the text of the spectrum description. You can see here, that replacement of pattern1 was also made in the name of the incrementer.

Unveiling the secret: Why the pattern with underscores: "_01_" is better than just: "01"

Now you are able to understand the fact that as a pattern1 we have chosen _01_ not just 01 (so: without underscore characters).

> Thanks to these two underscores we are sure that such a pattern will not be incidentally found inside one of numeric data representing the binning, or range of the spectrum.

For example let's imagine that we have chosen pattern as "naked" 01. Imagine also, that in this particular spectrum description the binning data has a numeric value 101. This can make problems, because the replacing procedure would find this number 101 and understood that these last two digits of this number (1**01**) are a pattern to be replaced with 02, then 03. This could be a nonsense, if we do not want to make any change in *binning* data.

This is where our trick with underscores – is going to help us.

By defining the pattern wider – together with these two underscore characters – we are sure, that such a pattern will not be found in numbers representing some other numeric data (binning).

Time to finish preparation of our cloning procedure:

5) Look at the list on names of all new spectra, which you are going to produce.
6) Look at the contents of the spectrum description – checking if there are changes in places, where you really need.
7) Press the button "Make defined multi-cloning" and all the spectra will be produced (their names will appear on the list of the User-defined Spectra Manager).

Ooops…

If you made some nonsense – you may easily remove the wrong definitions of spectra. You can also start the smart-cloning procedure again, the old products will be overwritten by new products.

5.3. 2. Helper which can produce a 'numeric' sequence text

We just learn how to make smart-cloning of spectra. Now you will see, that in the Smart-cloning dialog – there are some tools, which make your work even more pleasant.

Let's concentrate on a text field with label %1. Here we can type strings which should replace the pattern1, which can be used during a process of producing many versions (smart clones) of the original (cloned) spectrum.

In this text field %1 we typed two strings: _02_ _03_

Question: if we want to make clones for all 40 detectors – should we place here strings:

_02_ _03_ _04_ _05_ _06_ _07_ _08_ _09_ *and so on till* _37_ _38_ _39_ _40_

The answer is "yes".

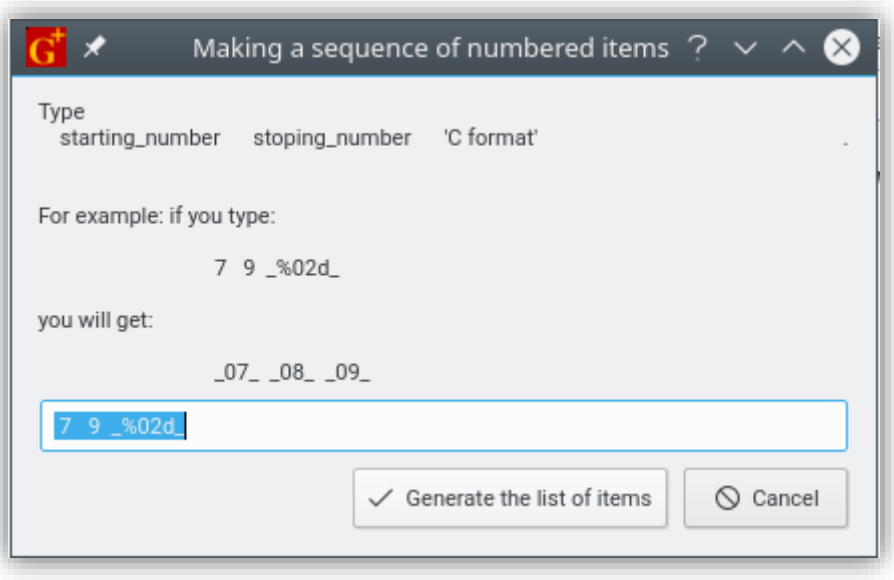
You may ask: *So much to type?*

Do not worry. There is a small tool, which will help you to produce such "replacement texts".

You can start this helper by clicking a button called "Help me to generate a list of numbers using my format"

By this click you will open a small dialog window.

This helper dialog can generate the sequence of numbers in a format which you would like.



All what you have to do – it to provide him some information:

- a starting number for a first item of your sequence,
- a final number of a last item of your sequence.
- a format skeleton of every item in the sequence.

In case of our example:

a) the starting number would be 2,

b) final number 40.

So we need to type this two numbers separated by space.

Now we need to type a format which you like. This is C-language format, which was used by the printf function (stdio library).

c) In our case we would like, that every item of our sequence looks in this style: _06_.
So we type a desired format as:

_%02d_

Here is the meaning of this format:

Every item should start with underscore character and should finish with another one.
Then it should be a integer number (%d), but we want it to have (always) two digits (%2d), even if the number in less than 10, so has one digit only. In this case in front we want a "filling character", which should be zero. (%02d)

Let's try

To make our example result shorter, let's imagine, that we want to produce a sequence of numbers from 2 to 11.

- We need to type (in one line) all these three information: 2 11 _%02d _
- Now we press "Generate the list of items" button. This will produce such result:
  _02_ _03_ _04_ _05_ _06_ _07_ _08_ _09_ _10_ _11_
- If this is a sequence which you wanted – just press a button "OK, copy to the clipboard". By this the generated sequence will be automatically copied to the clipboard.
- Point a cursor at the %1 text field and make "paste" (Ctrl-V). Your sequence should appear here.

Time to see if this is what we really need.

- Look at the bottom of the dialog window. In the left frame C you can see names of files which will be result of your smart multi-cloning procedure. Is it what you expected? If not – come back to the field %1 and make another choice for the replacement text.
  Or maybe even you should change a little the definition of the pattern1?
- If you are happy with names of files, look carefully at the frame on right-hand side D. Here you see what was replaced inside the file with spectrum description. (Changes are displayed in color).

5.3. 3. Helper which can produce an 'alphabetic' sequence

Sometimes detectors do not have numbers, but letters. So a name of the corresponding user spectrum can look like this:

user_rising_**A**_energy_raw.spc

If we want multi-clone this spectrum – the pattern1 which we want to replace is _A_. We want to replace it not with numbers, but with characters. For example we would like to have spectra

user_rising_**B**_energy_raw.spc
user_rising_**C**_energy_raw.spc
user_rising_**D**_energy_raw.spc
user_rising_**E**_energy_raw.spc
…

No problem, in this case we need to have such replacement sequence in our %1 text field:

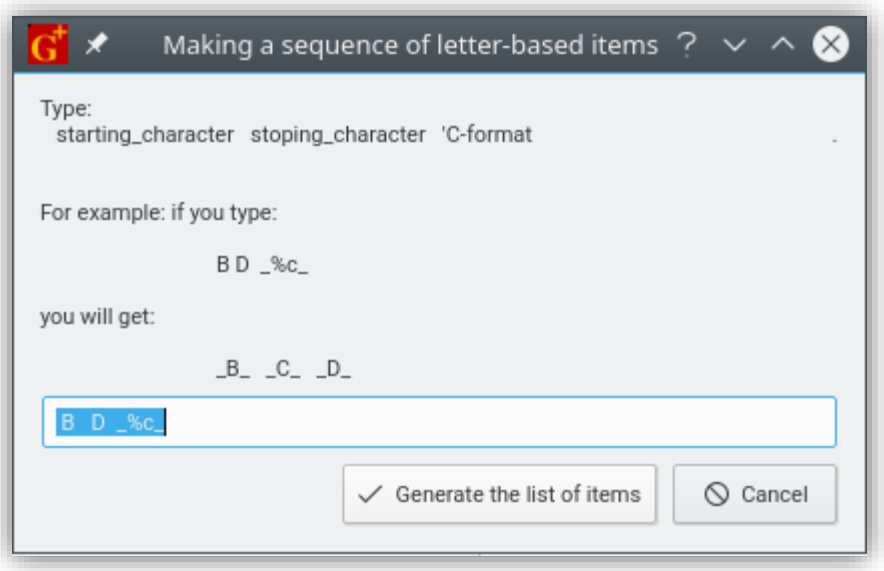
_B_ _C_ _D_ _E_

If you think, that your sequence is too long to type, you can use another helper which is specialized on generating sequence based on changing characters. Just press the button:

"Help me to generate a list of characters using my format"

By this you will open a small dialog window.

This helper dialog can generate the sequence of characters in a format which you would like.



All what you have to do – it to provide some information:

- first character of your sequence,
- final number of your sequence.
  "first" and "final" are in alphabetic order;
- format of every single member of the sequence.

So to get a sequence mentioned above it is enough to type:

B E _%c_

When you press a button "Generate the list of items" the sequence will be created. You will see it in a new message box.

Now, if you can press a "OK, copy to the clipboard" button, the generated sequence will be copied to the clipboard.

5. 4. The multi-clone procedure using two patterns

Sometimes your detectors (with names A B C…) have also sub-elements called 0, 1, 2…

Let's imagine, that you already have a user-defined spectrum called:

user_rising_**A_00**_energy_raw.spc

This spectrum (in its definition) is using a incrementer called:

rising_**A_00**_energy_raw

Now imagine, that you would like to multi-clone this spectrum to produce following spectra:

user_rising_**A_00**_energy_raw.spc
user_rising_**A_01**_energy_raw.spc
user_rising_**A_02**_energy_raw.spc

user_rising_**B_00**_energy_raw.spc
user_rising_**B_01**_energy_raw.spc
user_rising_**B_02**_energy_raw.spc

user_rising_**C_00**_energy_raw.spc
user_rising_**C_01**_energy_raw.spc
user_rising_**C_02**_energy_raw.spc

To make such multi-cloning possible we need two patterns which we are going to replace:

- pattern1 _A_ which will be replaced by _A_ _B_ _C_
- pattern2 _00_ which will be replaced by _00_ _01_ _02_

This is possible to do in our multi-clone thanks to a possibility of using two patterns.

What will be a system of replacing? If you know a little about programming you will easily understand a following analogy:

The replacing will be applied just as if it were: **two nested** loops

```
for( sequential replacement of the pattern1 by: (_A_, _B_, _C_) )
{
    for(sequential replacement of the pattern 2 by: (_00_, _01_, _02_) )
    {
            // action of replacing patterns in: using_rising_%1_%2_energy _raw
    }
}
```

This was an idea.

How to define such multi-cloning practically?

1) In the User-defined Spectra Manager – select a user-defined spectrum which you would like to clone (just click on it).
   Here we assume that it is a spectrum called:
   user_rising_A_00_energy_raw.spc.
2) Go to the multi-cloning dialog window (by pressing button: "Multi-clone smartly by pattern"). The new dialog window will appear.
3) Find a text field pattern1 and type there: _A_
4) Click on the checkbox below – marking that you need to work also with a pattern2
5) In the text field pattern2 type: _00_

Now you should see a text informing that both patterns were found in the file-name.

On the right-hand side of the dialog window a new appropriate information will be displayed. Here you need to define what items should replace the designated patterns.

6) Find A the replacement text field %1 and type there a following sequence: _A_ _B_ _C_ (Note: the three items of the sequence should be separated by spaces)
   If you have to type not just 3 items, but many of them – it may be good to use the provided helper (§5.3. 3, page 39).
   *As you need here a "sequence of **characters" –** press a button called ""Help me to generate characters using my format"*
   This will start a procedure (a helper) which will help you to generate desired sequence. The result you will paste into the %1 field.
7) In the replacement text field %2 (B)you need to type following strings (separated by spaces): _00_ _01_ _02_
   If you have to type many strings – it may be good to use the provided helper (§5.3. 2, page 37).
   *As you need sequence of **numbers** you need to press a button called "Help me to generate numbers using my format".*
   With this helper you will generate desired sequence, which you can paste to the %2 field).
8) After doing this, it is time to look at the preview of results. Look at the bottom of the dialog window. Check if the proposed names (C) of the cloned spectra are correct.
9) Look also at the content of the cloned definition (D). All changes are in color.

If this is what you wanted – press a button "Make defined multi-cloning"

If not –go back to the patterns and to the replacement sequences and change them.

✧

Here, in this example, we defined such replacement sequences that cloning will overwrite the original spectrum definition. Do not worry. The procedure is intelligent enough and will ask you if really you want to overwrite the original.

6. “Total” spectra made easy using the collective incrementers

Motivation

While defining a 1D spectrum, the user can decide that this spectrum should be powered by more than one incrementer.

In other words: during the analysis, after analyzing a particular event, this spectrum has a chance to be incremented more than once.

For example in the RISING experiment 105 germanium crystals were detecting γ quanta. The experimenter often wants to watch the spectrum which is a **sum** of 105 energy spectra produced by individual crystals. Such a spectrum we call "**total** γ energy spectrum". The user can define such spectrum himself. He just need to place on the list of spectrum incrementers all the 105 incrementers representing the energies of γ-quanta registered in particular crystals. That’s all.

Similarly, "total time" or "total_scattering_angle" spectra can be created.

The problem is solved, but… juggling 105 incrementers in the spectrum definition is very inconvenient.

Solution

This lead to the concept of a *meta-incrementer*.

> We call it a **collective incrementer.** This is an incrementer which looks like a regular incrementer, but it is actually kind of a list of other (ordinary) incrementers. Its name usually starts with „ALL_”.

The collective incrementers are prepared by the programmer who wrote the Spy program, and (by default) they are available to use.

Here is an example. The following list of 105 incrementers:

rising_A_1_energy_cal
rising_A_1_energy_cal
...
rising_R_7_energy_cal

is available also as one collective incrementer called:

ALL_rising_energy_cal

When it is useful?

The collective incrementers are used to collect a large group of incrementers of the same physical meaning (belonging to the array of similar detectors).

In the Spy program you have a plenty of collective incrementers representing other data, e.g. about the time or event an angle (between the quantum direction γ and the direction of the recoil nucleus).

Collective incrementers used when defining a matrix (two-dimensional spectrum).

Let’s imagine that we need a matrix: total “γ–energy versus γ –time“.

In this case, it is enough to put (in the table with the list of X-incrementers) the collective incrementer called:

ALL_rising_time_cal_when_good

and in the table with the list of Y-incrementers the collective incrementer:

ALL_rising_energy_cal_when_good

Although both tables contain only one row – just a collective incrementer, we have to remember, that these are the **collective** incrementers. So each of them is like a list of 105 simple incrementers. Therefore, in the selection field of desired combinations of incrementers, we need to select the appropriate option.

> *In our case "total γ–energy versus γ –time" matrix , we need to choose an option "when X and Y are from the **SAME** detector"*
> *(more in §4.2. 6, page 25).*

The incrementing procedure is intelligent enough that it can correctly judge whether the (simple) components of the X and Y collective incrementers are from a particular germanium crystal or from different crystals.

On the following Figure 14 you can see an example of a matrix produced using some collective incrementers.

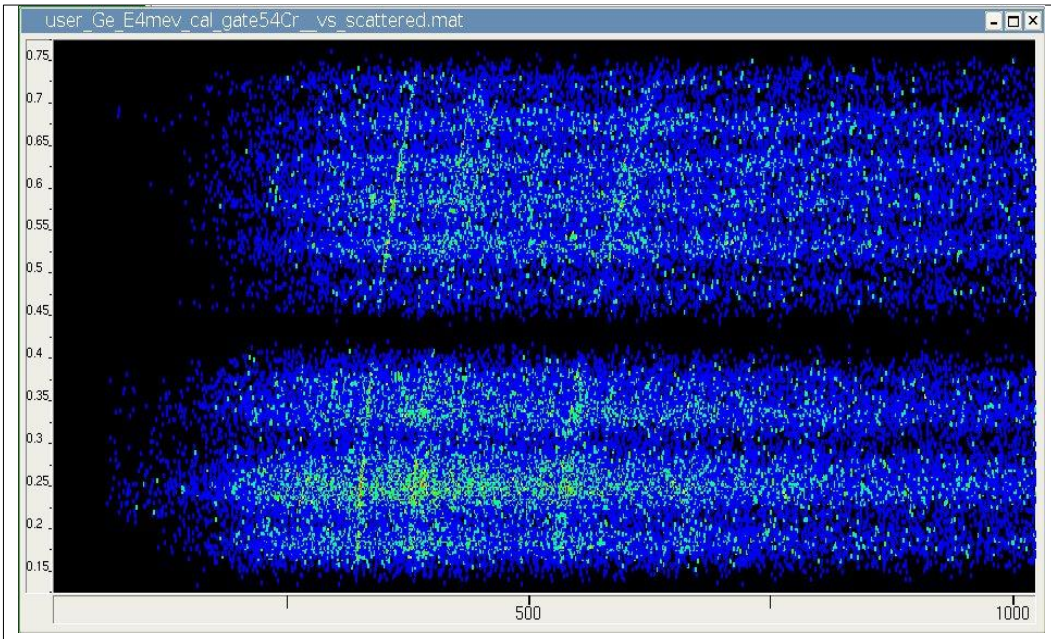


Figure 14. Two dimensional spectrum (matrix) created using two collective incrementers. It depicts relation between scattering angle of ion (Y), versus gamma quantum energy emitted by this ion (X).

The collective incrementer used on X axis was

ALL_rising_energy4MeV_cal_when_good,
and the collective incrementer on Y-axis was

ALL_rising_theta_radians_between_gamma_and_scattered_particle.

Applied policy option was: "when X and Y are from the same detector".

As the nucleus, which emitted γ-quanta was moving, the measured energies were affected by the Doppler effect.

This is why on this graph you can see that quanta, which in case of scattering angle close to zero were measured as 511 keV, in case of scattering angle 0.45 radians were measured as ca. 560 keV and in case angle 0.75 radians ca. 625 keV.

KOTWICA

There is nothing difficult in using collective incrementers. Just remember that is a list of elementary simple incrementers.

7. Conditions defined by the experimenter

We have already learned how to create our new user-defined spectrum definition which uses chosen incrementers.

> Let's put it clear: if it was only about such spectra – most likely all this work would be unnecessary. The Spy analysis program is already equipped with these spectra as build-in standard.

No wonder: the incrementers are made of interesting, important variables from the analysis program, so – since the variable is really interesting – it is obvious that in the Spy program defines a corresponding build-in spectrum.

> *For example: in the program there is a variable representing the calibrated energy value of γ quanta. Such a variable is a good candidate for an incrementer. There is no point in letting the experimenter create his spectrum using this incrementer, because such a spectrum is already available as build-in spectrum.*

So what for are these incrementers?

Generally speaking – an experimenter usually wants to see a spectrum under specific **conditions** (set on some physical data).

> *In case of experiments with the RISING spectrometer, most desired condition was a requirement that a registered γ quantum is associated with a particular isotope provided by the FRS fragment separator.*
> *The gamma energy spectrum incremented **without** this condition – will show an energy spectra of all detected gamma quanta. Such spectrum is worthless, because it shows mainly a background radiation. So it can only be used to monitor whether a given germanium detector is working properly or not.*

To get spectra with really interesting physical content – we should increment these spectra under certain conditions. Conditions specific to a particular experiment.

This chapter presents how such conditions can be created by the experimenter.

Conditional spectrum or just a condition?

In the GREWARE analysis a condition is a separate object. Once a condition is created, many spectra can *use* it, referring to its name. This solution makes analysis process faster.

> During the analysis, after analyzing a particular event, the program at first evaluates the condition. From now on, the state of this condition is true/false (fulfilled/not fulfilled). Before incrementing a conditional spectrum, the program checks the value of the condition assigned to the spectrum. If it is "true" – the spectrum is incremented.

To sum up, we can say that in the described approach to data analysis we have:

- separate objects representing spectra defined by the experimenter,
- separate objects representing the conditions he defines.

Their mutual relations can be shown in the figure (Figure 15).

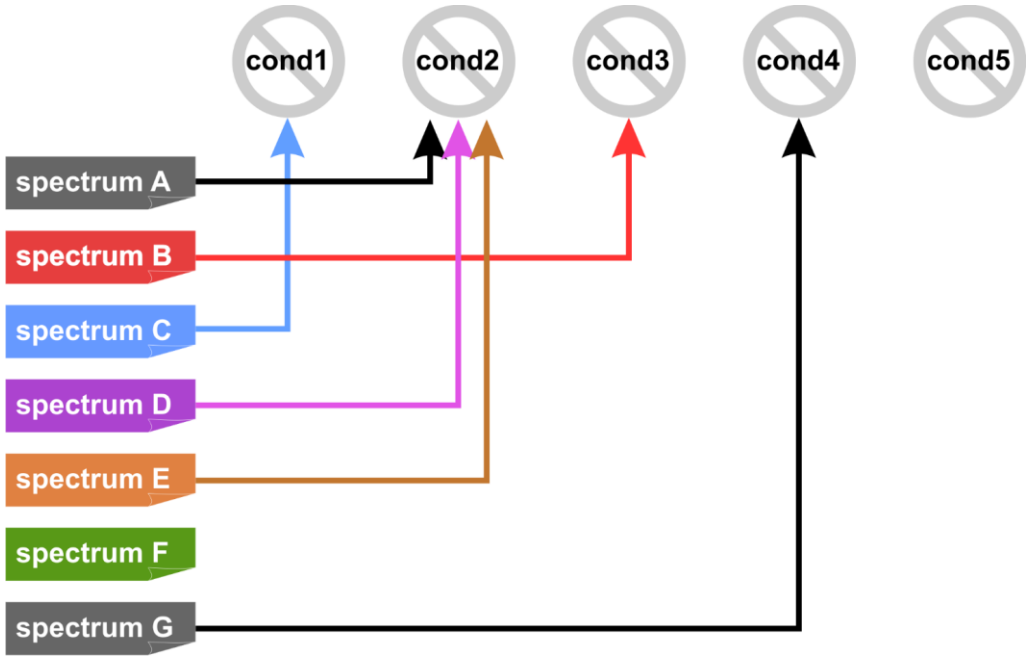


Figure 15. Spectra and conditions are separate objects. One condition can be used by many spectra.

The picture shows many spectra and many conditions. The spectrum (during its definition) may be informed that it can be incremented if a condition (with a specific name) is true.

Note:

- One condition can be used by many spectra (cond2).
- There may be some conditions that no spectrum uses (cond5).
- There may be spectra that do not use any condition (spectrum F).

There is only one limitation: the spectrum can point to only one condition…, but this is not a problem.

> *As you will see later – you can create conditions that use a logic of other conditions. This allows you to create even very complex logical expressions of the conditions.*

As you can see, the spectra and conditions are separate objects and a condition can be assigned to spectrum. While analyzing events, the program first evaluates conditions values (i.e. checks their fulfillment). When this process is completed, the spectra are incremented (or not) depending on the current logical value of "their" conditions.

7. 1. Basic conditions

To make the problem of conditions easer, let's divide it into simple situations. There are two types of elemental conditions:

- basic one-dimensional condition (also called a **gate**),
- basic two-dimensional condition (also called **polygonal gate**).

A basic one-dimensional condition (a gate)

It is described by two real numbers a, b which define the range [a, b] – in which should be a value of the chosen incrementer *z*.

> Evaluation of this condition means checking the following logical expression:
>
> $$(z \geq a) \wedge (z \leq b)$$

The basic 1D condition is useful when the experimenter wants to set a simple gate on one of the variables from the analysis program.

For example, to check if the energy registered by the germanium detector is within [500 – 580]

A basic two-dimensional condition

On a given two-dimensional spectrum – (which is currently displayed on the screen by GREWARE GUI) – the experimenter can draw a polygon. This polygon can be used to create a 2D condition.

> Evaluating this kind of condition means to determine whether (in a current event) the incrementers used by this matrix have such values (x and y) – that the point P (x, y) on the matrix is inside the selected polygon.

For historical reasons, often these types of conditions are colloquially called **banana gates**.

7.1. 1. Condition as a combination of several expressions from basic conditions

Out of basic conditions 1D and 2D – one could build any complex condition. However, such an basic level – would be too basic for the experimenter. Building the conditions that are during your experiments would be a very tedious job. Therefore…

> …the conditions that are available in the GREWARE can be a *combination* of basic conditions.

Don't worry, you do not have to use **all** possibilities listed below.

A GREWARE condition can be a combination of the following eight expressions:

- an alternative of any number of basic 1D conditions,
- conjunction of any number of basic 1D conditions,
- an alternative to any number of basic 2D conditions,
- conjunction of any number of basic 2D conditions,

and even expressions which are built on other conditions:

- conjunction of any number of other conditions,
- an alternative of any number of other conditions,
- negation of the alternative (NOR) of any number of other conditions,
- conjunction negation (NAND) of any number of other conditions.

The last four possibilities allow the creation of logical expressions from other, previously defined conditions. Because up to 1000 levels of such nesting are possible – so the experimenter can build even the most complex logic of conditions.

This was a theory. Now we will see how we handle conditions in practice. There are helpful tools to do this.

7. 2. Conditions manager

To create a condition in the GREWARE GUI program – you should launch a procedure called the Conditions Manager. Look at its dialog window (Figure 16).

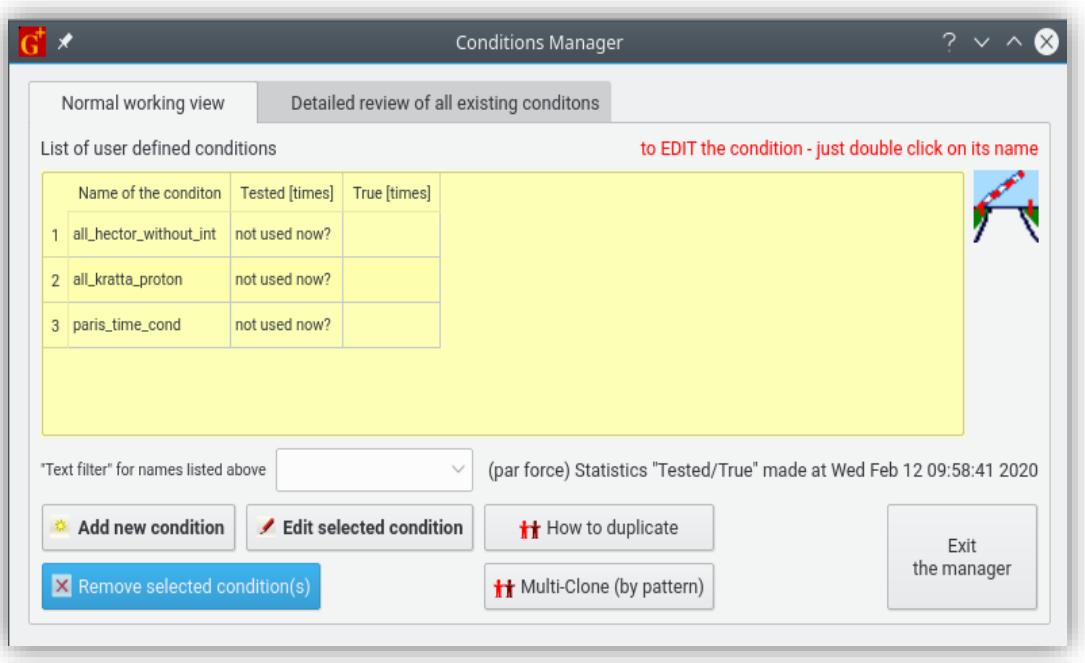


Figure 16. Conditions manager view. It shows a fragment of the list of currently defined conditions. Next to the names of conditions is the statistics for checking the conditions. (How many times it was checked and how many times it was true). At the bottom you can also see the buttons which start procedures for creating a new condition or modifying an existing condition.

As you can see, the Condition manager allows you :

- to view a list of the definitions of all existing conditions,
- to **create new** conditions or **modify** existing ones,
- to **create clones** of existing conditions
- to see statistics on the fulfillment of conditions.

The conditions manager window has a button "Add a new condition" that starts the procedure for creating a new condition. This procedure is runs a so-called "condition wizard".

7. 3. Condition Wizard

The condition creator helps the experimenter in creating a condition. Step by step, the wizard will lead you from page to page, where you can build individual parts of the new condition.

> *For example, there is a page on which you can build an alternative of basic 1D conditions (gates), on another page a conjunction of such gates, on another alternative an alternative to two-dimensional gates, etc.*

You do not have to use each of these possibilities, the page left empty – is treated as non-existent. Ultimately, the logical value of the entire condition will be calculated as a conjunction of those tables (pages) in which the experimenter has placed some items.

Persistence of the condition and possible cloning

The condition is a separate object. To refer to it – the condition must have its name. Under this name, the definition of the condition will be saved to disk. Each time the Spy (event analysis program) is started, it will load the definition of the condition. So as you can see – the condition definition is permanent. In the analysis program, you can use conditions that were invented during previous experiments many months ago.

You can start the wizard to make a new condition

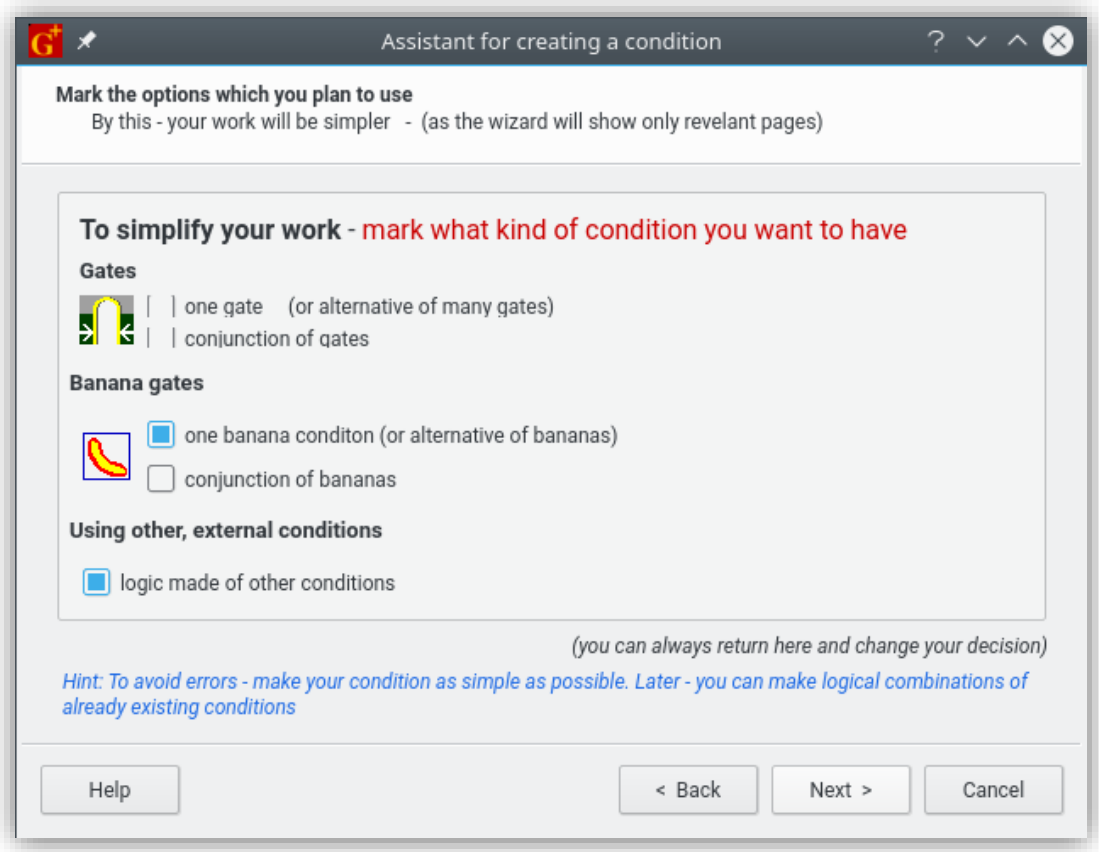
– by pressing a button: "Add new condition"

> Note: The definition of a new condition can also be created from another existing definition. This "cloning" of the condition is used when creating a condition similar to an existing one. The cloning gives a 1:1 copy, but usually this copy is immediately modified.
> *Remember to rename the "cloned" condition – otherwise the modified definition will replace the current one.*

Page 1. So, on the first page of the condition wizard you select the name of the condition

Then appears a next page.

Page 2. What kind of condition you like to have this time?



On the second page you can tell the wizard that you want to simplify the next conversation. You do this by marking what kind of condition you are interested in.

You can see here the places, where you will check/uncheck the possibility which you would like to use. Knowing this – the wizard will show you only relevant pages.

You can always return to this page of the wizard and change your decision. Your choice made here is not part of the condition. It is just a hint for the wizard to not to bother you too much with not relevant questions.

A good advice:

> You can create even a very complex condition using all possible pages of the condition wizard – but don't do this!
>
> It is better to make many simple conditions at first, and then to combine them into one bigger – by using last page of the wizard: "condition of other conditions".
>
> By doing so, you will easier find possible errors: "*why my condition in never true*", or "*why it does not work as expected*".

So take this advice: **in one condition – use just one of the following pages of the wizard**.

Now let's look at possible options, given to us on next pages of the wizard.

7.3. 1. Page 3. Alternative one-dimensional basic conditions

If you want to make a simple gate, or an alternative of simple gates, you can do this on the following page of the condition wizard.

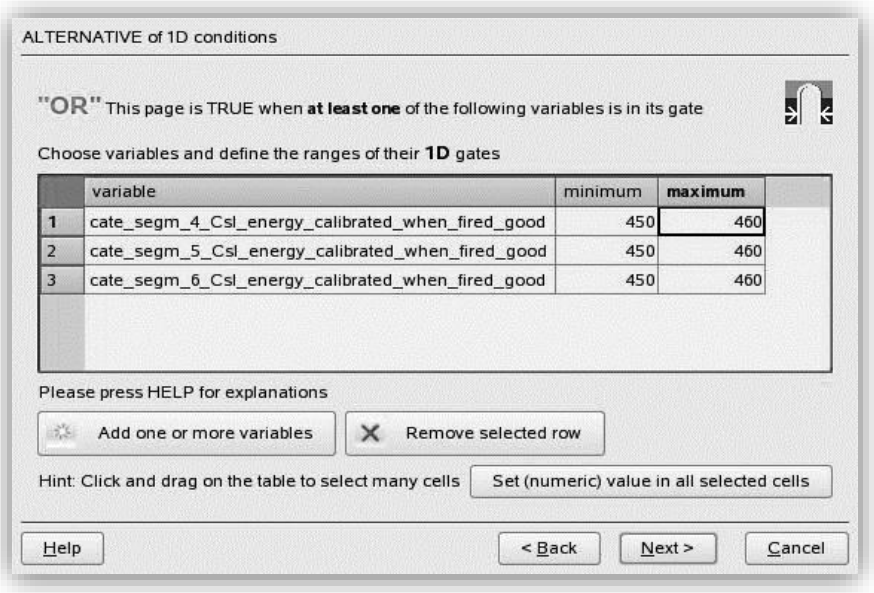


Figure 17. A page of the condition wizard, where you can create an alternative of gates (elementary 1D conditions). If you need just one gate –use only just a first row of the table.

KOTWICA

This page has a table with many blank lines. In one of them you can put the name of some interesting incrementer. Then you follow it by two numbers denoting the lower and upper limits of the range. If the current value of the incrementer will be in the range described – then this basic condition (gate) will be considered met.

As you see – here we call the Spy variables: "incrementers", although now they are not used for incrementing anything, but just to set a gate on them.

To create your basic condition, you need to

1) Press the button "Add one or more variables". It will open a small dialog box with a list of available incrementers.
2) From this list you can choose the incrementer which you need.
   If you choose more than on item – all they will be placed in next rows of the table.
3) As before – in the next columns of the table, you need to type numbers specifying the desired range of the gates.

If several basic 1D conditions (gates) are defined in this table (i.e. several rows of this table have been filled), then the logical value of this condition table is an alternative to individual rows

row1 V row2 V row3 V row4 V...

This list of basic conditions has a logical value of "true" when at least one of the basic 1D gates is true.

If only one row is used in this table – it is obviously difficult to talk about an alternative. We just used one gate. A logical value of this basic line – becomes the value of this wizard's table (page).

7.3. 2. Page 4. Conjunction of one-dimensional basic conditions

If you need a condition which is a **conjunction** of several basic gates – you can do it on the following page of the wizard.

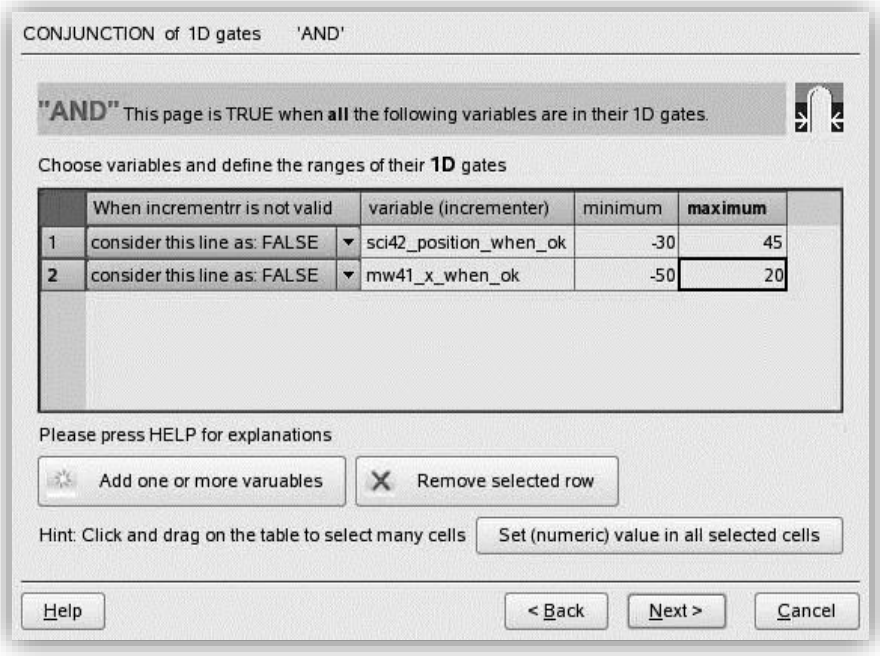


Figure 18. The wizard's page that realizes the 1D elementary gates conjunction. As you can see, there is an additional column that defines how to evaluate the condition if the incrementer has an undefined value in the current event.

There is a similar table on it, but a logical value of the entire expression created in this table can be written as:

$$row1 \wedge row2 \wedge row3 \wedge row4 \wedge \ldots$$

However, in this table there is a difference

In a conjunction situation (like here), we should take into account that sometimes the incrementer used in a given row of the table, may not contain a meaningful data

e.g. when it was impossible to calculate its value, or the appropriate detector did not fired.

So a problem arises: how to calculate the conjunction in that case.

A) The answer immediately comes to mind: "consider such an basic condition to be false".

But sometimes this solution may not be a good.

Let's imagine that we want to create a condition that tells us if all the γ quanta registered in a given event had time data in the range of 500-600. When creating such a condition, we place 105 incrementers in this table and specify each range [500, 600].

By the way, you do not have to type this numbers 105 times. Just select 105 table cells with the mouse and press the "Set (numeric) value in all selected cells" button (shown in Figure 18 near the bottom).
A small dialog box will appear and here you enter the value 500. By this the number 500 will be placed in all selected boxes. Similarly, you can do with typing the number 600.
However, you can make this in an even more clever better way:
you can use place here one collective incrementer representing all 105 incrementers.

Regardless of which way of typing gates we choose, the problem still remains.

To understand the problem better, let's assume, that we wrote these 105 incrementers.

So, now we have 105 incrementers and their ranges in the range [500 – 600]. Will we get the condition we wanted? No. In this way, we have created a conjunction that will almost never be true.

> It is almost impossible that all 105 detectors will fire in the same event. If at least one does not work – the entire conjunction is false.

Therefore, treating basic conditions (belonging to detectors that did not fired) as having a logical value *false* is not a good choice.

B) The second possible answer is: "a row with a gate on a incrementer that (in a given event) has no meaningful value – should be treated as true".

This may also not be a good option. Let's imagine the situation of the conjunction of basic gates referring to the position of the ion passing through two multiwire chambers:

mw41_x is in the range [–5, +5]
mw42_x is in the range [–5, +5]

This condition is to be true if the horizontal position of the passing ion – measured by the given mw41 wire chamber and by the mw42 wire chamber – is within a narrow range [-5, +5]. Unfortunately, in some events one of these values is impossible to calculate (because e.g. one of these wire chambers in a given event did not work). How then should one treat such a row in the table (basic condition)?

In the previous example, we saw that the "treat it as false" solution was not good. What then? Let's treat as true? Also not. It is obvious that if one of the wire chambers did not provide data – then the basic condition cannot be true; and the whole conjunction should also be false.

As you can see, in the conjunction table, sometimes one approach is needed, and sometimes another.

That is why in the first column there is a selection box (so-called combo box) offering a selection.

What should you choose in a particular case? The answer *"no one knows this better than the experimenter himself creating this condition"* – is unreal; especially in an complicated experiment, where the groups of experimenters change almost every week. So to avoid confusion – at this point in the condition wizard, the program offers a help.

> When assessing the type of increment (and whether it has its own validator), the wizard suggests what the most likely option should be used in this case. You may accept its suggestion or not.

7.3. 3. Page 5. An alternative of basic two-dimensional conditions

The next two pages of the condition wizard allow you to use a two-dimensional condition (e.g. a *banana-shaped gate*). There are tables on these pages where you can place several such basic conditions.

The first of these pages (page 5) allows you to create an ***alternative*** of such conditions, and the second (page 6)– a ***conjunction***.

The

Figure 19 shows a page dedicated to an 2D alternative.

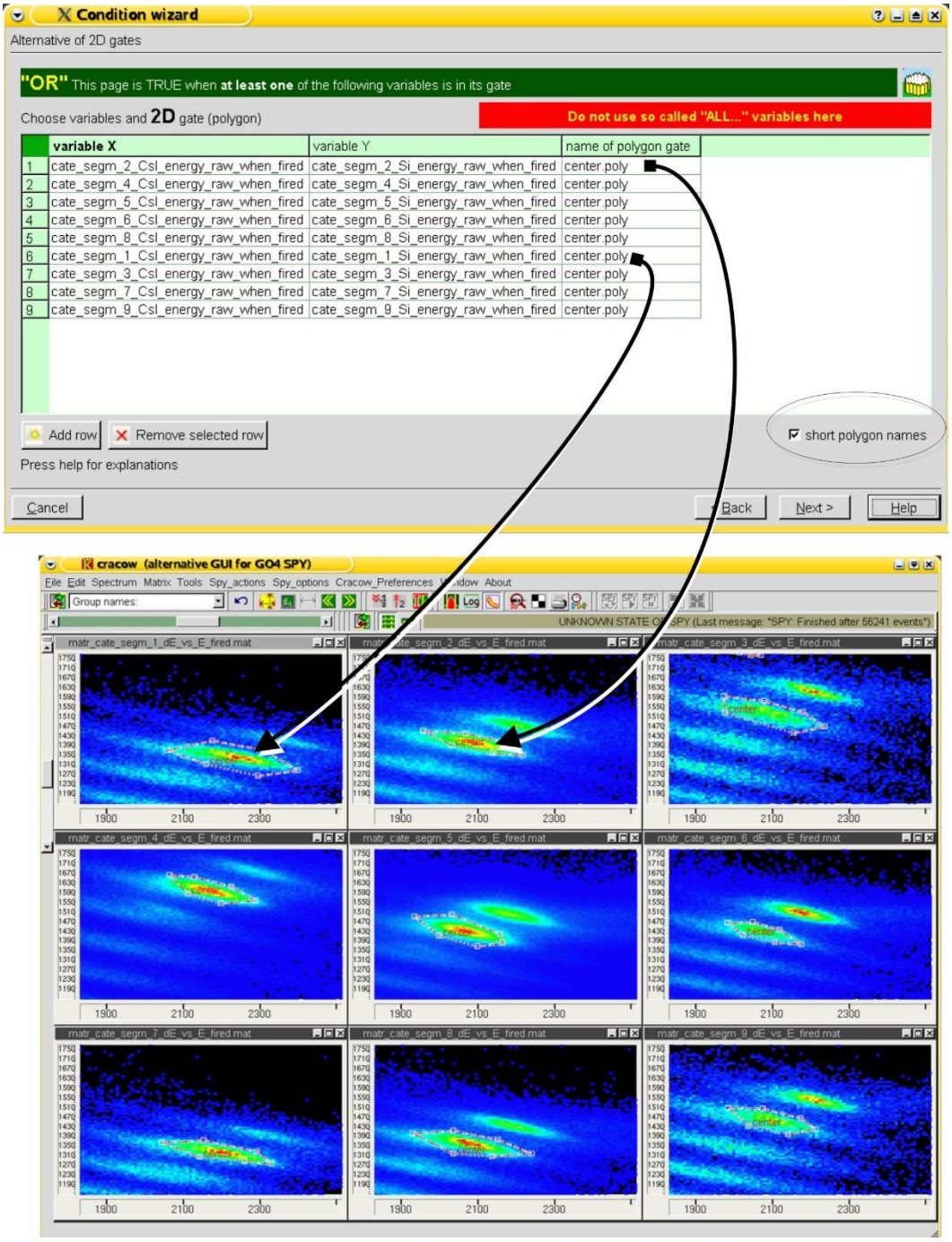


Figure 19. The wizard page (top) where the experimenter can create an expression that is an alternative to two-dimensional gates.

In this case, it was a condition on nine segments of the CATE recoil-detector used in the relativistic beam campaign, which was used to identify nuclei leaving the shield Each of the nine segments of this detector allowed to build its own matrix ΔE vs E enabling the identification of the mass of ion that hit it. These matrices are visible at the bottom. Each of them has a polygon gate drawn. (You can draw them using a mouse).

If the nucleus hits one particular CATE segment, then (of course) it did not hit another – this is why we are interested in the **alternative** of nine polygonal gates.

As you can see, all polygons had the shorten name "center". In fact, the name of the polygon is enriched with the name of the matrix on which it was drawn, so there is no risk of name collisions.

The *alternative* of 2D conditions (banana gates) is useful for example in situations where our measuring system has a detector which consists of several segments.

A particular ion (or γ quantum or β particle) can be registered in one of the segments. If this happens, we require that the parameters of this particular hit – meet some condition described by the corresponding polygonal gate.

The condition is considered true if at least one segment registers such a situation. (That's why we use *alternatives*).

How to produce this type of condition?

1) At first you must draw a polygonal gate on selected matrices by selecting the region of interest. You need to invent some name to this new polygonal gate.

   Note: a polygon gate is not a condition yet. It's just a polygon described by a set of its vertices.

2) You can use this polygon to create a basic 2D condition. To do this, just put in the first and second column of the table two increment names (x and y), and in the third column – the name of the polygonal gate.

   Of course these x and y incrementers should be the same incrementers, which were "powering" the matrix, where your polygon was drawn.

How this basic condition will be processed?

After completing the analysis of each event, the online analysis program will check the current values of the incrementers (x and y) – it will check if the point P (x, y) lies inside the given polygon. If so, this basic condition is considered to be true.

In the table shown on

Figure 19 (above) you have seen alternative of 9 such basic conditions. The result of the conditional expression built on this page of the wizard is an alternative to individual rows of the table

$$row1 \ \vee \ row2 \ \vee \ row3 \ \vee \ row4 \ \vee \ ...$$

7.3. 4. Page 6. Conjunction of basic two-dimensional conditions

The next page of the condition wizard is very similar to the previous one – except that the basic conditions listed in the table form a *conjunction*. (Figure 20).

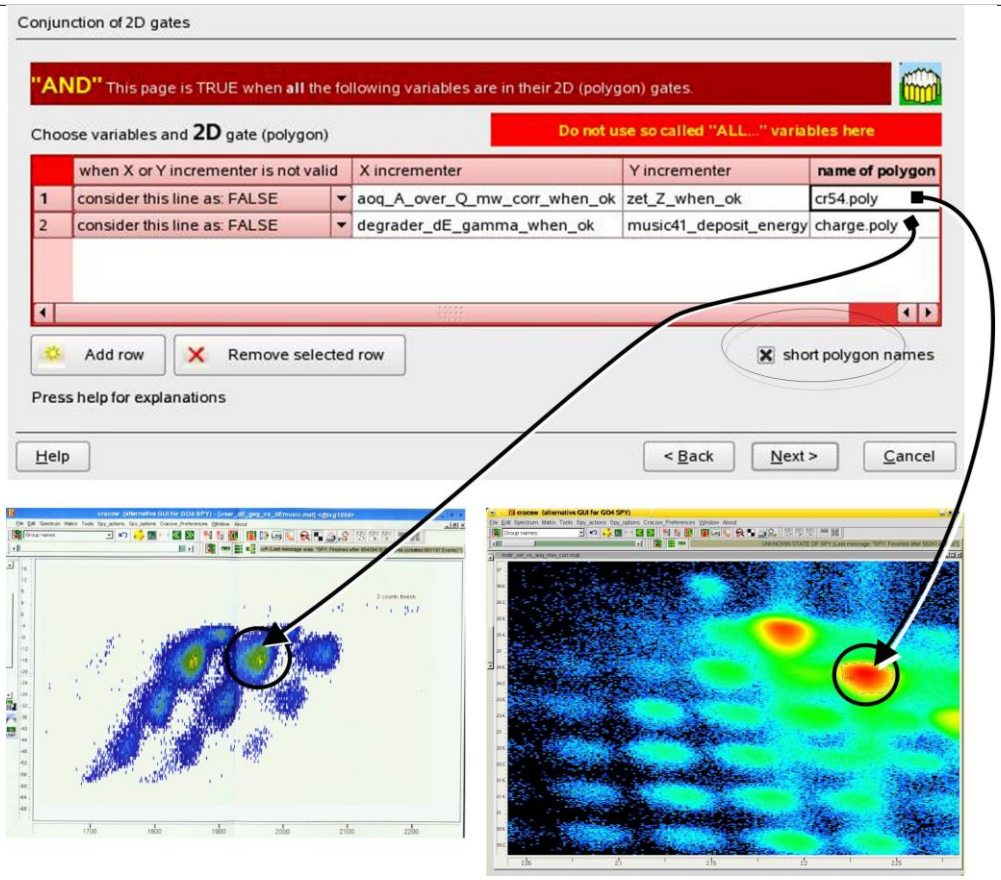


Figure 20. (**Top**) The page of the condition wizard, where the experimenter can create a conjunction of two-dimensional basic conditions.

This is an example of a situation when, in addition to the gate on the Z versus A/q matrix, an additional one is required – taking into account the loading conditions.

The spectrum in the bottom **right** is the spectrum of Z versus A/q. The polygonal gate drawn on it is used to select the desired kernel (in this case it was 54Cr).

The spectrum in the bottom **left** is the spectrum in which the energy loss in the ionization chamber music41 is shown on the Y axis, and the energy loss in the absorber located in the s2 focal plane on the X axis. The polygonal gate placed on it, is used to select the charge state of the nucleus.

KOTWICA

This table allows you to create a logical expression that is true when all the two-dimensional basic conditions placed in the rows of this table are simultaneously met. In other words – it is a conjunction of basic two-dimensional conditions.

$$line1 \wedge line2 \wedge line3 \wedge line4 \wedge \ldots$$

As in the case of the previous 1D conjunction, the problem discussed earlier appears also here:

what to do when the evaluation algorithm proceeds when the selected variable (incrementer) does not have a specific value in the current event (because e.g. one of the detectors did not work in the given event).

Should such a basic condition have the logical value true or false? Because there is no one general answer to this – the user has the option (in the first column of each row of the table) to determine the behavior of the algorithm that calculates the conjunction in his particular situation.

A good advice

> As mentioned before – it is better to make *two separate conditions* from the two lines presented in the table above.

7. 4. Page 7. Conditions which are built of other conditions

The four pages of the condition creator which were presented here so far – allowed us to build even very sophisticated conditions. In this form, the GREWARE/Spy has been used for many months. However, there was a wish coming from the experimenters: the ability to create logical expressions from other existing conditions.

Here is an example of this situation. The experimenter has already prepared a condition that allows him to select the correct ion produced by the fragment separator. He can now copy (clone) this condition and modify it by enriching it with some additional basic conditions. This is the most correct procedure, but copying it will cause that the same basic conditions…

(e.g. "does point P (x, y) lie in the polygonal gate or outside of it?")

…has to be calculated twice (for the same event) – which is inefficient.

Therefore, it would be beneficial to be able to build the condition according to the following principle:

"I would like to create a new condition that is "true" when the other condition (called N) is "true"– and when some additional requirements are also true. Here there are these additional requirements...".

The GREWARE was therefore equipped with procedures that allow a condition – to refer to the current value (true/false) of other conditions. It is not only a matter of convenience or economy. The problem appeared for the first time when the experimenters used the so-called veto-detector. It was about building a condition that is true when some other condition is not true.

To make this possible, the condition wizard has one more page. It lets you to build conditional expressions from the results of other (already existing) conditions

.

Figure 21. A condition wizard page on which you can create logical expressions from other existing conditions. The appropriate tables contain the names of the conditions selected from the list of available conditions. As you can see, in this case the experimenter has put four names in the AND table (conjunction of other conditions).

KOTWICA

There are four tables on this page of the wizard. In this tables you can put names of other existing conditions. (You can easily select them with a mouse – from the list of all available conditions).

These four tables allow four logical operations on the results of other conditions.

a) The first table performs the AND operation,

b) the second OR operation,

the next two are

c) NAND

d) NOR.

You do not have to use all of these tables. An empty table is considered non-existent. Those that have been used – must all have a Boolean value true, because the entire logical expression represented by this page of the wizard is a conjunction of these four tables.

$$(warA \wedge warB \wedge warC \wedge \ldots) \quad // \leftarrow \text{table AND}$$
$$\wedge$$
$$(warX \vee warY \vee warZ \vee \ldots) \quad // \leftarrow \text{table OR}$$
$$\wedge$$
$$\overline{(war1 \wedge war2 \wedge war3 \wedge \ldots)} \quad // \leftarrow \text{table NAND}$$
$$\wedge$$
$$\overline{(warM \vee warN \vee warP \vee \ldots)} \quad // \leftarrow \text{table NOR}$$

7.4. 1. Risk when nesting conditions

Chain

Allowing conditions to use the result of other conditions may cause some problems. Let's imagine such situation. There is a condition A. We define a condition B, which uses the condition A. Let's write this situation as: B → A

The online analysis program will be able to calculate the logical value of the condition B only when it first calculates the value of the condition A. We can also build a condition C, which uses the condition B. Let's write this situation as: C → B → A

As you can see, **a chain of conditions** is created. The calculation of the condition C value is possible if the logical value of the condition B is calculated beforehand, and this requires the prior calculation of the value of the condition A.

If the event analysis program, when calculating the logical value of a condition, sees that it needs the value of another condition for this calculation, it suspends the calculation of the former for a moment and calculates the value of the second condition. When it calculates, it returns to condition one and takes advantage of this result.

Tree

Note that a condition may use logic of some other conditions, so by this **a structure of "tree"** is created (see Figure 22).

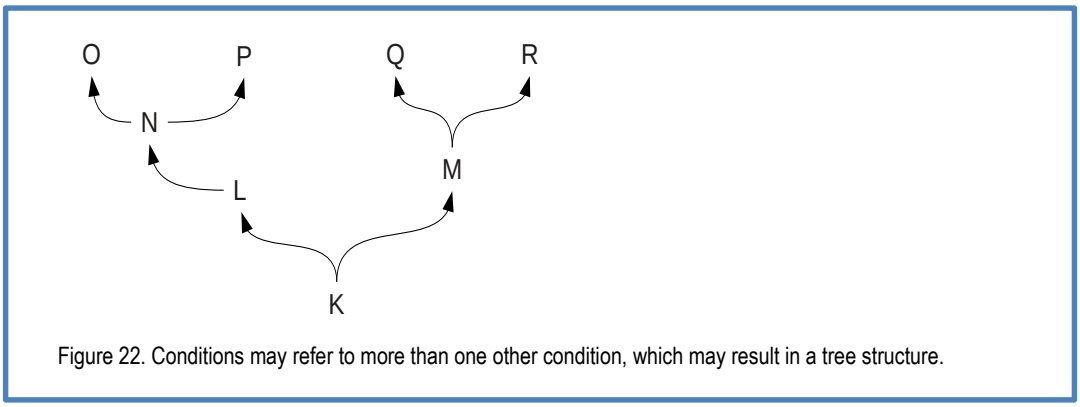


Figure 22. Conditions may refer to more than one other condition, which may result in a tree structure.

Such a complex form of the condition is not a problem for the event analysis program – it simply uses recursion when calculating the conditions.

The problem appears elsewhere:

If we create always only new conditions, there is no risk, but if we modify the already existing condition W, it may happen that we think that it should use the result of another condition V. However we do not remember that condition V refers to ours W.

$$W \rightleftarrows V$$

As a result, when the analysis program begins to calculate the value of the condition *W* – it decides that first it must calculate the condition (*V*). So it starts to deal with the *V*, in which it turns out that the value of the *W* is needed, therefore it will deal with the *W*, in which he will again be sent to the *V*... – and so on, forever.

"Vicious circle" (magic wheel) of checking conditions.

This infinite loop may not be visible at first glance. One can imagine a chain situation where only a 7th or 12th link refers to the first – closing it with "circulus vitiosus" ("a magic wheel").

> To prevent such a logical error, it is enough to remember that in the **chain** of conditions – the same condition cannot occur twice.

Lucky, the Spy (analysis program) is able to detect this chain situation and inform the experimenter about the logic error of a given condition.

In the case of complex logic of conditions – which are forming a *tree* – the situation is slightly different.

> However in a **tree** ,the same condition can occur even several times (without causing a logical error) – as long, as it does not occur twice in one "path".
> *The path is – using a dendrology analogy – every path from any leaf of the tree to the main trunk.*

The same condition can be in different leaves, or in different branches of a tree, as long as these different branches appear in one path.

7. 5. Making *Veto* conditions

Sometimes experimenter wants to create a condition that is false when another condition is true.

For example: Did this detector fired? – If yes, that means that the nucleus flew somewhere, where it should not – so such event should be discarded.

One "veto-detector"

To create such a veto condition, a simple negation operation is sufficient. If there is one "veto-detector" in the measuring system, then:

1) first, we create a condition in which we pretend that **we want** the detector not only to work, but in addition to register something in the forbidden range of values;
2) Then we build the second (proper) condition, in which on the last page of the wizard we put the name of the first condition in the NAND table (or in the NOR table);

   In case there is only one condition – it doesn't matter if you choose the NAND or NOR table.

Such negation is simple and intuitive for everyone.

However, if we have two veto detectors – some users have problem while building the veto condition

Here is an example. The user has already created two conditions – stating that detector A and detector B have triggered and registered a forbidden situation. Now the user wants to create a veto condition that is true when veto detector A "did not protest" and at the same time veto detector B also "did not protest". In other words, this is the case

$$\overline{(detA)} \wedge \overline{(detB)}$$

By creating this new veto condition – are you wondering in which table should you put the names of conditions A and B?

Unfortunately, not everyone remembers de Morgan's law:

$$\overline{(detA)} \wedge \overline{(detB)} \equiv \overline{(detA \vee detB)}$$

According to this rule, both conditions should be placed in the NOR table, i.e. the one in the lower right corner in Figure 21, (page 55).

More user friendly

Rules are rules, but you have to take into account that we are talking about online analysis, where the experimenter often works at night and in fatigue – which promotes making mistakes. At 3 a.m. not everyone knows what means NOR. Therefore, in the header of the condition wizard tables there are also explanation hints with "common" language. They say that…

> The given array of foreign conditions is logical *true* if:
> array [AND] → when all the conditions given are true
> table [OR] → when at least one of the conditions given in the table is true
> array [NOR] → when all given conditions are false
> array [NAND] → when at least one of the conditions is false.

7. 6. Assigning a condition to the spectrum

After completing the selected pages of the condition wizard and completing the wizard – the condition definition is saved to disk. The condition exists, but since no spectrum uses it, the Spy analysis program will not evaluate it.

To make use of the condition – you should:

1) Go to the Spectra Manager.
2) Select a chosen spectrum definition and open it for editing (with the spectrum wizard).
3) Go to the last page of the spectrum definition, where you can specify what condition must be met in order to allow the spectrum to be incremented.
   By default, the "No condition" option was selected there.
   Now you can select a condition from the list of all available conditions.

After this, the spectrum becomes really a *conditional* spectrum.

7. 7. Example of logical expressions created from other conditions

Here is an example which will demonstrate result of the logic of other conditions.

1) We will use a two-dimensional spectrum which displays a cross-section of the beam detected by the Time Projection Chamber detector (called tpc42). Such a spectrum can be simply defined using incrementers:
   tpc42_x_when_ok
   tpc42_y_when_ok

The result, a spectrum called user_tpc42_xy, is shown in the upper left corner of Figure 23 (page 59).

2) In this spectrum, two polygonal gates can be drawn. One with the shape of a vertical belt and the other with a horizontal shape, with a rounded right edge. (In the drawing they are visible as drawn with white and red lines).

3) On the basis of each of these polygons, conditions are built. One is called vertical and the other is horizontal.

4) Then we clone the user_tpc42_xy spectrum giving it the name user_tpc42_xy__cond_pion and assign it a vertical condition.

5) We clone the same spectrum again. This time we give it the name user_tpc42_xy__cond_level. We also decide that this spectrum can be incremented if a "horizontal" condition is true.

Now we analyze some data. The result can also be seen in Figure 23 (page 59). Look at the second and third spectra in the top (first) row. As you can see the conditions work correctly, because the matrices are filled only in the areas designated by the appropriate polygons.

To check the logic of external conditions, let's define four new conditions

6) In the first of them, the names of the horizontal and vertical conditions are placed in the AND table, in the second in the OR table, in the third NAND, and in the fourth condition in the NOR table.

7) To see the effect of these four conditions, we again (four times) clone user_tpc42_xy. We name each new spectrum and assign the above conditions.

So new conditional spectra were created with the names:

user_tpc42_xy__conditions_AND
user_tpc42_xy__conditions_OR
user_tpc42_xy__conditions_NAND
user_tpc42_xy__conditions_NOR

These spectra are shown in a figure below (second and third row). The shapes, appearing on the spectra, convincingly confirm the correct operation of logical operations on the conditions.

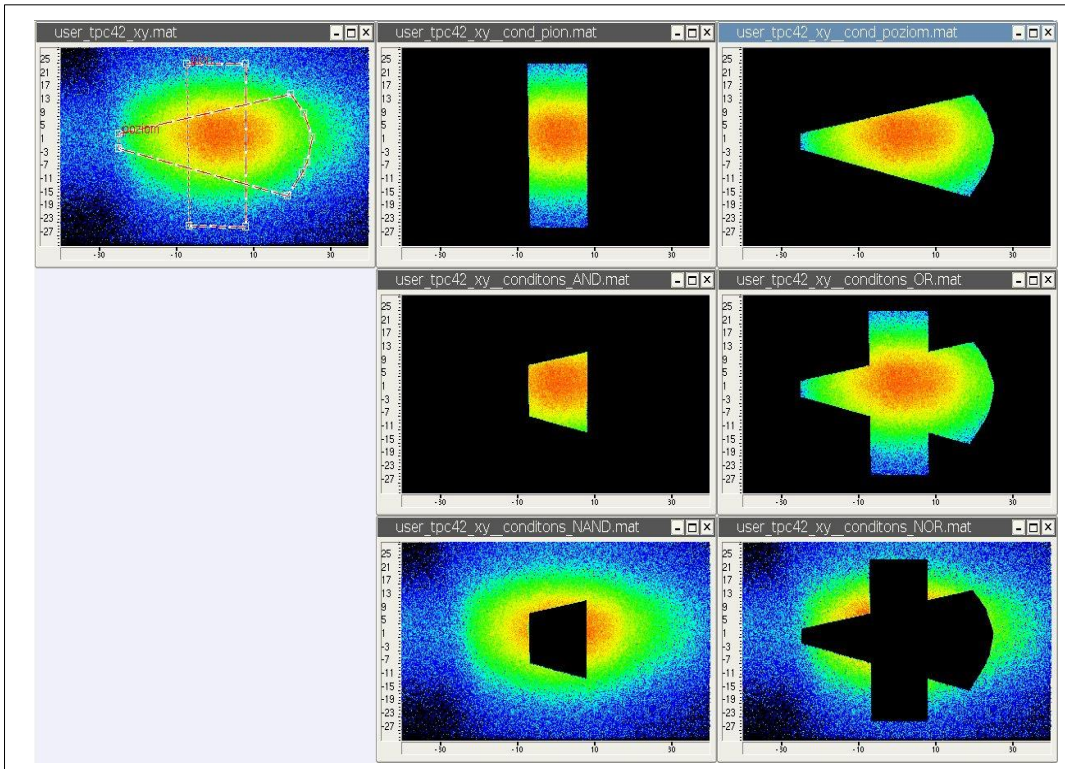


Figure 23. Illustration of the result of applying a condition which is using a logic of some other conditions.
In the upper left corner – a matrix with two polygonal gates marked on it. One polygon is "vertical", the other more "horizontal".

All other matrices in this picture are copies of this matrix, except that different conditions have been applied to them.

- The second matrix in the first row – is a matrix with a condition built on a "vertical" polygon,
- the third matrix – is a matrix with a condition build on a "horizontal" polygon.

Most interesting spectra can be seen in the rows 2 and 3 of this picture.

They were created thanks to the conditions in which one of the logical expressions from "foreign" conditions (here: "horizontal" and "vertical") was used.

- For spectrum 4 – it is an AND logic operation,
- for spectrum 5 – OR logical operation,
- for spectrum 6 – NAND logical operation,
- for spectrum 7 – logical operation NOR.

8. Self-gate, a local condition set on other incrementers of the same detector

With tools presented in previous chapters you can create your own spectra and your own conditions – whatever you need. However, sometimes it is tedious. While creating a complex conditional spectrum, you need to provide some intermediate stages; this mean you need to create many uncomfortable indirect spectra or many indirect conditions.

The error-prone aspect of this process is – how you invented a logic of your conditions.

It may happen that you create conditions that can never be met.

> Especially one type of the user error is so frequent, that I had to create a special condition type, to avoid this error.

This tool is presented in this chapter.

8. 1. Conditions with a wrong logic

Imagine, that you are working with a spectrometric device such as RISING-array of detectors. The most basic spectrum needed in the experiment is the energy spectrum of the γ quanta detected by individual crystals of germanium cluster detectors (named A1, A2, ... R6, R7).

Using GREWARE you can easily create such a spectrum with a help of the spectrum wizard.

- On the list of incrementers of such a spectrum – you should put 105 incrementers:

  rising_A_1_energy_cal_when_good
  rising_A_2_energy_cal_when_good
  rising_A_3_energy_cal_when_good
  ...
  rising_R_6_energy_cal_when_good
  rising_R_7_energy_cal_when_good

  No problem, all what you have to do, is to mark these 105 items on the list of available incrementers and confirm your choice.

- Another, maybe "more modern" way is to use one collective incrementer called:

  ALL_rising_energy_cal_when_good

  This incrementer – as we remember – contains a list of the above 105 incrementers.

Regardless of which of the ways we choose, we will obtain the desired spectrum.

Will this spectrum make you happy? Not yet. Such a spectrum is available in the analysis program as build-in, so this was not a reason to create it once more. We did created this spectrum as “user-defined”, because want to increment it under some condition.

First case – a very “general” condition

Let's assume that we want a condition set on a “time of flight” value.

In our experiment the ion which passes between the sci21 scintillator and the sci41 scintillator. The time if this flight is measured in the acquisition system by a TDC converter and the result is available in the event as raw data.
This value is recalculated to picoseconds by applying calibration factors. We can see its values in the build-in spectrum tof_21_41_picoseconds.spc. This value is also available to us as an incrementer.

Using the incrementer tof_21_41_picoseconds, we can build a condition (requesting, for example, that the measured time of flight (measured in a given event) is in the range *[a, b]*). After assigning such a condition to a previously created spectrum, it becomes a conditional spectrum.

Schematically, the definition of this conditional energy spectrum can be shown as follows:

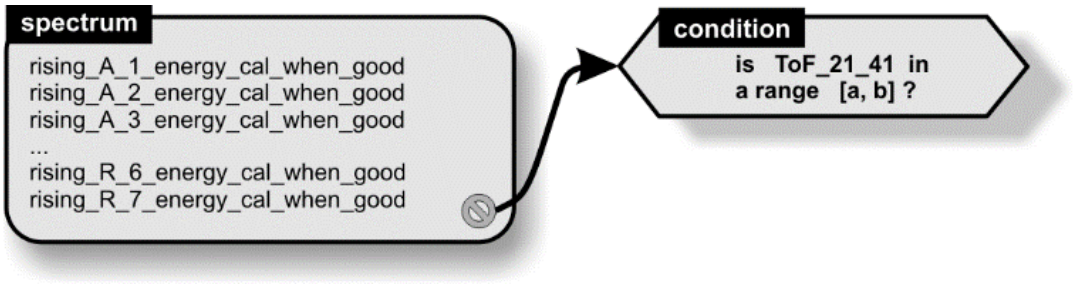


Figure 24. Schematically presented method of creating a conditional energy spectrum of γ quantum recorded by RISING detectors, provided that the time of flight of the nucleus between two separator scintillators (sci21, sci41) is in the time interval [a, b]

This spectrum is correct and meets the expectations of the experimenter.

Second case – confusion arrives

Now imagine, that you would like to have this spectrum under another, more demanded condition. We want a similar energy spectrum of γ quanta, but only if they were recorded in a specific time period.

As we know, the germanium detector provides information about energy and the time of the recorded γ quanta. This data is also available as a set of the following 105 incrementers:

rising_A_1_time_cal_when_good
rising_A_2_time_cal_when_good
rising_A_3_time_cal_when_good
...
rising_R_6_time_cal_when_good
rising_R_7_time_cal_when_good

Let us assume that we want them to be in a gate [2000, 2050] (arbitrary units).

Therefore you must therefore create the appropriate condition. The condition wizard gives two possible expressions for 1D gates:

- conjunction of basic 1D gates,
- alternative basic 1D gates.

In the condition wizard, you can put the following 105 incrementers in the table representing the conjunction of gates:

when incrementer is not valid, consider as	Incrementer	min	max
true	rising_A_1_time_cal	2000	2050
true	rising_A_2_time_cal	2000	2050
...	...	...	...
true	rising_R_7_time_cal	2000	2050

(The same effect can be obtained more simply by using one collective incrementer ALL_rising_time_cal.)

After assigning this condition to your spectrum – you can start the analysis.

You will be surprised. Observing the spectrum, you will note that it has fewer counts than one would expect.

The condition is probably done incorrectly. Why?

Let's look at this situation more carefully

Imagine an event in which 4 gamma quanta were emitted. Three of them were emitted during the time in the desired range [2000, 2050], but the time of the fourth one is already out of range. Since this is a conjunction of four gates, and one of them is unfulfilled, the whole condition is unfulfilled.

As a result, the energies of the three "good" γ quanta will not have the chance to increment the spectrum.

You probably made some logic mistake…

So what to do?... Perhaps you should not use a *coincidence* but an *alternative* of 1D such gates?

You change the definition of your condition and now you place the incrementers in the **OR** table:

Incrementer	min	max
rising_A_1_time_cal	2000	2050
rising_A_2_time_cal	2000	2050
...	...	...
rising_R_7_time_cal	2000	2050

(The same effect can be obtained more simply by using one collective incrementer ALL_rising_time_cal).

After making this change and restarting the analysis – you will notice that now there are many more counts in his spectrum.

However, a careful look at the spectrum can convince you that in the spectrum appear even the energies of such γ quanta, which you know that should definitely not be there.

Why it is so?

Well, let's imagine that we have an event in which six quanta were registered. The registration time for three of them is within the desired range [2000, 2050], but the time for the remaining three is not. However, since it is an alternative – the whole condition has a logical value true. Thus, all 6 g quanta energy can increment the spectrum (i.e. even these 3 quanta from outside of the time interval).

As we can see – also in this situation you made a logic error while building the condition.

Where is the error in this logic?

In short: during thinking about the condition for your spectrum, (in your mind) you used a logical expression. The mistake is that in that expression you put parentheses in the wrong places.

- You should **not** build a spectrum from many incrementers and *en-bloc* create a condition for it.
- You should rather create a separate spectra for each of the 105 quantum energies – and to each of them assign a condition set on the *corresponding* time of its γ quantum should be attached.

In other words, instead of the situation, as in the following diagram:

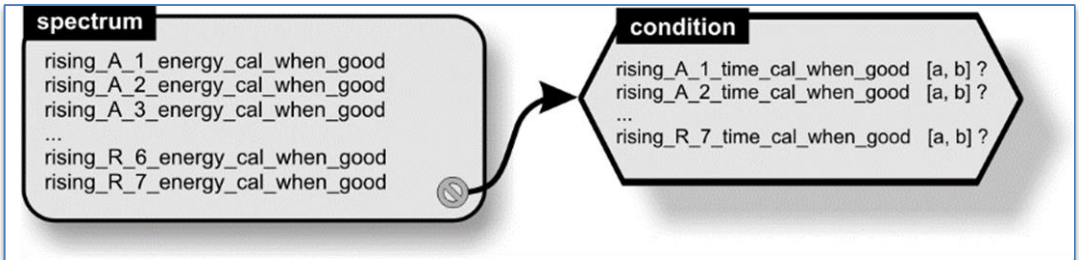


Figure 25. An example of bad logic of the time condition set on "sister" incrementers from the same detector. (Closer discussion in the text).

we should made it as it is shown schematically in a next diagram (Figure 26).

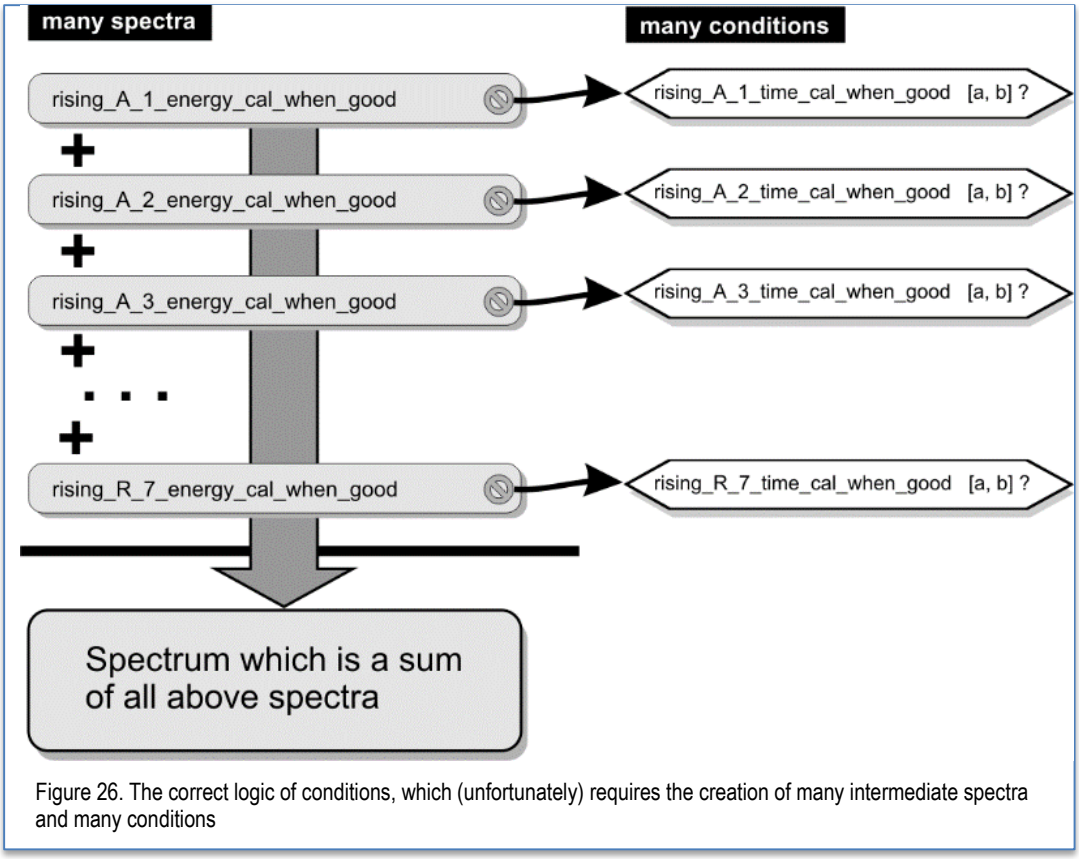


Figure 26. The correct logic of conditions, which (unfortunately) requires the creation of many intermediate spectra and many conditions

This means that – to build the desired conditional spectrum correctly – you must:

1) define 105 time conditions for each germanium crystal;
2) define 105 spectra of energy of each germanium crystal; to each of them assign a corresponding condition (made above),
3) perform an analysis of events, as a result of which our spectra will be collected;
4) after finishing the analysis add all 105 spectra, resulting in the desired spectrum.

The issue of creating these 105 spectra and 105 conditions may sound dramatic, but in the GRETWARE GUI program you have a powerful tool to do this (in the user defined spectra manager and the user condition manager).

(see. Smart cloning §5. 3, page 32)

Do not start doing this smart-cloning yet. There will be something better: a self-gate!

Just for your curiosity: if the self-gate tool was not available – you should make your task it manually

ad 1).

a) It is enough to create a condition for the A_1_time incrementer.

b) Then by pressing a smart multi-cloning button you can clone this condition

the pattern A_1 should be replaced with patterns from A_2 to R_7

Similarly with spectra.

ad 2).

a) Create a spectrum for the incrementer rising_A_1_energy_cal. This spectrum should have assigned a condition set on of A_1,

b) then pressing a smart multi-cloning button you can "intelligently" clone this spectrum definition

the pattern A_1 should be replaced with patterns from A_2 to R_7

By this we've easily created 105 conditional spectra.

ad 3) and 4)

- GREWARE GUI allows you to automatically create the sum of selected spectra (and update it every 60 seconds), so you do not have to wait until the end of the analysis to see the current result. Already during it we will see the desired spectrum, whose counts increase (because every 60 seconds they are refreshed).

The problem seems to be solved, the correct logic of conditions has been created, the final spectrum is what we meant, but ... we have 105 spectra and 105 conditions on the disk.

It can be even worse. This type of "total energy" spectrum is so popular – we may need a version of this spectra with slightly different conditions. So again there will be next hundreds of intermediate spectra (which no one will ever look at), there will also be hundreds, maybe even thousands of conditions ...

Everything seem to be *lege artis*, but… is it really a user-friendly analysis system?...

8. 2. Solution: a local condition called a self-gate

This is where the idea of introducing a new type of condition came from. A condition that will solve the problem presented above.

Once more, what we need:

- We would like to be able to build a spectrum that is powered by many (e.g. 105) incrementers,
- each of these incrementers can increment a spectrum only when it meets **its own** specific local condition.

So we need a new type of condition. A local condition which is assigned not to a spectrum, but to an incrementer. If we use this local condition it for an incrementer belonging to a detector

(for example: 'energy' of detector K_4),

this condition need to check something on another incrementer belonging to the **same** detector.

(for example: 'time' of detector K_4).

"Words, words, words ..."

This type of local condition has been called a *self-gate*. It was a name that would suggest applying this type of condition.

> This gate is assigned directly to the incrementer, and it relates to the condition placed on other incrementers *from its own detector*. As if on his "siblings".
>
> The gate itself is able to recognize automatically who is a sibling.

In practice: if you want to use an incrementer from one of 105 germanium crystals in the spectrum definition, then you just put a name for the self-gate condition next to the name of this

incrementer. This self-gate will know how check some local "sister" parameters (e.g. time) of particular incrementer.

Self-gate conditions are useful when:

- there are many detectors of the same type, and…
- each of them provides several incrementers (e.g. energy, time, dispersion angle, geometrical position),
- we want to create a summary spectrum of one type of incrementer ...
- ... only if another incrementer from the same detector fulfills some condition.

8. 3. How to create a self-gate condition

You can create a self-gate definition directly in the spectrum creator. If (during the definition of the spectrum) you realize that an own condition is needed for some of the incrementers of this spectrum – you can initiate the creation of self-gate definition. You just to need to press a button: "Create a self-gate". (See bottom-right corner of the picture below).

After pressing this button, the following dialog box opens (Figure 27):

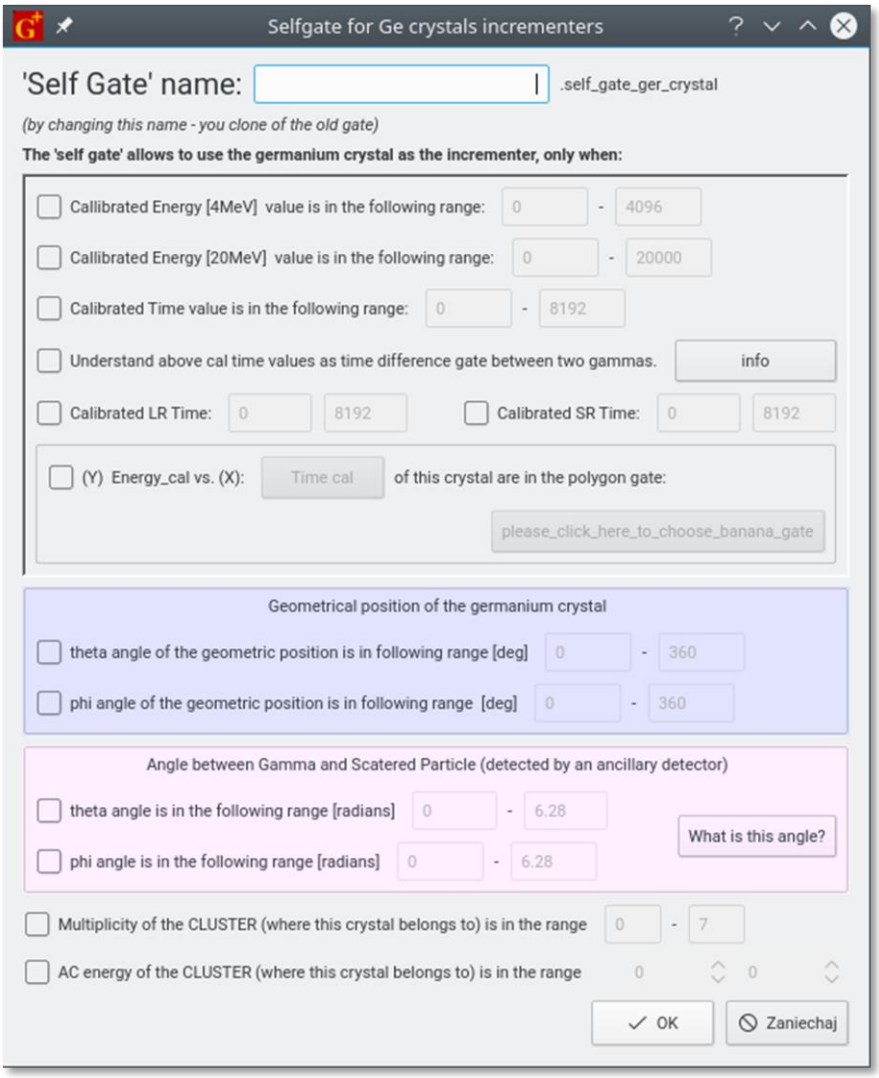


Figure 27. Dialog window for defining the self-gate condition of incrementers coming from an object representing a germanium crystal of a cluster type detector.

As you can see, it is possible to place local gates not only on "one of the sibling incrementers", but even on several of them simultaneously. So you can see possibility of setting the gates on energy, time, geometric position (useful for analyzing angular distributions) and also on the angle between the direction of γ quantum and the direction of flight of the scattered nucleus behind the shield. The meaning of individual items in this definition will be discussed in detail later (on §8.4. 4, page 71).

When the experimenter finishes defining the self-gate – this condition is saved to disk, so the self-gate is permanent.

8.3. 1. Here is an example of using self-gate

The diagram below shows the list of incrementers in the spectrum definition. Suppose the experimenter has just created a self-gate called delayed_time.

Incrementer	Self-gate name
rising_A_1_energy_cal_when_good	delayed_time

rising_A_2_energy_cal_when_good	delayed_time
...	...
rising_R_7_energy_cal_when_good	delayed_time

Next to the incrementer's name is a column in which the cells are so-called combo boxes (drop down selection lists) containing the names of all available self-gates. Just select a name of the proper self-gate (or leave the text "No_selfgate" – if you are not interested in it).

In this case, there are 105 rows in the table – so it is impractical and tedious.

> A best advantage of the self-gate condition is that it can be used the collective incrementer.

This simplifies the spectrum building process very much. Here is a simple, "single-line" definition of the "total energy" spectrum, which caused so much trouble in this chapter.

Incrementer	Self-gate name
ALL_rising_energy_cal_when_good	delayed_time

8. 4. Review of different types of self-gates

Not every incrementer can have its own self-gate condition – because not everyone needs it.

- For the AGATA experiment, the self-gate condition was useful for a germanium crystal.

In the case of the RISING experiment it turned out to be useful for incrementers from objects representing:

- (as above) RISING germanium crystal,
- a set of 7 germanium crystals forming the entire **cluster detector**,
  used in cases of energy from an algorithm called addback (not described here),
- a crystal germanium detector from the MINIBALL system,
- a BaF2 crystal detector from the HECTOR system.

Each of these detectors has a slightly different specificity, which is why the self-gate conditions defined for them look slightly different.

The user can use one of few types of the self-gate condition. The spectrum creator offers some options, as shown in a following Figure 28.

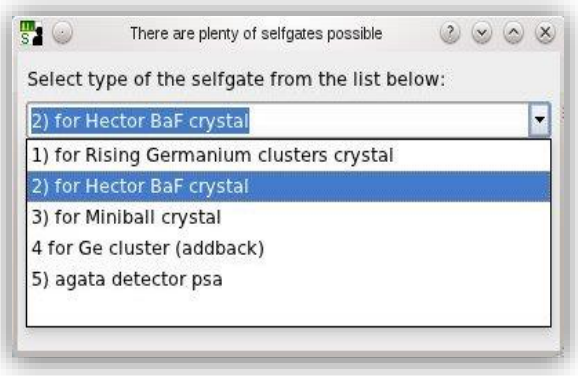


Figure 28. When creating a new self-gate condition, the experimenter has some types of self-gates to choose from. (This number will change after adopting the analysis to the other experiments). Now there are also self-gates for KRATTA detectors.

After selecting the *type* of the self-gate, a dedicated self-gate dialog will be opened.

An example of the "self-gate" condition dialog box for a cluster germanium detector crystal was shown already on page 71 (§8. 3).

In the self-gate dialog the user need to define a name. Then selects the gates of interest to him, which he can put on sister incrementers from the same specific detector. You can see a list of possible parameters on which you can place "sister" local gates.

To reduce the time it takes to evaluate the condition during the analysis – there are "enable" fields. Parameters, that we do not need to check, are marked as inactive.

If these enable fields were not present, one would have to set wide open gates on all other parameters, which are not interested for you in this case.

You may wonder…

For a particular incrementer we can use the self-gate condition.

*for example we are interested in an incrementer representing e.g. the **energy** measured by the detector.*

But look: in this self-gate condition, we have a chance to put the gate on time, on the corners, but also on the **energy** itself.

You may wonder: **Is it allowed to do that? Absolutely yes.** But think if this is what you want.

*If we create an **energy** spectrum in the range of 0-4096 keV, and put the self-gate on the incrementer with the **energy** condition described in the range of 500-600 keV, then the result will be a spectrum that will only have counts in channels in the range of 500-600 keV. The rest of the spectrum will be empty. This is not an error, but do you really want such a spectrum?*

So, as you see, it makes more sense to put self-gate on other sister parameters of the detector (gates at angles, times, times, etc.).

✧

In the following paragraphs we will see examples of self-gate dialog windows dedicated to different types of detectors – starting from the simplest, then going to the more complex. Some of the detectors mentioned here can be unknown (or historical) for you, so treat them as an illustration.

8.4. 1. HECTOR self-gate

HECTOR (High Energy g-ray Detector) was a system of eight BaF_2 detectors designed to measure high energy quanta (up to 30 MeV). They provide energy and time data for the recorded γ-quanta. Each of the BaF_2 detectors occupies a place in space, and its position is described by the

spherical angles ϕ and θ. They are normally used in nuclear spectroscopy to create a spectrum of the angular distributions radiation.

The Spy (event analysis program), knows geometric positions of every BaF2 detector.

Of course these angle data are constants, but anyway it can be useful to be able to set a gate at these angles.

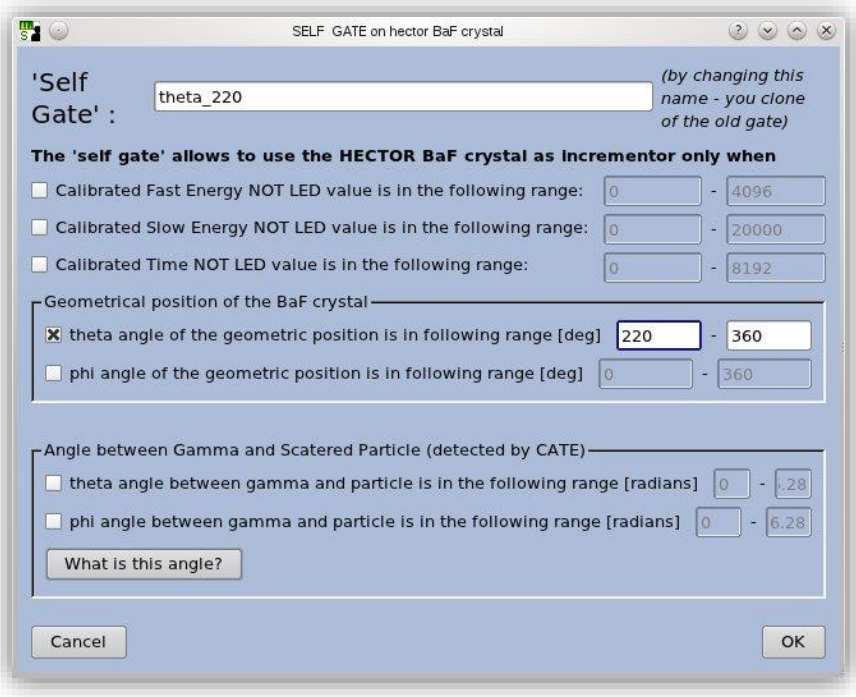


Figure 29. Self-gate condition wizard that can be placed on selected "sister" barium fluoride crystal incrementers of the HECTOR detector.

KOTWICA

You can also set gates on the angle of the scattered particle. When these angles can be useful?

There are experiments with a thin target, (where the product nucleus leaves the target and flies further). Often there are position-sensitive detectors (position-sensitive) can be set up along the path of the atom. They are necessary for calculating the Doppler correction to the gamma quanta energy, measured in the BaF_2 detectors. These detectors allow you to calculate the angle between the direction of flight of the nucleus and the direction of quantum emission. (This calculation is performed during the event analysis by the object representing the BaF_2 detector.)

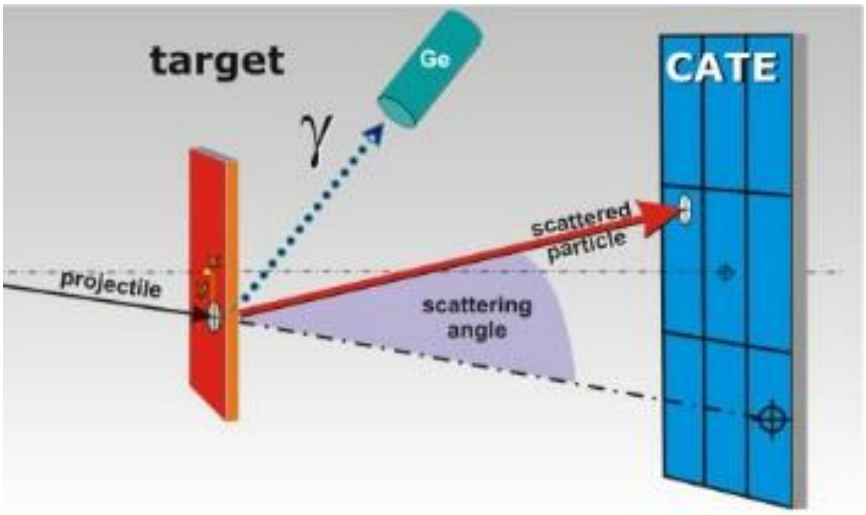


Figure 30. What is a scattering angle, you can see on this diagram. (In this case there was not a HECTOR detector, but a CATE detector).

Therefore this angle available in the form of an incrementer – so the experimenter can also set the self-gate condition on it (two lines at the bottom of the dialog box).

8.4. 2. MINIBALL self-gate

There is also a type of self-gate dedicated to the MINIBALL detector system. These are also gamma radiation detectors, so no wonder, that a set of sister parameters (on which the self-gate can be placed) is similar.

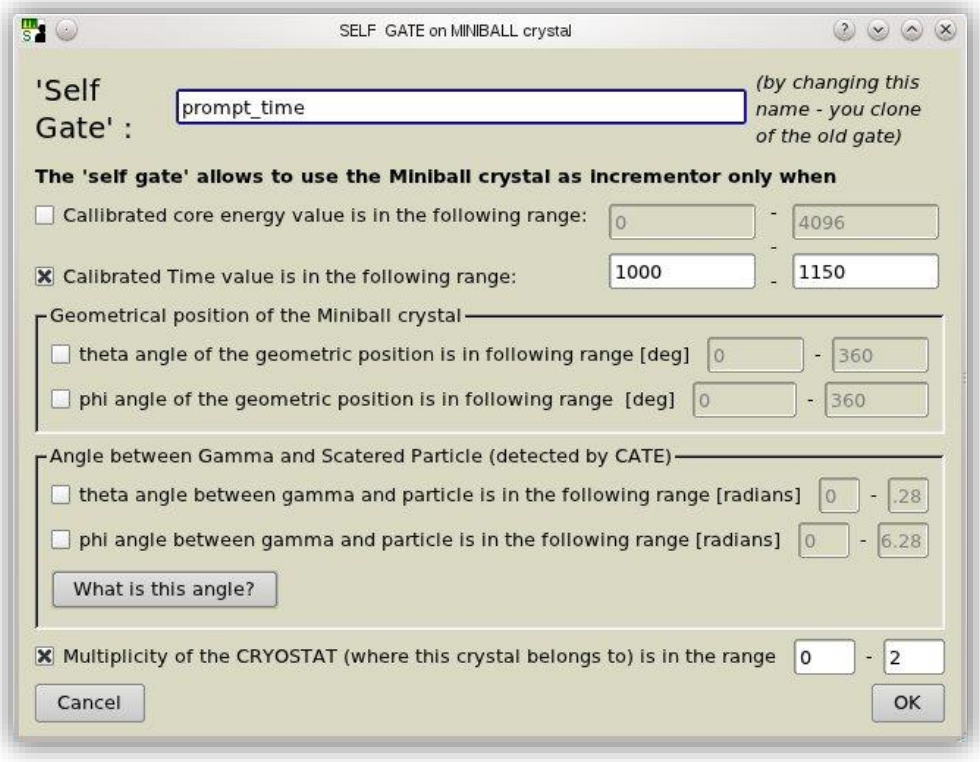


Figure 31. Dialog window where the experimenter can define the self-gate condition for MINIBALL detector crystals.

At the bottom of the dialog box you can see a new thing: the MINIBALL germanium crystals are mechanically grouped three in a common *cryostat*. As you can see, you can put the condition on this how many gamma quanta were registered by a given cryostat).

> *For example, we require only one Ge crystal to "fires" in a given cryostat, and the other two do not register any gamma quanta. This condition help to eliminate the process of the Compton scattering from crystal to crystal.*

As far as such condition is concerned, it is enough to include in the definition the range [1, 1]:

Multiplicity of cryostat (where this crystal belongs to) is in the range: 1 – 1

8.4. 3. AGATA detector self-gate

The self-gate condition for AGATA-array detectors looks very similar. Thanks to the pulse shape analysis (PSA), these detectors know where inside the germanium crystal a detection occurred. The coordinates of this point are available, and they are then used to analyze the γ quantum path in the Ge detector crystal.

In next Figure 32 we can see that they can be used when building a self-gate.

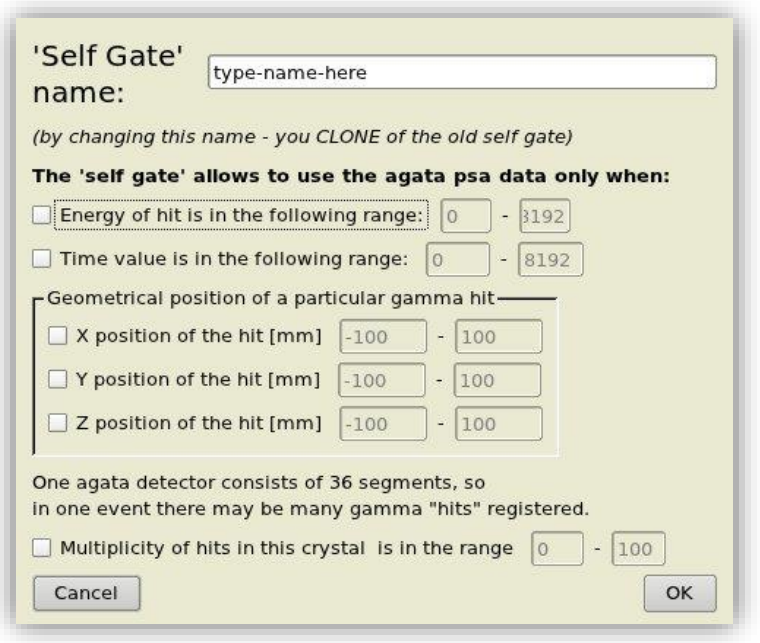


Figure 32. Defining self-gate for incrementers of the AGATA array.

In every event we have such a situation described by space coordinates (x, y, z) – relative to a given detector of the AGATA germanium system. So we can also set a self-gate condition on these coordinates.

8.4. 4. Self-gate for the RISING cluster detectors

Because the most important detectors in the RISING experiment were cluster germanium detectors, therefore the self-gate condition for these detectors is the most extensive.

Self-gate dialog offers us many "sister" variables in which we can place gates.

As the RISING clusters self-gate definition dialog box was shown on page 67, so here we will discuss individual items, in a form of a table.

✧

Every crystal of the germanium detector (while detecting a gamma quantum) provides information about its energy, its recording time (in relation to the time of the trigger triggering the event registration process).

	Name of the “sister” parameter	min	max
a)	Calibrated energy [4 MeV] value	...	...
b)	Calibrated energy [20 MeV] value	...	...
c)	Calibrated time value	...	...
d)	Understand above *Calibrated time* as a time difference between two gamma quanta.	Yes/No	
e)	Calibrated LR time	...	...
f)	Calibrated SR time	...	...
g)	(Y) Energy_cal vs. (X) **Time_cal** of this crystal are inside the following polygon ….	Name of a polygon gate	
	(Y) Energy_cal vs. (X) **LR Time_cal** of this crystal are inside the following polygon ….		
	(Y) Energy_cal vs. (X) **SR Time_cal** of this crystal are inside the following polygon ….		
h)	Theta angle of the geometry position is in a following range	...	...
i)	Phi angle of the geometry position is in a following range	...	...
j)	Theta angle between a gamma quantum and a scattered particle is in a following range	...	...
k)	Phi angle between a gamma quantum and a scattered particle is in a following range	...	...
l)	Multiplicity of the cluster (where this crystal belongs to) is in a following range	...	...
m)	BGO energy of the cluster (where this crystal belongs to) is in a following range	...	...

a) b)

In the first campaigns of RISING experiments, information from this type of detectors was read in two energy ranges: 0-4 MeV and 0-20 MeV. To preserve the ability to analyze older data, there are still both in self-gate.

e) f)

Starting from the RISING g-factor experiment campaigns, there two additional time measurements available. The so-called *Short Range* (SR) and *Long Range* (LR). Both of them are included in the self-gate definition dialog box, in items e) and f).

m)

At the bottom of the list there is an item *m)* regarding information from the BGO anti-Compton shields. In some experiments, such shields were installed on detectors to reduce the Compton background in radiation spectra.

✧

d) There is some sophisticated sorting method – available due to this option

Understand above Calibrated time as a time difference between two gamma quanta.

Selecting it causes that in the option c) one line above, the description text changes to this:

| (X time_cal – Y time_cal) | is in a range ... – ...

This option was useful when you want to create a time coincidence matrix. With this option, you can set the condition on the absolute value of the difference in the time between detection of a two gamma quanta – whose energies were selected as the X incrementer and the Y incrementer of the created matrix.

> This option was very useful in the study of nuclear isomeric states. ("RISING Campaign with a passive stopper").

Now some more details.

The point on such a matrix is incremented thanks to two x, y incrementers. Let's call them $E\gamma_x$ and $E\gamma_y$. These incrementers are used in the matrix definition. Each of these "energy" incrementers has a sister "time" incrementer. Of course, you can put a self-gate on it. Here, however, we have the possibility that:

- the $E\gamma_x$ incrementer self-gate check procedure will search for "its time" $T\gamma_x$
- then reaches the $E\gamma_y$ incrementer and finds the time associated with it $T\gamma_y$,
- and then calculate the difference of both times.

The absolute value of this time difference can be checked by the desired condition.

This option makes sense only for incrementers used in two-dimensional spectra (matrices).

Using this option assumes that a self-gate is defined and assigned to the incrementer X. The same self-gate is also assigned to the incrementer Y. When checking the fulfillment of the gate, the time value associated with the incrementer X is stored. When the fulfillment of the same gate for the Y incrementer is checked immediately afterwards – this stored time value, together with the current one – are used to calculate the time difference between both registered gamma quanta.

✧

g) Here are the further entries for the self-gate condition definition

In position g) we see the possibility of setting a polygonal gate. It is checked here whether the point with coordinates P (x, y) where x = time, y = energy, is inside the given polygon.

- Energy4MeV represents energy data,
- As time – the experimenter has three optional variables to choose from. The proper selection is done by cyclic pressing the button on which the following text will appear:

  Time_cal / LR Time_cal / SR Time_cal

Even more down, you can see a button for selecting a polygon gate. To use this option…

- at first make a $E\gamma$ vs $T\gamma$ matrix (for the selected type of time);
- then you can draw on this matrix a polygonal gate marking the area of interest;
- then, during defining this self-gate condition, you can use this polygonal gate.

✧

9. User-defined incrementers

There are many thousands of incrementers defined in the GREWARE/Spy. However, sometimes the experimenter would like to have a special, new incrementer. An incrementer which represents a value of an arithmetic expression of two other incrementers.

The simplest example is: the sum of the values of two incrementers containing information about energy losses in two detectors.

An ion passing through one TPC41 detector loses part of its energy in it and then it pass to the next TPC42 detector. The values of both energy losses are available thanks to appropriate incrementers and two corresponding spectra can be built.
However, if the experimenter would like to know the sum energy lost in both of these detectors, it would be incorrect to simply sum up the above two ***spectra****. To make this correctly – in every event we should* ***sum up the values of both of these incrementers****. This sum value should increment a spectrum.*

9. 1. Defining the user-incrementer in the GREWARE GUI program

You can define your own incrementer. Its value is will be calculated – according to one of the following expressions:

New incrementer = incrementerA @ incrementerB

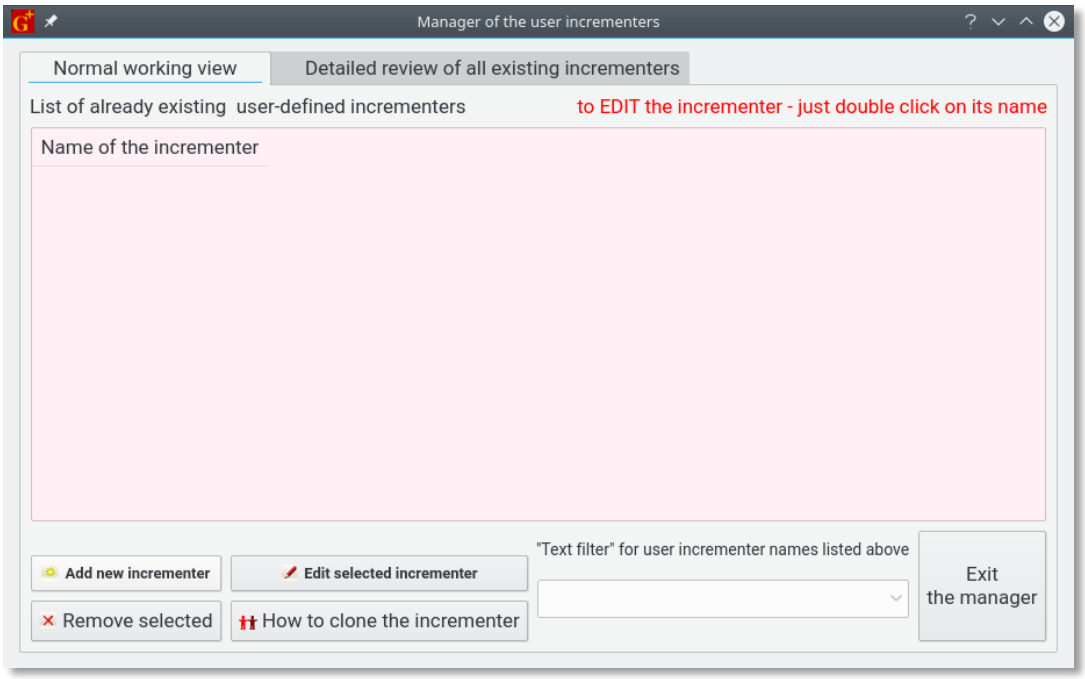
or

New incrementer = incrementerA @ numeric constant

where the symbol @ represents one of the following operators

\+ adding
\- subtraction
* multiplication
/ dividing
arithmetic average

To make a definition of your own incrementer – there is a special procedure in GEWARE. At first go to Menu Bar → Edit → "User defined incrementer manager".



The incrementer manager not only shows the list of user defined incrementers made so far; it can be used to create new incrementers as well as to modify existing ones.

To create a new incrementer, you need to launch the wizard, which will guide you through the stages of creation. You do this by pressing the button called Add new incrementer.

At first you need to invent a name for your incrementer. It must begin with the prefix "user_".

After this, you can construct an expression in which you use:

- names of two other incrementers

or

- name of one (other) incrementer and some constant value.

The result of this expression will be a value of your new incrementer. It will be calculated after every event.

Look at a following Figure 32. To make the definition process more friendly – the wizard uses the terms "expression_green" and "expression_blue".

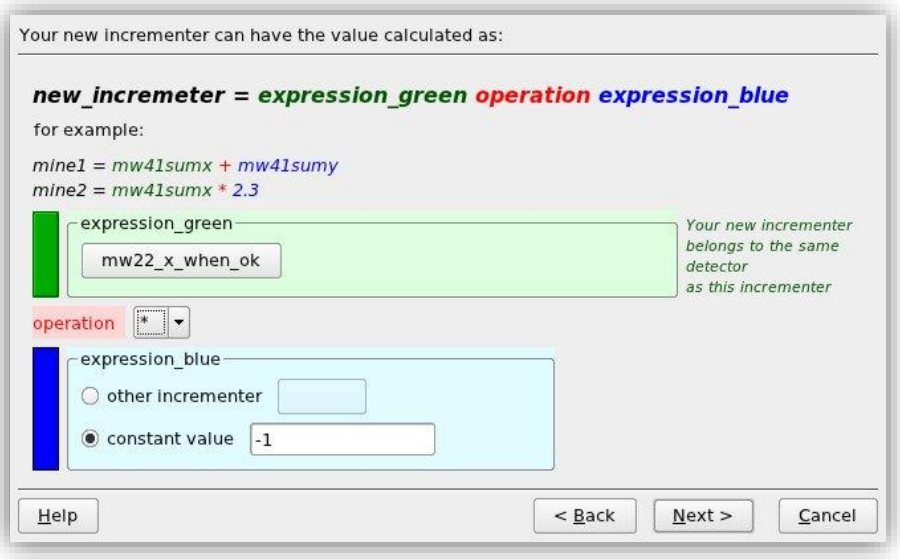


Figure 33. A view of the incrementer wizard page. Here the user can define the expression which will be used every event to calculate a current value of the new incrementer.

Here are the meanings of each term:

expression_green

- it can be any other incrementer selected from the list of available incrementers,

operation

- the checkbox offers (so far) five possible operators – addition, subtraction, multiplication, division, arithmetic mean,

expression_blue

- there are two possibilities here; either it is a some other incrementer or a fixed real number.

Simplest example of an user-incrementer

The Spy offers an incrementer representing in the position-sensitive detector the horizontal x coordinate measured by the mw22 multiwire chamber

```
mw22_x_when_good
```

We can use it when creating the incrementer in the following expression

```
value = mw22_x_when_good * (-1)
```

Thanks to this we get the value of x with the opposite number. The XY graph drawn according to this value will be a mirror image.

Memory persistence

After completing the incrementer wizard, the details of the definition of the new user-incrementer are saved to disk as a file. The definition is therefore permanent and from now on it can be used many times in the future.

The file with the incrementer definition contains full information about the expression. That is, if you primarily decided that the second operand should be a numeric constant, then its written value on the disk. If you change your mind and decide that the second operand is to be an incrementer, then this ("old") constant is also written to the disk anyway (although it will not be used). Thanks to this, if the you change your mind again and return to a numerical constant – the older value is waiting for you.

9. 2. Some examples of user-defined incrementers

9.2. 1. Energy deposit – example

In some RISING experiments, the TPC (Time Projection Chamber) detectors were used. The TPC detectors, in addition to information about time, provide information about the energy loss of the ion passing through it.

In the fragment separator, two such detectors tpc41 and tpc42 were standing close to each other.

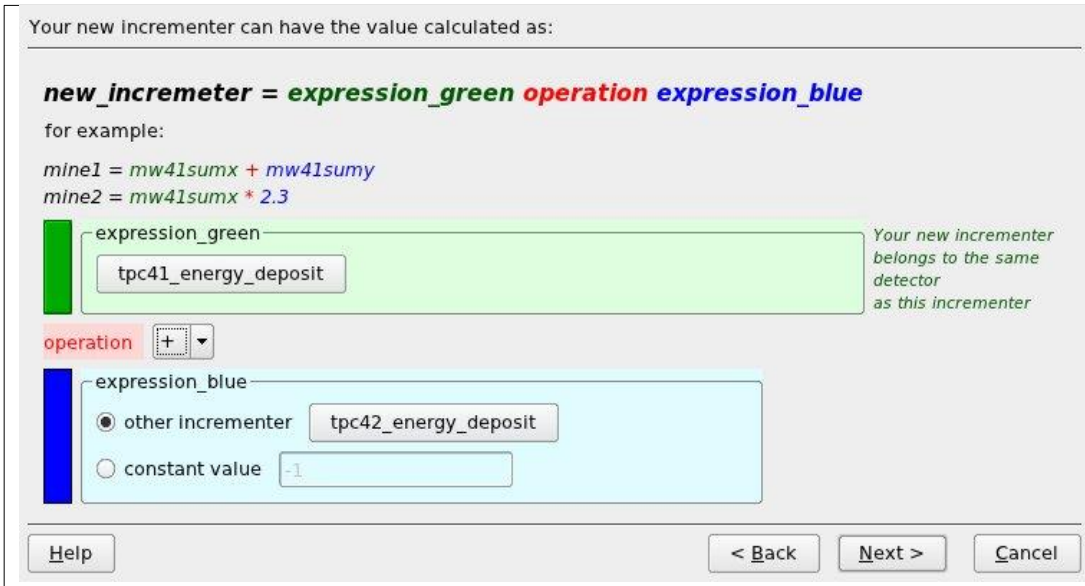


Figure 34. Defining a new incrementer, which represents a total energy loss in two neighboring detectors tpc41, tpc42.

Each of the objects representing a single TPC detector provides an incrementer containing information about energy loss (deposit energy) in a given detector. It may happen that the experimenter would like to see the spectrum illustrating the **sum of energy loss** in two TPC detectors. Although there is no such incrementer – the user can create it himself.

The definition of such an incrementer is shown in picture above.

With an incrementer defined this way, it can be used to build a spectrum in the usual way. The result is shown in a following Figure 35.

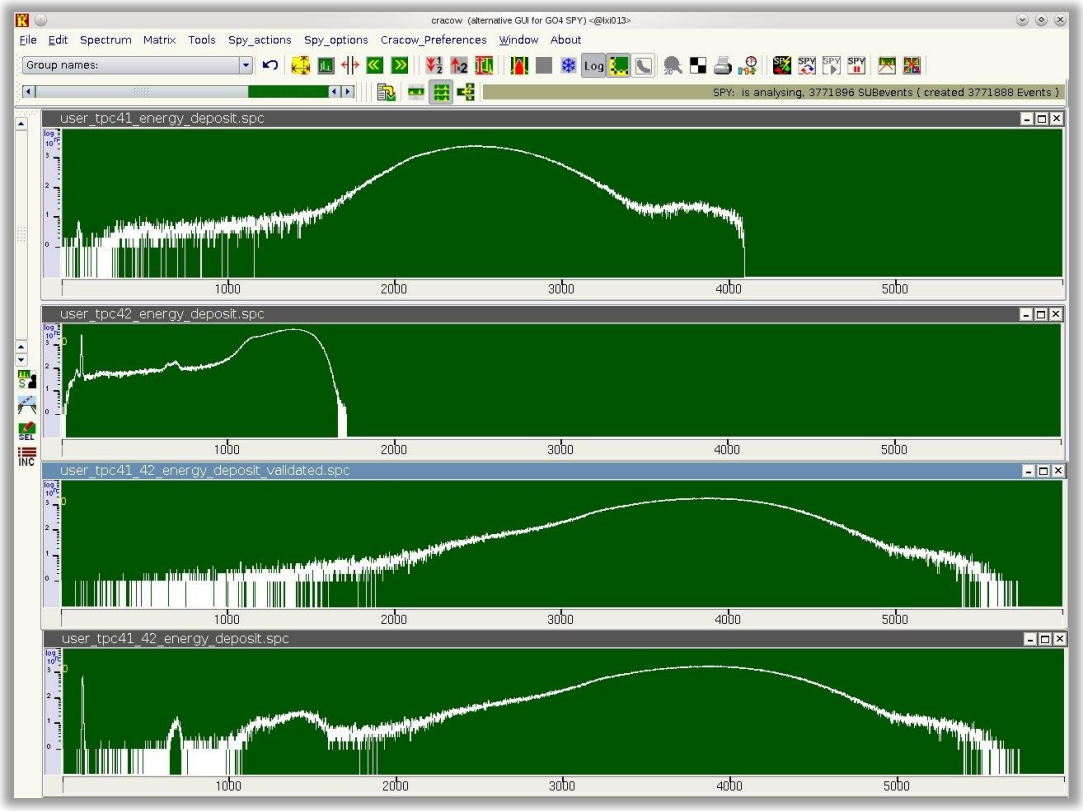


Figure 35. Four spectra shown on the GREWARE-GUI screen.

The first was built using the tpc41_energy_deposit_when_ok incrementer.

The second using the tpc42_energy_deposit_when_ok incrementer

The third spectrum uses the **newly created incrementer**, whose value is calculated as the sum of the values of both of the above incrementers. Obtaining this spectrum was the goal of the experimenter.

The fourth spectrum is an interesting fact. It uses a poorly defined incrementer, which creates the sum of the incrementers named tpc41_energy_deposit and tpc42_energy_deposit. **Note the lack of "when_ok" in their names**. This means that these components do not have validators. If so, the resulting user incrementer also has no validator.

How will this affect the spectrum incremented with such an incrementer?

In a situation where one of the TPC detectors in the event did not fired, its incrementer (without validator) has a value 0. The fourth spectrum is then incremented by the sum of this value of zero and the value of the other incrementer (probably non-zero). The structure around channels 600 and 1200 - 1700 are just traces of the situation when only the tpc42 detector worked and the tpc41 detector did not.

The trace of the opposite situation, i.e. the tcp42 detector does not work, and the tpc41 operation - there is a slight (due to logarithmic scale) bulge around the 2200 -2500 channels.

9.2. 2. Average flight time – example

In the FRS fragment separator, the time of flight of a projectile is measured between the sci21 scintillator and the sci41 scintillator – using a time-to-amplitude converter (TDC).

This information is used to calculate the A/q value of the ion flying on this segment.

The start and stop signals for this converter come from photomultipliers working in the mentioned scintillation detectors. Due to the fact that two photomultipliers ("left" and "right") work in each of these detectors, therefore the time measurement data is duplicated.

The acquisition system therefore provides the flight time between two "left" photomultipliers (tof_21_41_LL) and the flight time between two "right" photomultipliers (tof_21_41_RR). Both of

these raw signals in the analysis program are calibrated, and the incrementers providing such values are available to the user as:

tof_21_41_tof_LL_cal
tof_21_41_tof_RR_cal

Using the user-defined incrementer discussed in this chapter, the experimenter can create an incrementer that represents the arithmetic mean of both times. A following figure illustrates the definition of such an incrementer.

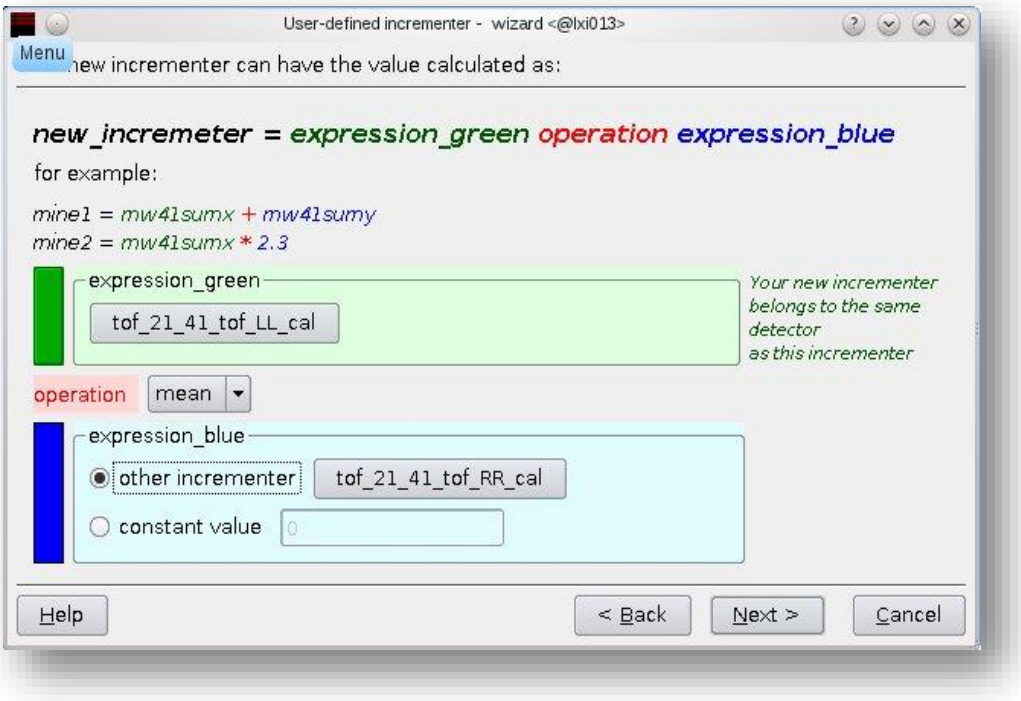


Figure 36. Definition of the incrementer representing arithmetic mean value of the time of flight between scintillator sci21 and scintillator sci41.

KOTWICA

The new incrementer defined in this way can be used to create the user spectrum. A following Figure 37 shows the result. See a description under the figure.

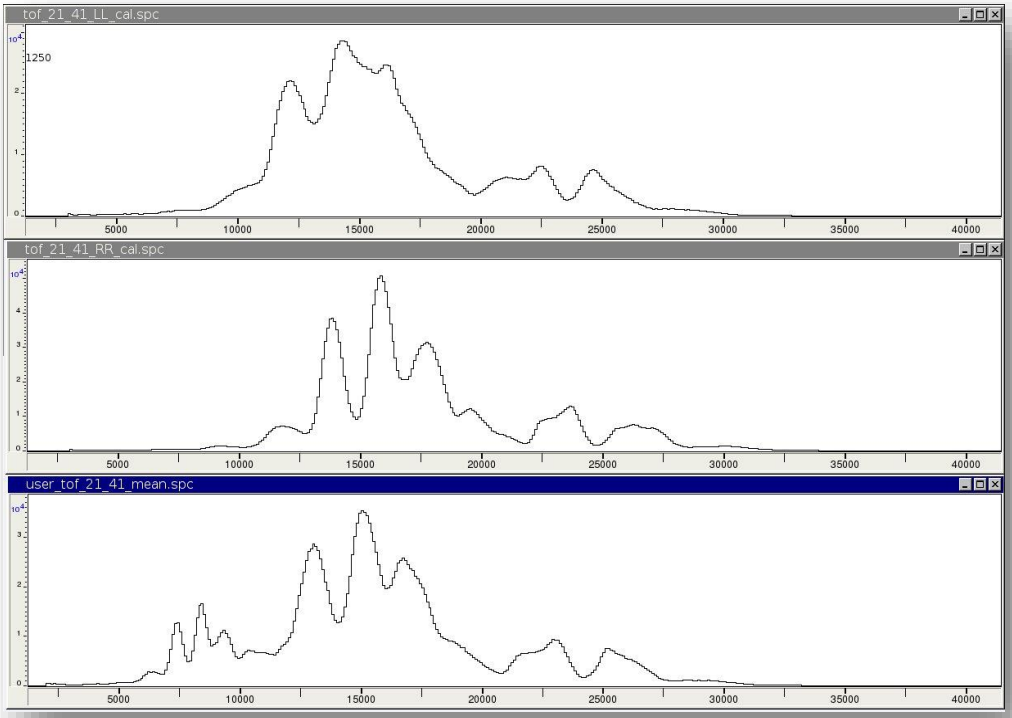


Figure 37. An example of using the arithmetic mean of two incrementers. The first spectrum illustrates the time of flight measured using start and stop signals from the "Right" photomultipliers of both scintillators, and the second spectrum shows the "Left" photomultipliers.

The spectrum at the bottom was made with a new user-defined incrementer. It was defined to calculate an arithmetic mean of the above two incrementers carrying information about ion transit times between the scintillators sci21 and sci41.

(The structure around channels 7500-9000 assures us that the output incrementers do not have validators. If the incrementers with validators were used – this structure would not appear).

KOTWICA

✧

If you need to construct your own incrementer, perhaps it is a good idea to tell this to the Spy programmer (Jurek?). He may include your idea as a future default incrementer.

✧ ✧ ✧