REST-for-Physics Framework

An overview of REST-for-Physics software project status

20.06.2023 - Javier Galan - javier.galan@unizar.es

Centro de Astropartículas y Física de Altas Energías
Universidad Zaragoza

REST-for-Physics contextualization

REST-for-Physics is **not best represented by the simulation** category..

Although it is true that in REST-for-Physics we can:

1. Perform Geant4 simulations that are initialized using our framework setup, and produce the results in a a format that can be further processed inside the framework.

2. Include basic detector response algorithms that allow us to reproduce what we observe in our detectors, without major complexity.

REST-for-Physics is an event data processing framework aiming for **combined experimental and MC truth** event processing.

REST-for-Physics motivation.

It is born to **cover the needs** of experimental data taking programs, such as:

1. Detector background and signal simulation including detector response.
2. Detector rawdata processing and event reconstruction.

But also to **provide**:

1. Physical meaning to stored data.
2. Prototyping for common tasks and event data processing.

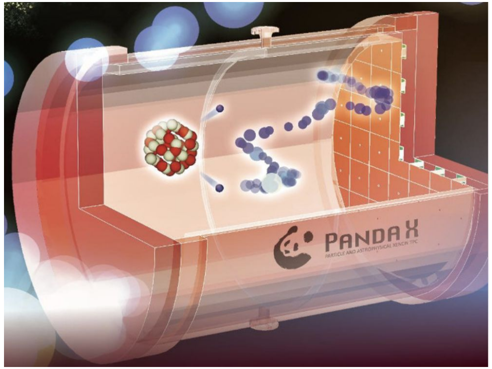
And to **assure**:

1. Long term coherent access to the data upon a unified event format scheme.
2. Data reproducibility and traceability.

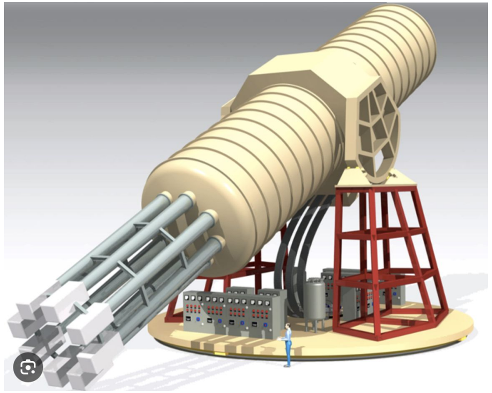
For which experiments needs?

REST-for-Physics is being exploited by different experiments.

- CAST
- PandaX-III
- TREX-DM



Presently REST-for-Physics **developments benefit from an ERC** funding program for IAXO.



Other experiments have claimed interest on REST-for-Physics: **Liquido** and **CONUS+**.

Introducing REST-for-Physics

- The REST-for-Physics (Rare Event Searches Toolkit) Framework is a collaborative software effort that provides common tools for:
    - acquisition,
    - simulation,
    - data analysis

- It was originally designed to work with data of gaseous Time Projection Chambers (TPCs), although by construction is not limited to.

- It is mainly written in C++ and it is fully integrated with ROOT I/O interface.

- It was born at the University of Zaragoza and it is strongly used in academia for undergraduate and postgraduate studies.

https://rest-for-physics.github.io/

REST C++ class initialization through config files

Any REST class inheriting from the TRestMetadata class allows to identify C++ data members with XML parameters, which allows us to produce XML configuration files for C++ class initialization.

C++ header

Where TRestAxionSolarFlux inherits from TRestMetadata

```
//! A metadata class to load tabulated solar axion fluxes. Mass
class TRestAxionSolarQCDFlux : public TRestAxionSolarFlux {
   private:
    /// The filename containning the solar flux table with conti
    std::string fFluxDataFile = "";   //<

    /// The filename containning the solar flux spectra for monoc
    std::string fFluxSptFile = "";   //<

    /// It will be used when loading `.flux` files to define the
    Double_t fBinSize = 0;   //<

    /// It will be used when loading `.flux` files to define the
    Double_t fPeakSigma = 0;   //<
```

```
<TRestAxionSolarQCDFlux name="LennertHoofABC" verboseLevel="warning" >
    <parameter name="couplingType" value="g_ae"/>
    <parameter name="couplingStrength" value="1.e-13"/>
    <parameter name="fluxDataFile" value="ABC_LennertHoof_202203.N200f"/>
    <parameter name="fluxSptFile" value="ABC_LennertHoof_202203.spt"/>
    <parameter name="seed" value="137" />
</TRestAxionSolarQCDFlux>
```

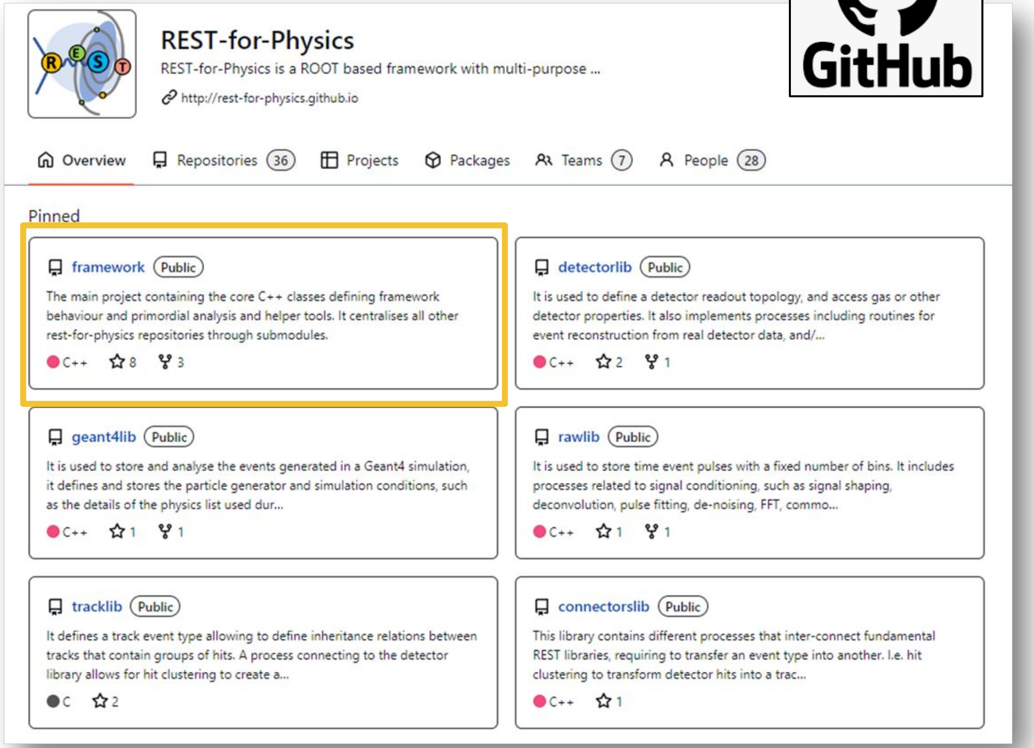
RML config file

Minimal coding required. The parameter names automatically identify with the C++ data members using the naming convention.

C++		RML
fVarName	⟷	varName

The full REST-for-Physics project is splitted in different Github repositories

- **Main project**
  - Framework
- **Libraries** for montecarlo and detector data processing
  - Rawlib / Geant4lib
  - Detectorlib / Tracklib
  - Axionlib
  - Connectorslib
- **Packages** that exploit REST libraries
  - restG4
  - restSQL
  - …

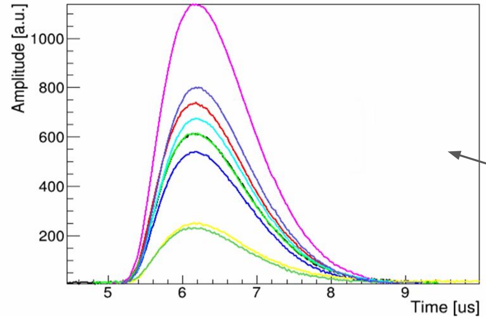


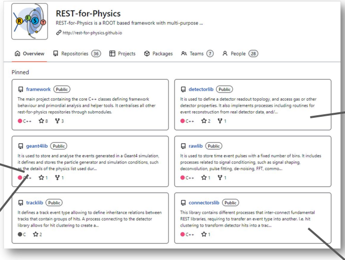
REST-for-Physics event types

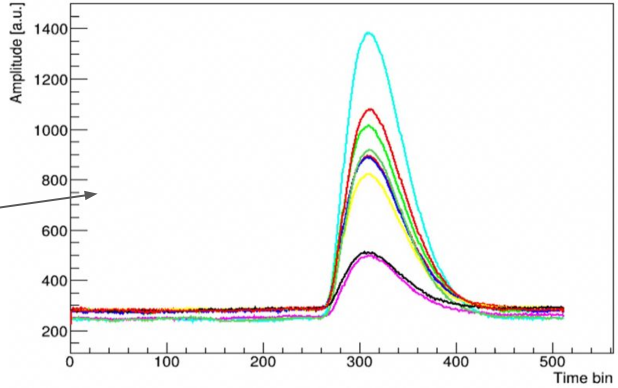
Each **event library** defines <u>one event type</u> (only exception detectorlib)

TRestRawSignalEvent

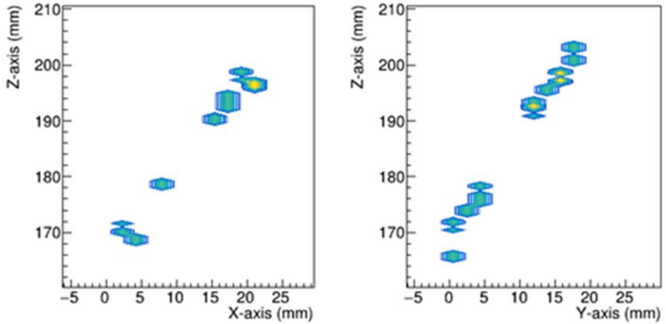
TRestDetectorSignalEvent



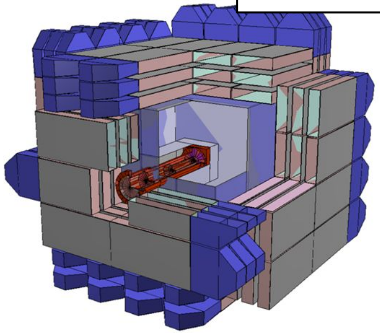


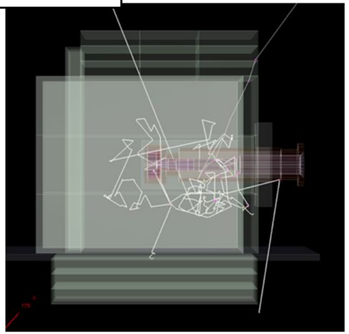


TRestDetectorHitsEven
t



TRestGeant4 Event



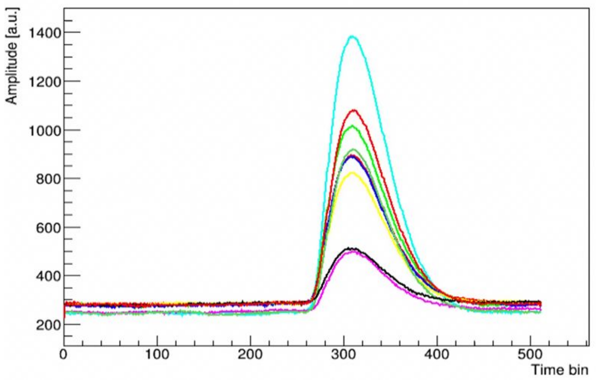


Data class to store a set of pulses.

TRestRawSignal is not only storage, it also implements methods that operate on the waveforms, such as retrieving risetime or pulse amplitude.

TRestEvent
    Int_t fId;
    TTimeStamp fTimeStamp;
    ....

TRestRawSignalEvent
    std::vector <TRestRawSignal> fSignal;

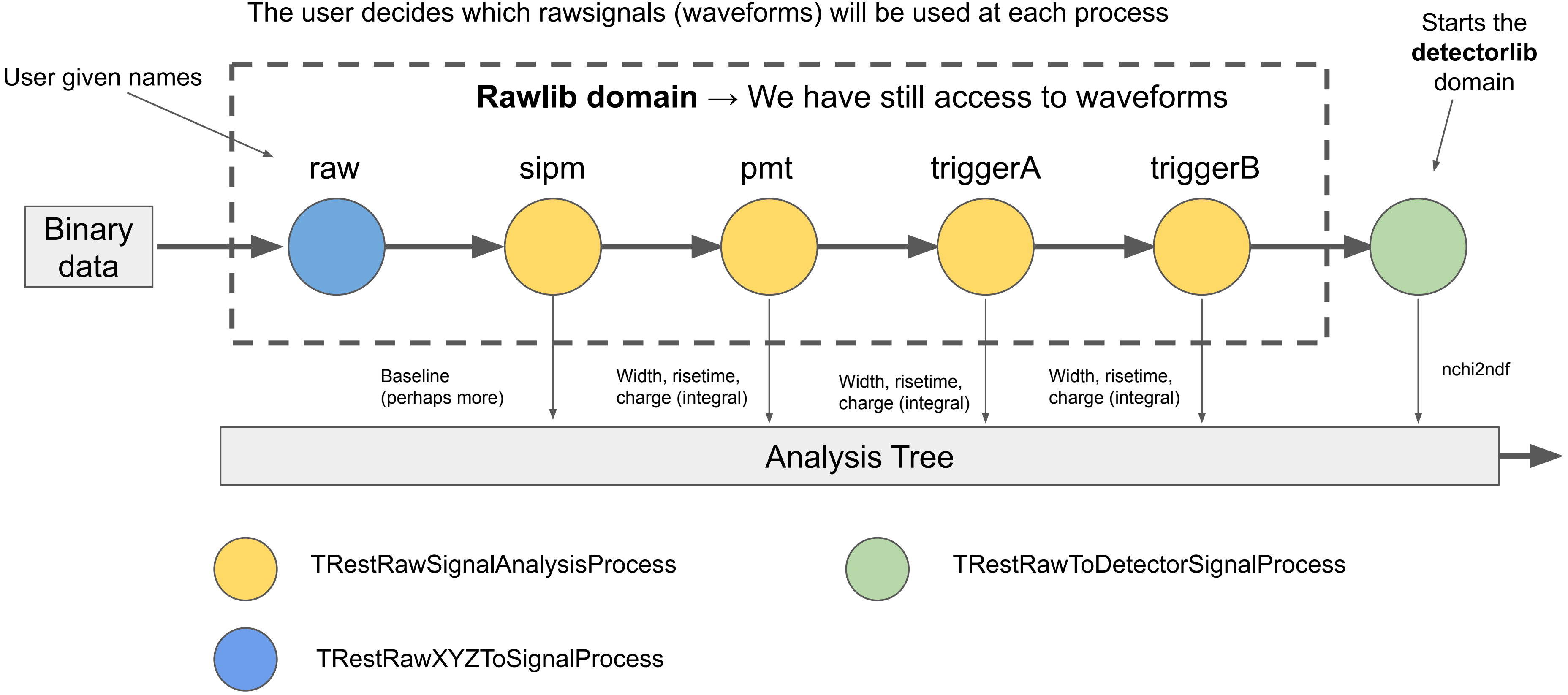


TRestRawSignal
    Int_t fSignalId;
    std::vector <Short_t> fSignalData;

A **TRestRawSignal** contains a given number of samples with fixed sampling time.

Values such as the sampling time is stored as a metadata member at a dedicated metadata class

The user decides which rawsignals (waveforms) will be used at each process



Contents inside the analysis tree

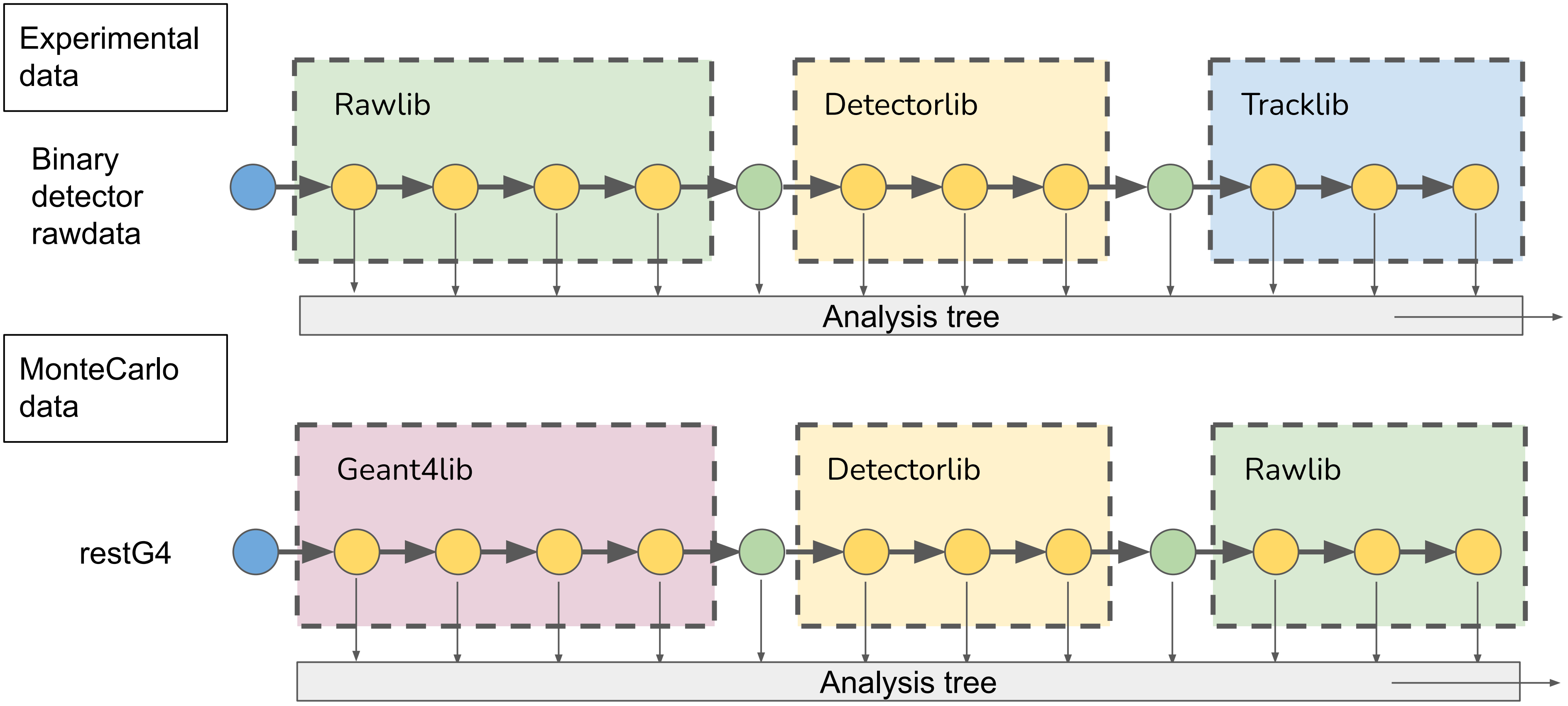
The **TRestAnalysisTree** is a ROOT TTree that we have extended to add extra functionalities and methods.

Each observable root name tells us the process that generated it.

In this example all processes were generated by a unique process named **rawAna**.

```
root [2] run0->PrintObservables()
Entry : 1
> Event ID : 3451923
> Event Time : 1664952185.90144
> Event Tag :
-----------------------------------------
Observable : rawAna_pointsoverthres_map       Value : {[120:65],[123:68],[180:0],[181:75
Observable : rawAna_risetime_map              Value : {[120:18],[123:20],[180:0],[181:23
Observable : rawAna_peak_time_map             Value : {[120:212],[123:213],[180:216],[18
Observable : rawAna_baseline_map              Value : {[120:264.638],[123:257],[180:263.
Observable : rawAna_baselinesigma_map         Value : {[120:28.9341],[123:35.6832],[180:
Observable : rawAna_max_amplitude_map         Value : {[120:529.362],[123:754],[180:379.
Observable : rawAna_thr_integral_map          Value : {[120:21815.5],[123:31511],[180:0]
Observable : rawAna_SaturatedChannelID        Value : {}
Observable : rawAna_BaseLineMean              Value : 261.038
Observable : rawAna_BaseLineSigmaMean         Value : 20.0425
Observable : rawAna_TimeBinsLength            Value : 512
Observable : rawAna_NumberOfSignals           Value : 10
Observable : rawAna_NumberOfGoodSignals       Value : 5
Observable : rawAna_FullIntegral              Value : 290788
Observable : rawAna_ThresholdIntegral         Value : 269656
Observable : rawAna_RiseSlopeAvg              Value : 20509.8
Observable : rawAna_SlopeIntegral             Value : 102549
Observable : rawAna_RateOfChangeAvg           Value : 0.2
Observable : rawAna_RiseTimeAvg               Value : 22
Observable : rawAna_TripleMaxIntegral         Value : 18888
Observable : rawAna_IntegralBalance           Value : 0.0377064
Observable : rawAna_AmplitudeIntegralRatio    Value : 42.7526
Observable : rawAna_MinPeakAmplitude          Value : 529.362
Observable : rawAna_MaxPeakAmplitude          Value : 2000.96
Observable : rawAna_PeakAmplitudeIntegral     Value : 6307.35
Observable : rawAna_MinEventValue             Value : -85
Observable : rawAna_AmplitudeRatio            Value : 3.15216
Observable : rawAna_MaxPeakTime               Value : 216
Observable : rawAna_MinPeakTime               Value : 212
Observable : rawAna_MaxPeakTimeDelay          Value : 4
Observable : rawAna_AveragePeakTime           Value : 214.2
Observable : TREXsides_tagId                  Value : 1
root [3]
```

REST-for-Physics processing stages



Geant4 simulations using restG4

All we need is the description of the simulation conditions (generator, active volumes, etc) and a GDML geometry.

See API documentation

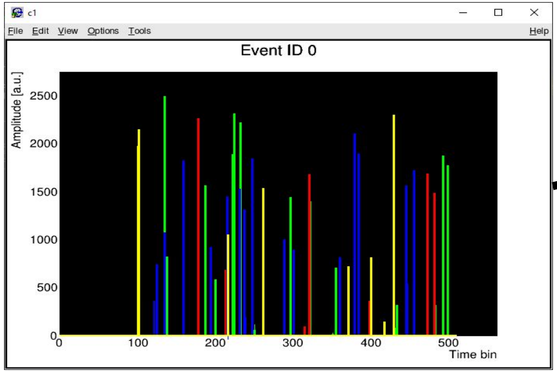
```
<TRestGeant4Metadata>
    <parameter name="gdmlFile" value="geometry/setup.gdml"/>
    <parameter name="seed" value="137"/>
    <parameter name="nRequestedEntries" value="1000000"/>
    <parameter name="saveAllEvents" value="true"/>

    <generator type="cosmic">
        <source particle="geantino">
            <energy type="formula2" name="CosmicMuons" nPoints="2000"/>
            <angular type="formula2" direction="(0.1,-0.5,1.25)" nPoints="500"/>
        </source>
    </generator>

    <detector>
        <volume name="gasVolume" sensitive="true" maxStepSize="1mm"/>
    </detector>

</TRestGeant4Metadata>
```

The data simulated with REST-for-Physics can be directly plugged into the processing to obtain the detector response.

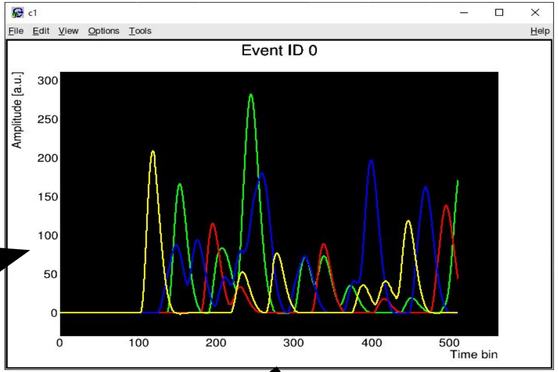
List of examples.

- 01.NLDBD
- 02.TREXDM
- 03.Fluorescence
- 04.MuonScan
- 05.PandaXIII
- 06.IonRecoils
- 07.FullChainDecay
- 08.Alphas
- 09.Pb210_Shield
- 10.Geometries
- 11.Xrays
- 12.Generators
- 13.IAXO
- 14.DetectorResponse

The data simulated with REST-for-Physics can be directly plugged into the processing to obtain the detector response.



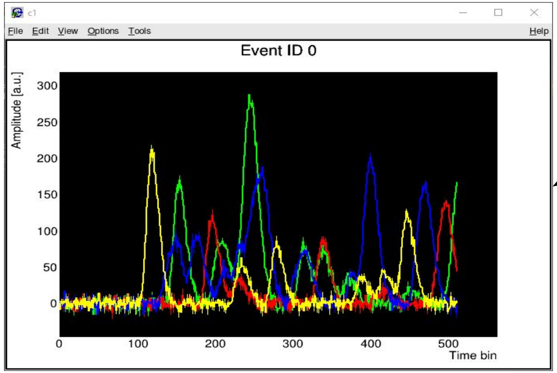
TRestRawSignal
Shaping
Process



Simulation

TRestRawSignal
AddNoise
Process

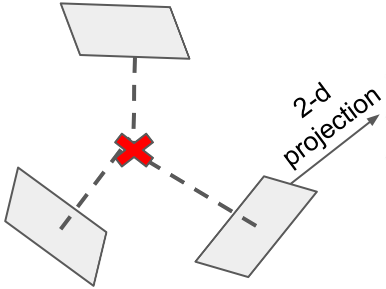
Add random noise to emulate experimental data

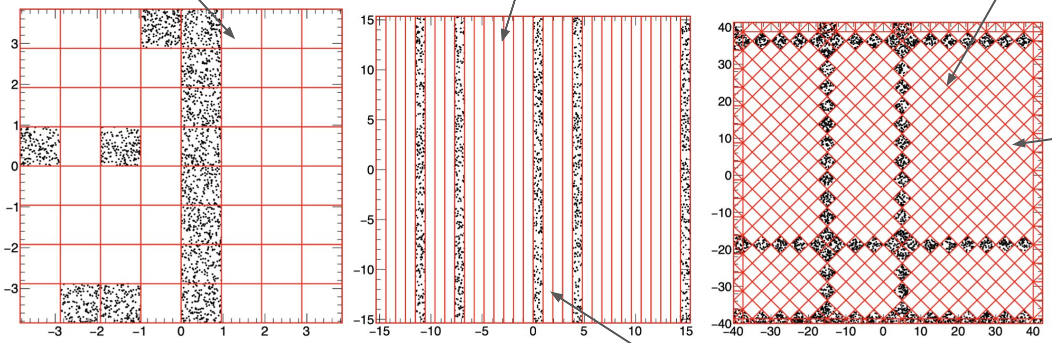


In the detectorlib domain we need to build a readout (see TRestDetectorReadout documentation) that identifies the waveform channel id with a physical detector readout channel.

TRestDetectorReadout allows building any readout topology. See the basic-readouts examples repository.

A readout is made of any number of readout planes in a **3-dimensional** space..





A **readout channel** is identified with the waveform. It has a channel id and a readout id → decoding.

IMPORTANT. There is a mapping that optimizes pixel identification!

Contents of a REST processed file

EventTree (TTree) for event storage

TRestRun object for run management

```
root [0] .ls
TFile**         R01185_00000_SignalToTrack_Ar2Iso_CalibWith9VetosFe55_cristina_2.3.4_vetoTh500_vetoTime190300.root
 TFile*         R01185_00000_SignalToTrack_Ar2Iso_CalibWith9VetosFe55_cristina_2.3.4_vetoTh500_vetoTime190300.root
  OBJ: TTree     EventTree       TRestTrackEventTree : 0 at: 0x55cfeccd0f80
  OBJ: TRestAnalysisTree         AnalysisTree    REST Process Analysis Tree : 0 at: 0x55cfecf8f500
  KEY: TRestRun IAXOD0-2021:2    IAXOD0 2021 data taking
  KEY: TRestProcessRunner        Signals;1       Signal to track analysis
  KEY: TRestDetectorSignalToHitsProcess signalToHits;1  A Signal To Hits reconstruction template.
  KEY: TRestDetectorHitsAnalysisProcess hitsAna;1       Hits analysis template
  KEY: TRestDetectorHitsGaussAnalysisProcess    hitsAnaGauss;1  defaultTitle
  KEY: TRestDetectorHitsToTrackProcess  hitsToTrack;1
  KEY: TRestTrackAnalysisProcess        tckAna;1        Track analysis template
  KEY: TTree     EventTree;1     TRestTrackEventTree
  KEY: TRestAnalysisTree         AnalysisTree;1  REST Process Analysis Tree
  KEY: TRestProcessRunner        RawSignals;1    Raw processing and analysis
  KEY: TRestRawMultiFEMINOSToSignalProcess      virtualDAQ;1    defaultTitle
  KEY: TRestRawVetoAnalysisProcess      veto;1  defaultTitle
  KEY: TRestRawSignalAnalysisProcess    sAna;1
  KEY: TRestRawZeroSuppresionProcess    zS;1
  KEY: TRestDetectorSignalChannelActivityProcess        chActivity;1    Channel activity process
  KEY: TRestDetectorReadout      iaxo_readout;2  IAXO-D0 readout 0.5 mm-Pitch 120+120 channels
root [1]
```

AnalysisTree (TRestAnalysisTree) for analysis observables

Analysis process metadata for traceability

REST-for-Physics code and data versioning

Versioning based on GitHub commit numbers.

Loading a particular version of REST-for-Physics

```
(base) -bash-4.2$ source /nfs/dust/iaxo/group/software/rest/v2.3.15/thisREST.sh
switching to REST installed in /nfs/dust/iaxo/group/software/rest/v2.3.15
  *****************************************************************************
  W E L C O M E   to  R E S T

  Commit  : 9bfed7c6 (2022-12-30 16:20:44 +0100)
  Branch/Version : master/v2.3.15
  Compilation date : 2022-12-30 16:34

  Latest release: : v2.3.15 - David R. Nygren - Fri 30 Dec

  Official release : Yes
  Clean state : Yes

  Installed at : /nfs/dust/iaxo/group/software/rest/v2.3.15

  REST-for-Physics site : rest-for-physics.github.io

  Remember that REST is made by physicists for physic
  who are supposed to toil and suffer till they becon
  ****************************************************
```

Git info

Release name

Flags identifying official release

Data inspection of a metadata class inside a REST processes file.

```
root [0] md0_xmm->PrintMetadata()
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
||                     TRestAxionTrueWolterOptics content                         ||
Config file : /afs/desy.de/user/j/jgalan/work/iaxo-simulations/montecarlo/signal/sensitivity/Baby...
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
||                              Name : xmm                                        ||
||              Title : Default TRestAxionTrueWolterOptics                        ||
||                       REST Version : 2.3.15                                    ||
||                    REST Official release: Yes                                  ||
||                         Clean state: Yes                                       ||
||                      REST Commit : 9bfed7c6                                    ||
||                    REST Library version : 1.1                                  ||
-----------------------------------------------------------------------------------
||                     - Optics file : XMM.trueWolter                             ||
```

Data processing. Multi-file analysis combination.

When generating or processing data we will usually produce a number of files that need to be combined later on ...

Run_XYZ_01.root

Run_XYZ_02.root

Run_XYZ_03.root

Run_XYZ_04.root

Run_XYZ_05.root

Run_XYZ_06.root

Run_XYZ_07.root

→

Only the analysis tree is merged into a dataset

Existing tools

- TRestAnalysisPlot
- TRestMetadataPlot
- TRestDataSet
- TRestDataSetPlot

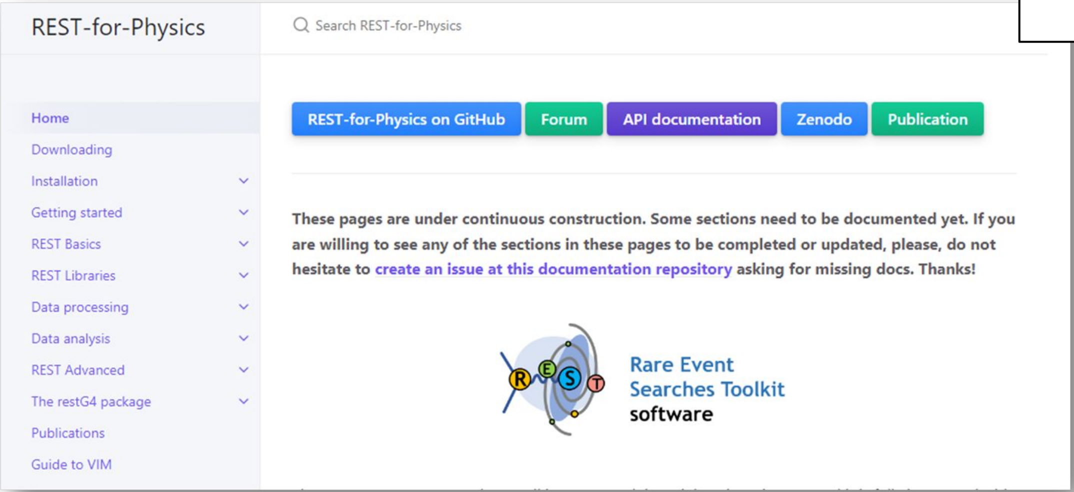
→

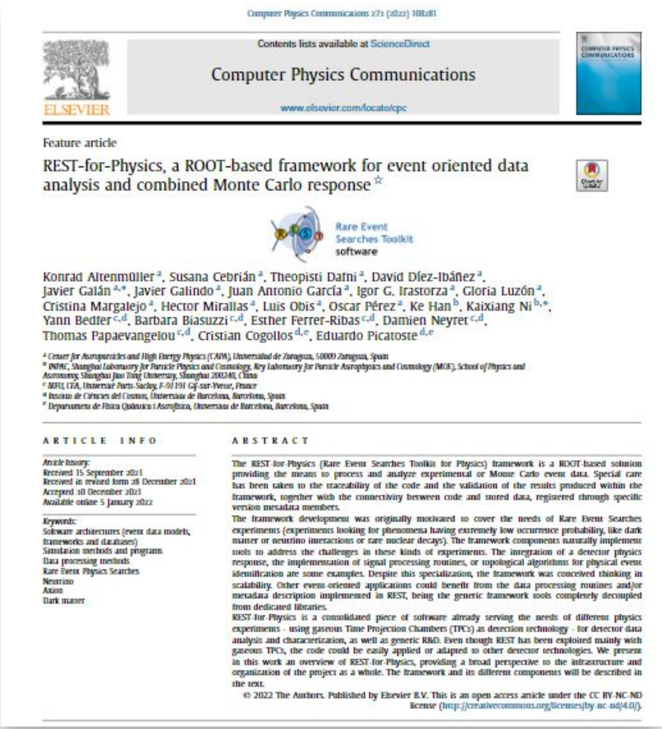
Used for

- Systematic plot generation
- Metadata correlation between different runs
- Metadata time evolution
- Dataset generation
- Export datasets (txt/SQL/tree/...)

Where to find more info: Main site and publication

https://rest-for-physics.github.io/





Computer Physics Communications 273 (2022) 108281

Contents lists available at ScienceDirect

Computer Physics Communications

www.elsevier.com/locate/cpc

Feature article

REST-for-Physics, a ROOT-based framework for event oriented data analysis and combined Monte Carlo response ☆

Konrad Altenmüller [a], Susana Cebrián [a], Theopisti Dafni [a], David Díez-Ibáñez [a], Javier Galán [a,*], Javier Galindo [a], Juan Antonio García [a], Igor G. Irastorza [a], Gloria Luzón [a], Cristina Margalejo [a], Hector Mirallas [a], Luis Obis [a], Oscar Pérez [a], Ke Han [b], Kaixiang Ni [b,*], Yann Bedfer [c,d], Barbara Biasuzzi [c,d], Esther Ferrer-Ribas [c,d], Damien Neyret [c,d], Thomas Papaevangelou [c,d], Cristian Cogollos [d,e], Eduardo Picatoste [d,e]

[a] Center for Astroparticles and High Energy Physics (CAPA), Universidad de Zaragoza, 50009 Zaragoza, Spain
[b] INPAC, Shanghai Laboratory for Particle Physics and Cosmology, Key Laboratory for Particle Astrophysics and Cosmology (MOE), School of Physics and Astronomy, Shanghai Jiao Tong University, Shanghai 200240, China
[c] IRFU, CEA, Université Paris-Saclay, F-91191 Gif-sur-Yvette, France
[d] Institut de Ciències del Cosmos, Universitat de Barcelona, Barcelona, Spain
[e] Departament de Física Quàntica i Astrofísica, Universitat de Barcelona, Barcelona, Spain

ARTICLE INFO

Article history:
Received 15 September 2021
Received in revised form 28 December 2021
Accepted 30 December 2021
Available online 5 January 2022

Keywords:
Software architectures (event data models, frameworks and databases)
Simulation methods and programs
Data processing methods
Rare Event Physics Searches
Neutrino
Axion
Dark matter

ABSTRACT

The REST-for-Physics (Rare Event Searches Toolkit for Physics) framework is a ROOT-based solution providing the means to process and analyze experimental or Monte Carlo event data. Special care has been taken to the traceability of the code and the validation of the results produced within the framework, together with the connectivity between code and stored data, registered through specific version metadata members.

The framework development was originally motivated to cover the needs of Rare Event Searches experiments (experiments looking for phenomena having extremely low occurrence probability, like dark matter or neutrino interactions or rare nuclear decays). The framework components naturally implement tools to address the challenges in these kinds of experiments. The integration of a detector physics response, the implementation of signal processing routines, or topological algorithms for physical event identification are some examples. Despite this specialization, the framework was conceived thinking in scalability. Other event-oriented applications could benefit from the data processing routines and/or metadata description implemented in REST, being the generic framework tools completely decoupled from dedicated libraries.

REST-for-Physics is a consolidated piece of software already serving the needs of different physics experiments - using gaseous Time Projection Chambers (TPCs) as detection technology - for detector data analysis and characterization, as well as generic R&D. Even though REST has been exploited mainly with gaseous TPCs, the code could be easily applied or adapted to other detector technologies. We present in this work an overview of REST-for-Physics, providing a broad perspective to the infrastructure and organization of the project as a whole. The framework and its different components will be described in the text.

REST-for-Physics publication:

https://doi.org/10.1016/j.cpc.2021.108281

CAPA

REST-for-Physics school organized at University of Zaragoza at the beginning of this year.
All materials are public allowing to follow the school offline. See schedule with corresponding GitHub tutorials.

The REST-for-Physics (Rare Event Searches Toolkit) Framework is a collaborative software effort that provides common tools for acquisition, simulation, and data analysis of gaseous Time Projection Chambers (TPCs). REST-for-Physics was conceived at the University of Zaragoza and it is intensively used in academia by undergraduate, master and PhDs students for thesis preparation. It is also used for generic R&D, and it has been adopted by experiments like IAXO, TREX-DM or PandaX-III to assist on the data processing and storage of official experimental data.

This school will provide a general overview of the different capabilities of the framework through interactive sessions. During the course we will go through basic examples that will allow us to reproduce some of the common tasks performed during data processing, storage and analysis.

In order to be able to follow the course the participants are required to have a basic knowledge of programming languages, such as python or c, certain programming experience (having written your own pieces of code) is also highly recommended. REST-for-Physics is written in C++, and therefore having previous knowledge of the basic concepts of C++ is mandatory.

If you have previous coding experience in C or python, it will suffice to study the main C++ concepts at the following NIST C++ course for scientists.

REST-for-Physics uses ROOT and Geant4 packages. Previous knowledge of those packages is an advantage to take maximum profit from this course.

CAPA | Universidad Zaragoza | IAXO | erc European Research Council

School exercises are available at : https://github.com/rest-for-physics/rest-school/

REST-for-Physics Tr... | REST-for-Physics Tr...

Participants

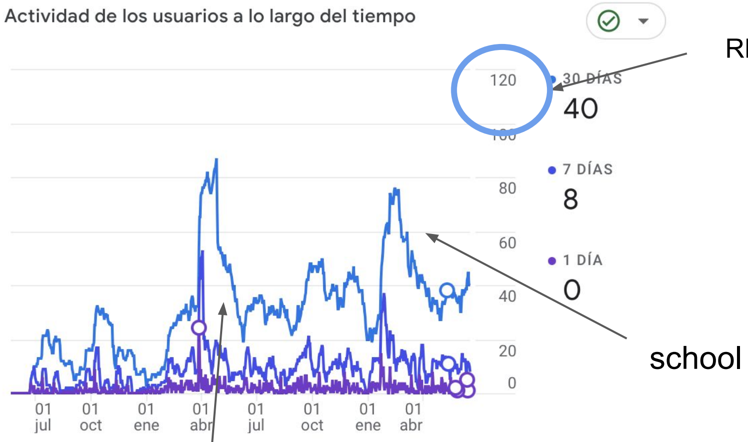
A Alfonso Yubero Navarro | A Ana Quintana | A Andrea Rubio | A Angelica Bravo Bohorquez | A asmaa aboulhorma
C Carlos Pobes | C Carmen Labiano | C Cloé Girard-Carillo | D Daniel Heuchel | D Dhiraj Gupta

REST-for-Physics statistics

Instantaneous rest-for-physics.github.io website activity

2021 2022 2023

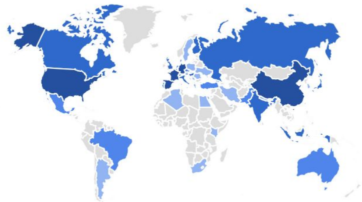
Averaged rest-for-physics.github.io website activity



publication

REST-for-Physics users worldwide

Usuarios por País



PAÍS	USUARIOS
Spain	165
China	154
France	125
Germany	75
United States	55
Italy	18
United Kingdom	16

API Documentation unique users per country

	País	Usuarios	Usuarios nuevos	Sesiones con interacción	Porcentaje de interacciones	Sesiones con interacción por usuario
		320 100 % respecto al total	316 100 % respecto al total	1.096 100 % respecto al total	56 % Media 0 %	3,43 Media 0 %
1	Spain	139	129	623	59,22 %	4,48
2	France	72	62	375	57,87 %	5,21
3	United States	33	33	4	10,53 %	0,12
4	Germany	20	16	30	29,13 %	1,50
5	China	19	18	13	46,43 %	0,68
6	Japan	6	6	3	50 %	0,50
7	United Kingdom	6	3	12	100 %	2,00
8	Italy	5	5	6	50 %	1,20
9	India	4	4	1	25 %	0,25
10	Switzerland	4	4	4	100 %	1,00

GitHub user documentation unique users per country

		652 100 % respecto al total	647 100 % respecto al total	860 100 % respecto al total	48,13 % Media 0 %	1,32 Media 0 %
1	Spain	165	146	339	54,07 %	2,05
2	China	154	151	79	33,05 %	0,51
3	France	125	101	223	50,57 %	1,78
4	Germany	75	71	94	43,32 %	1,25
5	United States	55	53	23	39,66 %	0,42
6	Italy	18	18	16	42,11 %	0,89
7	United Kingdom	16	10	18	56,25 %	1,13
8	India	11	11	9	64,29 %	0,82
9	Switzerland	10	10	7	43,75 %	0,70
10	Russia	9	8	3	21,43 %	0,33

Additional references

Publications

- PandaX-III: Searching for neutrinoless double beta decay with high pressure 136Xe gas time projection chambers. X. Chen et al., Science China Physics, Mechanics & Astronomy 60, 061011 (2017), arXiv:1610.08883.
- Background assessment for the TREX Dark Matter experiment. Castel, J., Cebrián, S., Coarasa, I. et al. Eur. Phys. J. C 79, 782 (2019). arXiv:1812.04519.
- Topological background discrimination in the PandaX-III neutrinoless double beta decay experiment. J Galan et al 2020 J. Phys. G: Nucl. Part. Phys. 47 045108, arxiv:1903.03979.
- AlphaCAMM, a Micromegas-based camera for high-sensitivity screening of alpha surface contamination, Konrad Altenmüller et al 2022 JINST 17 P08035

Conference talks

- REST v2.0 : A data analysis and simulation framework for micro-patterned readout detectors., Javier Galan, 2016-Dec, 8th Symposium on Large TPCs for low-energy rare event detection, Paris.
- REST-for-Physics, Luis Obis, 2022-May, ROOT Users Workshop, FermiLab.

Few final words

- At present, REST-for-Physics dev-team counts with **4 active core developers**, plus many other **sporadic contributions** from project contributors to different libraries.
- Up to now the project has been mainly **fed by ERC IAXO Grant**.
- We have dedicated many efforts, but we do not get yet to complete everything we would wish to have inside REST-for-Physics.
- REST-for-Physics was conceived for gaseous TPCs, however, by design, it can be easily extrapolated to other technologies.
- **Many common mathematical routines** can be already exploited by the community, such as: detector readout topology, waveform signal conditioning, etc.
- REST-for-Physics is mainly maintained by members **in an academic environment**.
- It would **be interesting to find synergies** inside the RD51/DRD1 communities, find potential users/developers willing to explore REST for RD51 applications.
- The project scales and it generates us certain maintenance overhead.
  - Validation pipelines
  - Release production
  - Bug correction
  - Communication and feedback

REST-for-Physics Framework

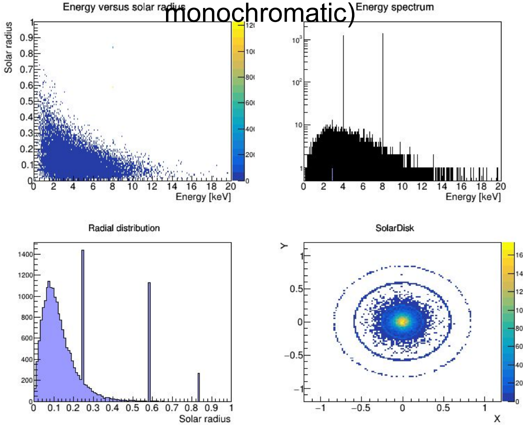
Backup slides

20.06.2023 - Javier Galan - javier.galan@unizar.es

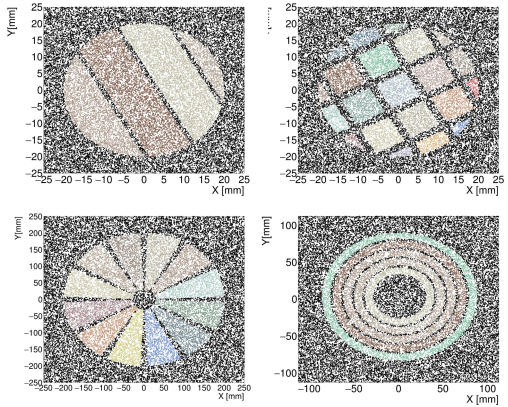
Centro de Astropartículas y Física de Altas Energías
Universidad Zaragoza

MonteCarlo ray-tracing

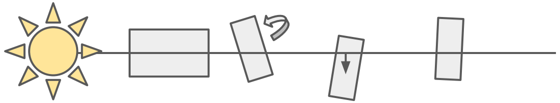
Arbitrary solar flux components (continuum + monochromatic)



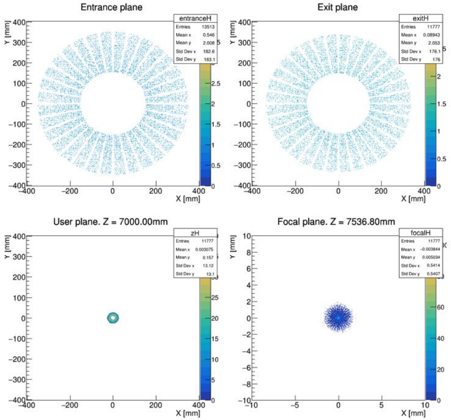
Masks used for optics and windows



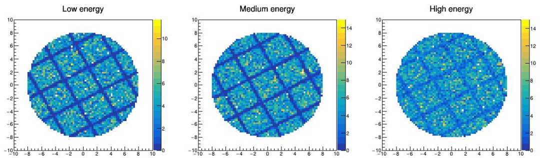
Arbitrary physical position and rotation for any component that inherits from TRestAxionEventProcess



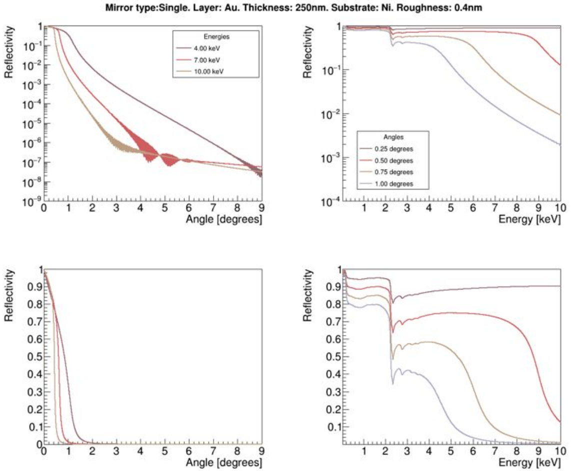
Optics



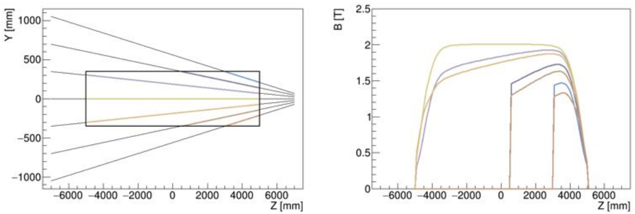
Window transmission



Mirror optical properties



Inhomogeneous magnetic field profile integration



Any REST library will implement specific objects that inherit from these 3 basic prototyping classes. Prototype classes define common data members and methods.

TRestMetadata: Any class inheriting from TRestMetadata will allow us to initialize the class data members from a configuration file, RML.

TRestEvent: It defines event data holders. Structures where we store event data that needs to be processed. Any class inheriting from TRestEvent will define and event id, a timestamp, and other common fields that define an event.

TRestEventProcess: It defines methods that allow for input/output event data processing. On top of that, this class inherits from TRestMetadata, so that the required process parameters can be retrieved from a configuration file.

Most of the classes present inside REST-for-Physics inherit from any of those 3 prototype classes.

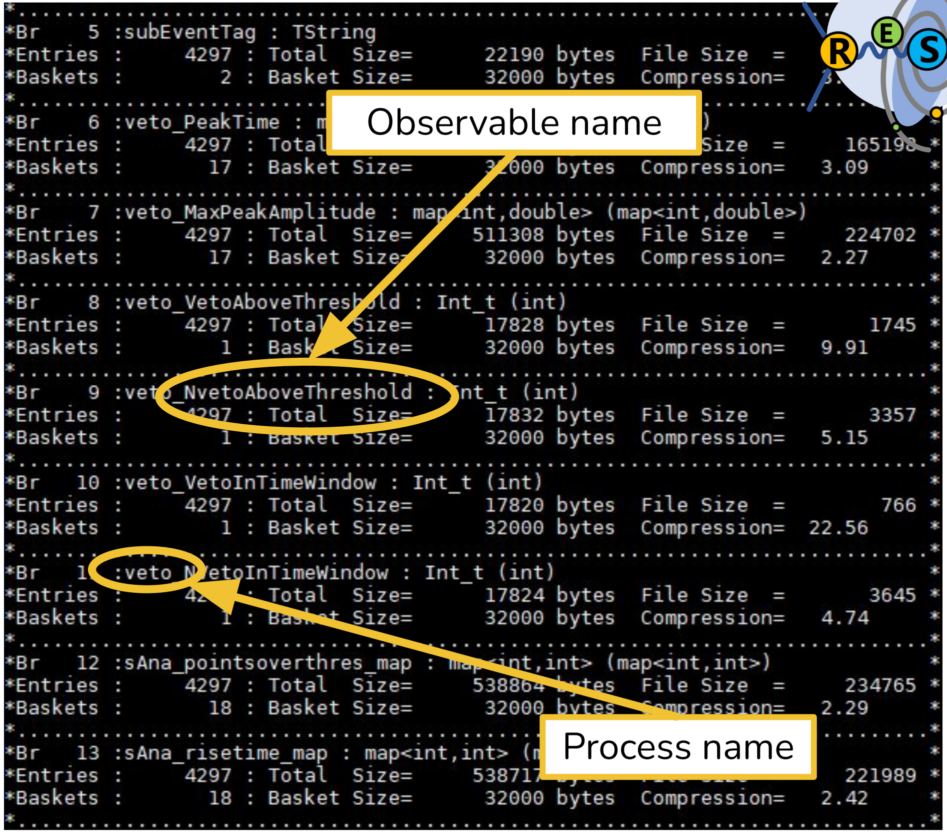
TRestMetadata	TRestEventProcess	TRestEvent
TRestRun	TRestEventSelectionProcess	TRestRawSignalEvent
TRestDetectorReadout	TRestRawSignalShapingProcess	TRestDetectorHitsEvent
TRestDetectorGas	TRestDetectorElectronDiffussionProcess	TRestGeant4Event
TRestGeant4Metadata	TRestTrackPathMinimizationProcess	...
...	TRestGeant4NeutronTaggingProcess	
	...	

REST basic classes: TRestAnalysisTree

The analysis tree is one of the most relevant products of an event data processing chain in REST.

Accumulative, once an observable is added it will always be present in future event data processing.

Each process can generate new observables inside this tree during the event data chain.



RML as an upgraded XML (I)

The RML uses XML format, but it introduces some necessary upgrades.

- System and local variables than can be invoked at any time using the **${variableName}** format.
- Common parameters defines inside the <globals> section that will be propagated to any metadata class defined in the same RML.

```
<globals>
    <variable name="SHAPING" value="OFF" />

    <parameter name="sampling" value="10ns" />
    <parameter name="electricField" value="100" units="V/cm" />
    <parameter name="gasPressure" value="1" />
</globals>
```

```
<addProcess type="TRestRawSignalShapingProcess" name="shaping" title="Signal shaping" value="${SHAPING}">
    <parameter name="shapingType" value="shaperSin" />
    <parameter name="shapingTime" value="10" />
    <parameter name="gain" value="1" />
</addProcess>
```

If the process does not define that parameter, then it will be just ignored.

RML as an upgraded XML (II)

The RML uses XML format, but it introduces some necessary upgrades.

- Mathematical formula interpretation
- Programming features, FOR loops and IF conditions.

```
<if condition="${RUN_TYPE}==RawData">
    <TRestDetector name="detParam" >
        <parameter name="detectorName"
    </TRestDetector>
</if>
```

```
// Last strip is special
<readoutChannel id="nChannels-1"  type="y">
  <for variable="nPix" from="0" to="nChannels-1" step="1" >
    <addPixel id="${nPix}" origin="(nChannels*pitch,pitch/4+${nPix}*pitch)" si
    <addPixel id="nChannels+${nPix}" origin="(nChannels*pitch,pitch/4+${nPix}*p
  </for>
  <addPixel id="2*nChannels" origin="(nChannels*pitch-pitch/2,pitch/4+(nChanne
</readoutChannel>
```

RML as an upgraded XML (III)

The RML uses XML format, but it introduces some necessary upgrades.

- Implements physical units inside parameter definitions.

```
<parameter name="electricField" value="1" units="kV/cm" />
```

```
<parameter name="electricField" value="1kV/cm" />
```

- Allows including sections that have been defined in separate files.

```
<globals file="globals.xml"/>

<TRestRun file="run.xml"/>

<TRestProcessRunner name="RawSignals"
```

Additional concepts on RML definitions

Inside an RML we may also identify different common keywords

- **constant:** It defines an internal local variable inside a RML section that can be invoked without using ${}.
- **parameter:** As we have seen, it identifies with a std:: data member at the corresponding class.
- **observable:** We will see this tomorrow, it will allow the user to configure which observables should be added to the analysis tree by a particular process.

```
<constant name="pitch" value="${PITCH}" overwrite="false" />
<constant name="nChannels" value="${N_CHANNELS}" overwrite="false" />
<constant name="pixelSize" value="${PITCH}" />

<readoutModule name="pixelModule" size="(nChannels*pixelSize, nChannels*pixelSize)" tolerance="1.e-4" >
    // We use for loops to generate any number of channels given by the CHANNELS variable.
    // The loop variable must be placed between ${} in order to be evaluated.
    <for variable="nChX" from="0" to="nChannels-1" step="1" >
```

Physics units in REST-for-Physics

REST-for-Physics defines a system of the most common units.

All the values stored in REST (there might be exceptions) are stored in the <u>default units</u> value.

The elementary units inside REST can be combined, such that we can write “kV/cm” or “g/cm^3”.

When reading a new parameter with given units, its value is transformed internally to match the units value of the <u>default unit</u>, i.e. if pressure is given in MPa, it will be converted internally to bars, which is the default pressure unit in REST.

Default unit = 1

```
// pressure field unit multiplier
AddUnit(bar, REST_Units::Pressure, 1.);
AddUnit(mbar, REST_Units::Pressure, 1.e3);
AddUnit(atm, REST_Units::Pressure, 1.013);
AddUnit(torr, REST_Units::Pressure, 760);
AddUnit(MPa, REST_Units::Pressure, 0.101325);
AddUnit(kPa, REST_Units::Pressure, 101.325);
AddUnit(Pa, REST_Units::Pressure, 101325);
AddUnit(mPa, REST_Units::Pressure, 10132500);
```

Utilities in REST-for-Physics

Apart from the main classes that define the framework behaviour, the main framework defines also common components and utilities.

TRestPhysics: It defines common geometrical and mathematical operations required in particle physics. It also defines physics constants. These methods are available inside the namespace REST_Physics.

TRestTools: It defines common tools such as filename operations, or basic ASCII/binary table access/reading/writing.Defined as static functions inside TRestTools class.

TRestStringHelper: It defines methods for common string operations, such as type and format conversion, timestamp formatting, and more. Defines inside the namespace REST_StringHelper.

Output levels

We may use predefined output formats, such as RESTMetadata, RESTInfo, RESTWarning, RESTError, RESTDebug, producing different output highlights.

```
root [0] RESTMetadata << "====" << RESTendl; RESTMetadata << " " << RESTendl; RESTMetadata << "This is the predefined output format for metadata classes" << RESTendl; RESTMetadata << " " << RESTendl; RESTMetadata << "====" << RESTendl;
==========================================================================================
||                                                                                      ||
||                    This is the predefined output format for metadata classes         ||
||                                                                                      ||
==========================================================================================
```

The different output formats help to identify critical information and to warn the user about any unexpected behaviour.

```
root [0] RESTInfo << "This is an info message" << RESTendl;
-- Info : This is an info message
root [1] RESTWarning << "This is a warning message" << RESTendl;
-- Warning : This is a warning message
root [2] RESTError << "This is an error message" << RESTendl;
-- Error : This is an error message
root [3] RESTDebug << "This is a debug message" << RESTendl;
-- Debug : This is a debug message
```

But the output formats are not only aesthetical, they also define a message priority or output levels!

Output levels (verbose level) exist such that messages are given certain priority.

Some examples of verbose level output

- If verboseLevel=0 (silent) no messages will be shown at all.
- If verboseLevel=1 (warning) only warning and error messages will be shown.
- If verboseLevel=2 (info) metadata and other info is shown on top of it.
- If verboseLevel=3 (debug) additional debugging output is printed out.

Any metadata class implements an independent verbose level that can be defined by the user at the RML level.

```
<TRestRun name="TREX-DM" title="TREX-DM test data analysis" verboseLevel="silent">
    <parameter name="experimentName" value="TREXDM_LSC"/>
    <parameter name="runNumber" value="preserve"/>
    <parameter name="runTag" value="preserve"/>
```

When using restRoot interactively we may define the desired output level.

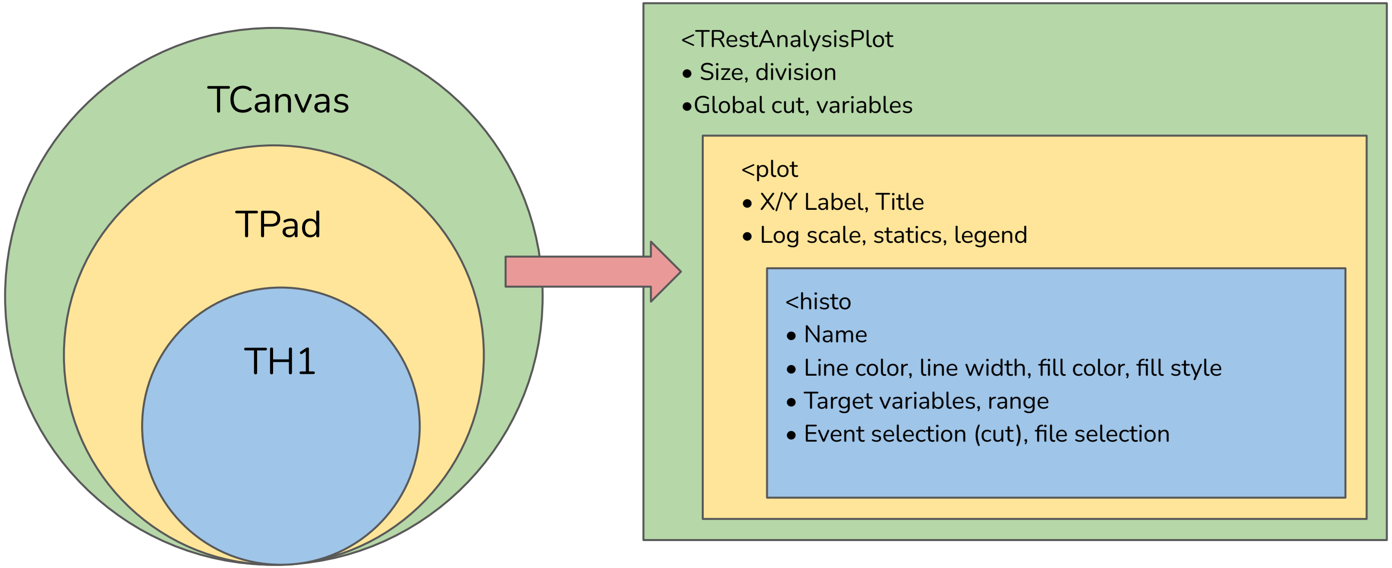
```
restRoot --v [VERBOSE_LEVEL]

Where VERBOSE_LEVEL=0,1,2,3 is equivalent to silent, warning, info, debug
```

TRestAnalysisPlot metadata tool

TRestAnalysisPlot is a metadata class (receives input from a configuration file) that allows to create plot definitions that can be invoked later on for different datasets.

It can be used for systematic plot generation, dataset comparison, and data quality control (or quickLook analysis).

Follows the same hierarchy as ROOT drawing scheme.



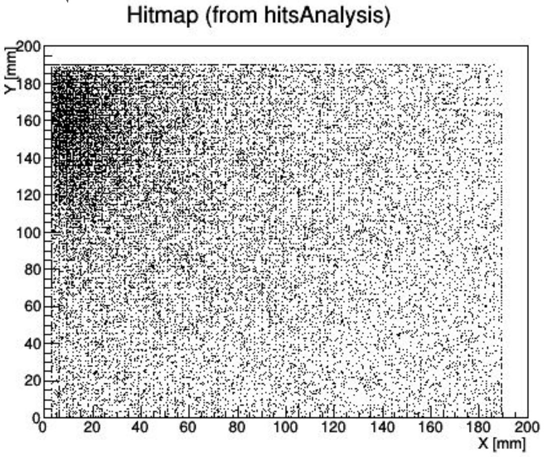
Systematic plot generation using TRestAnalysisPlot

```
    <plot name="Hitmap" title="Hitmap (from hitsAnalysis)" xlabel="X [mm]"
ylabel="Y [mm]"
          logscale="false" save="/tmp/file3.png"  value="ON" >
          <variable name="hitsAna_yMean" range="(0,200)" nbins="1000" />
          <variable name="hitsAna_xMean" range="(0,200)" nbins="1000" />
    </plot>
```

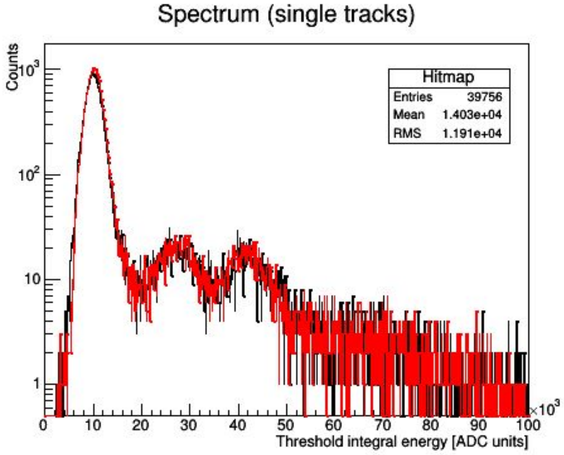
We can do 1D, 2D or 3D plots

```
    <plot name="Hitmap" title="Spectrum (single tracks)" xlabel="Threshold integral energy [ADC
units]" ylabel="Counts"
          logscale="true" save="/tmp/file4.pdf"  value="ON" >
          <variable name="sgnlAna.ThresholdIntegral" range="(0,100000)" nbins="1000" />

          <cut variable="tckAna_nTracksX" condition="==1" value="ON" >
          <cut variable="tckAna_nTracksY" condition="==1" value="ON" >
    </plot>
```

We can apply specific cuts to each plot definition



We can also define weights, i.e. using the value of another variable to weight each histogram entry.



TRestAnalysisPlot::PlotCombinedCanvas()

It will create a canvas with all the plots we defined inside our RML.

<canvas size="(1000,800)" divide="(2,2)" />

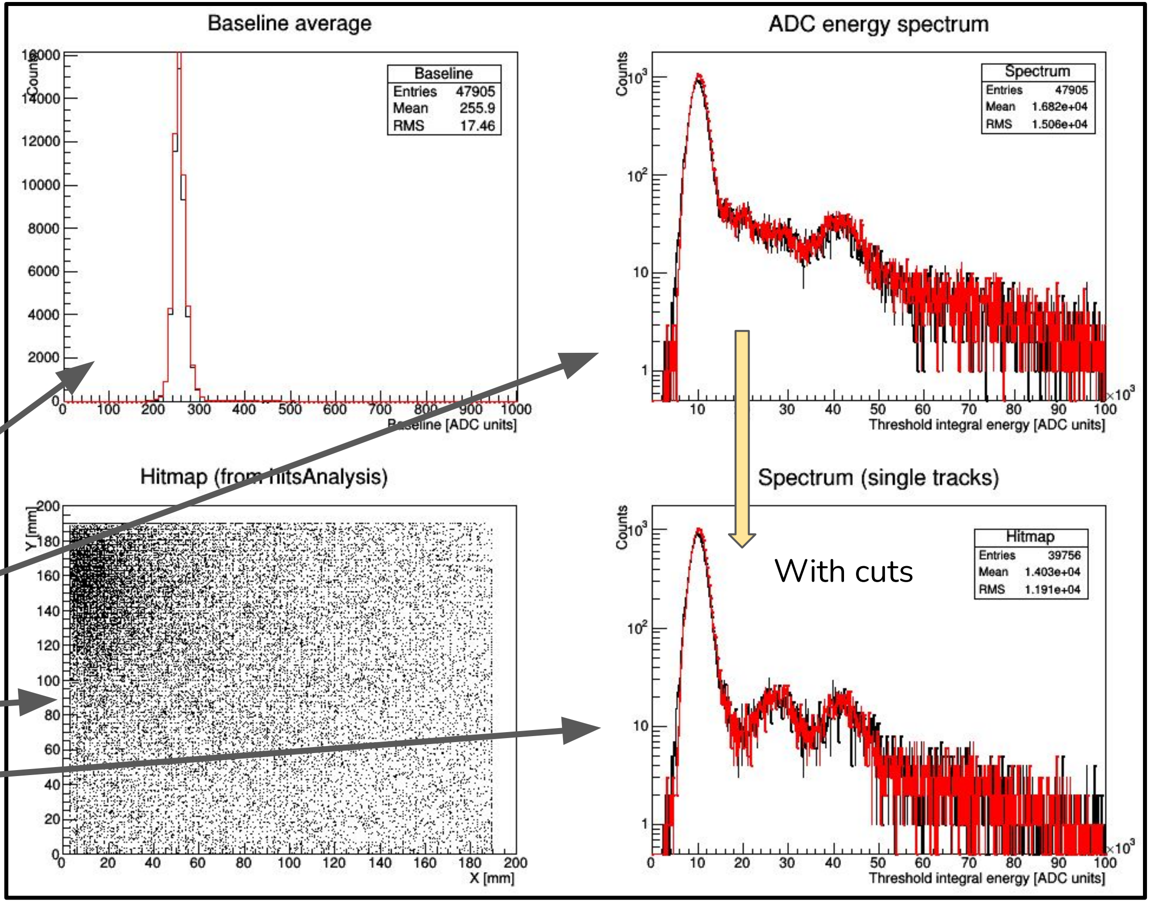
We may use the save option to write to disk the histograms generated in different formats (pdf/png images, ROOT file, or C-macro).

<plot name=”Baseline” ...>

<plot name=”Spectrum” ...>

<plot name=”Hitmap” ...>

<plot name=”Spectrum2” ...>



Systematic plot generation using TRestMetadataPlot

Full example at framework/examples/metadataPlot.rml

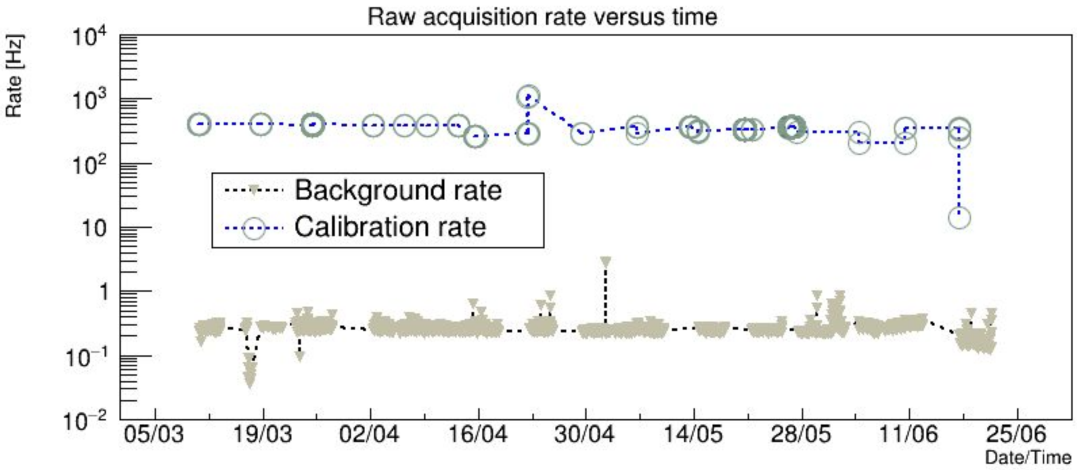
```
<TRestMetadataPlot>
          <plot name="rate" title="Raw acquisition rate versus time" xVariable="timestamp" ... >

          <graph name="meanRateBck" title="Background rate"   option="PL">
              <parameter name="yVariable" value="TRestSummaryProcess->fMeanRate" />
              <parameter name="metadataRule" value="TRestRun->fRunTag==Background_BIPO" />
          </graph>
      …
</TRestMetadataPlot>
```

Create a graph with any TRestMetadata member found at the ROOT file.

```
TRestXXX::fDataMember
```

Create a condition (metadataRule) to filter the files that should be considered.

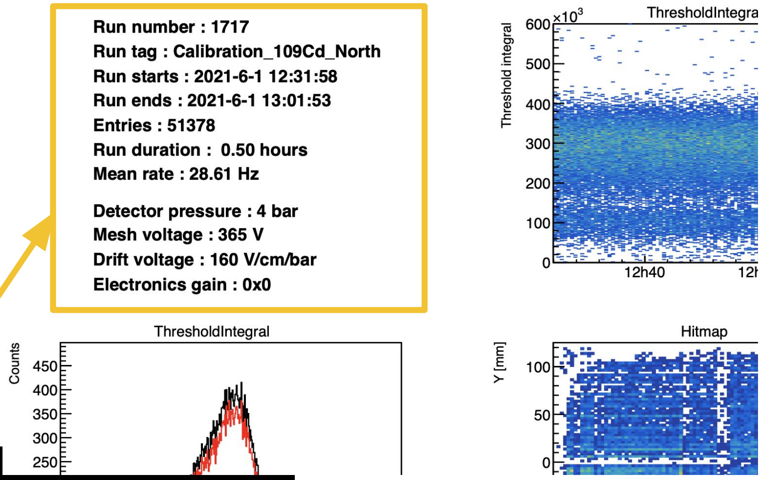
```
TRestRun->fRunTag==Background_BIPO
```



Systematic plot generation. Panel.

We can also generate a panel with information found inside the metadata objects written to disk together with the data.

In this example we extract information from TRestRun and TRestDetector.




```
<panel font_size="0.06">
    <label value="Run number : [TRestRun::fRunNumber]" x="0.25" y="0.9" />
    <label value="Run tag : [TRestRun::fRunTag]" x="0.25" y="0.82" />
    <label value="Run starts : <<startTime>>" x="0.25" y="0.74" />
    <label value="Run ends : <<endTime>>" x="0.25" y="0.66" />
    <label value="Entries : <<entries>>" x="0.25" y="0.58" />
    <label value="Run duration : <<runLength>> hours" x="0.25" y="0.50" />
    <label value="Mean rate : <<meanRate>> Hz" x="0.25" y="0.42" />

    <label value="Detector pressure : [TRestDetector::fPressure] bar" x="0.25" y="0.30" />
    <label value="Mesh voltage : [TRestDetector::fAmplificationVoltage] V" x="0.25" y="0.22" />
    <label value="Drift voltage : [TRestDetector::fDriftField] V/cm/bar" x="0.25" y="0.14" />
    <label value="Electronics gain : [TRestDetector::fElectronicsGain]" x="0.25" y="0.06" />
</panel>
```

Members between << >> are special members defined inside TRestAnalysisPlot and TRestMetadataPlot.

A TRestDataSet definition allows to use metadata conditions to make a selection of files and select the relevant observables we are interested in.

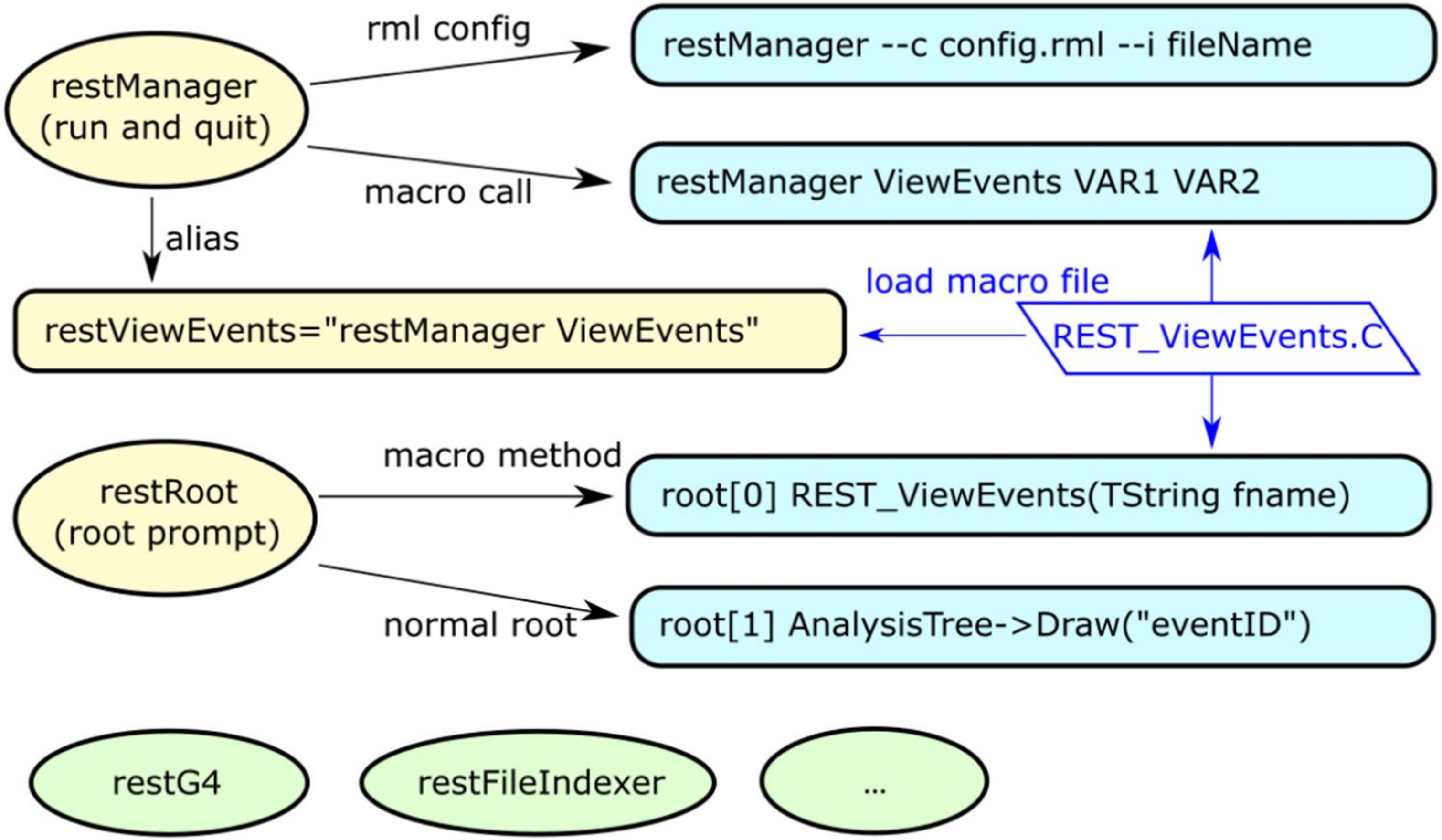
File range to be selected using glob pattern, date range, and any number of metadata filters

```
<TRestDataSet name="DummySet">
  <parameter name="startTime" value = "2022/04/28 00:00" />
  <parameter name="endTime" value = "2022/04/28 23:59" />
  <parameter name="filePattern" value="test*.root"/>
  <filter metadata="TRestRun::fRunTag" contains="Baby" />
  // Will add to the final tree only the specific observables
  <observables list="g4Ana_totalEdep:hitsAna_energy" />
  // Will add all the observables from the process `rawAna`
  <processObservables list="rate:rawAna" />
  <quantity name="Nsim" metadata="[TRestProcessRunner::fEventsToProcess]"
        strategy="accumulate" description="The total number of simulated events."/>
</TRestDataSet>
```

When we export the dataset, apart from the analysis tree observables we may add other relevant quantities that will be included inside the dataset export (e.g. at the TXT header).

Inside our dataset we then really select the few observables that we want to export to our dataset. See more details at the class documentation.

Different ways of invoking/using REST-for-Physics



Some examples of using REST commands

(1) You can also call REST packages without Python bindings (using !)

```
!restG4 --help
```

```
restG4 requires at least one parameter, the rml configuration file (-c is optional)

example: restG4 example.rml

there are other convenient optional parameters that override the ones in the rml file:
        -h or --help | show usage (this text)
        -c example.rml | specify RML file (same as calling restG4 example.rml)
        -g geometry.gdml | specify geometry file
        -i | set interactive mode (default=false)
        -s | set serial mode (no multithreading) (default=true)
        -t nThreads | set the number of threads, also enables multithreading
```

(2) Let's run a simulation with restG4!

```
!restG4 simulations/simulation.rml
```

(3) You can see config file contents via console or

```
!cat simulations/simulation.rml
```

(4) To see ROOT file contents:

```
filename = "restG4_CosmicMuons_run00001.root"

file = ROOT.TFile(filename)

file.ls()
```

```
TFile**          restG4_CosmicMuons_run00001.root
 TFile*          restG4_CosmicMuons_run00001.root
  KEY: TRestAnalysisTree        AnalysisTree;3  AnalysisTree
  KEY: TTree     EventTree;3    TRestGeant4EventTree
  KEY: TRestRun DemoRun;3       A Demo Run
  KEY: TRestGeant4Metadata      restG4 run;2    Cosmic Muons
  KEY: TRestGeant4PhysicsLists  default;2       Physics List implementation.
```

(5) To access simulation event information:

```
run = ROOT.TRestRun(filename)

run.Print()

print(f"This run has {run.GetEntries()} entries")

event = ROOT.TRestGeant4Event()

run.SetInputEvent(event)

run.GetEntry(0)

event.PrintEvent()
```