REST-for-Physics Framework

2.1 Introduction to REST-for-Physics

25.01.2023 - Javier Galan - javier.galan@unizar.es

Centro de Astropartículas y Física de Altas Energías
Universidad Zaragoza

Motivation

- Define common input/output data formats that allow us to preserve data in the long term.

- Use common algorithms for data processing.
    - Consolidate routines that lead to science results.
    - Receive feedback on existing routines: User feature request, use cases, bug report.
    - Encourage developers discussion on code evolution.

- Develop synergies in software development and maintenance. Encouraging collaborative design, testing, code implementation, bug reporting, know-how transfer, documentation.

Outline of the session

1. What is REST-for-Physics. Repositories, main site, and publications.
2. Prototype classes: TRestMetadata, TRestEvent and TRestEventProcess
3. Basic classes: TRestAnalysisTree.
4. RML Configuration features and physics units.
5. Utility classes: TRestPhysics, TRestTools and TRestStringHelper.
6. Output levels.
7. Merging datafiles: TRestAnalysisPlot, TRestMetadataPlot, TRestDataSet

What is REST-for-Physics

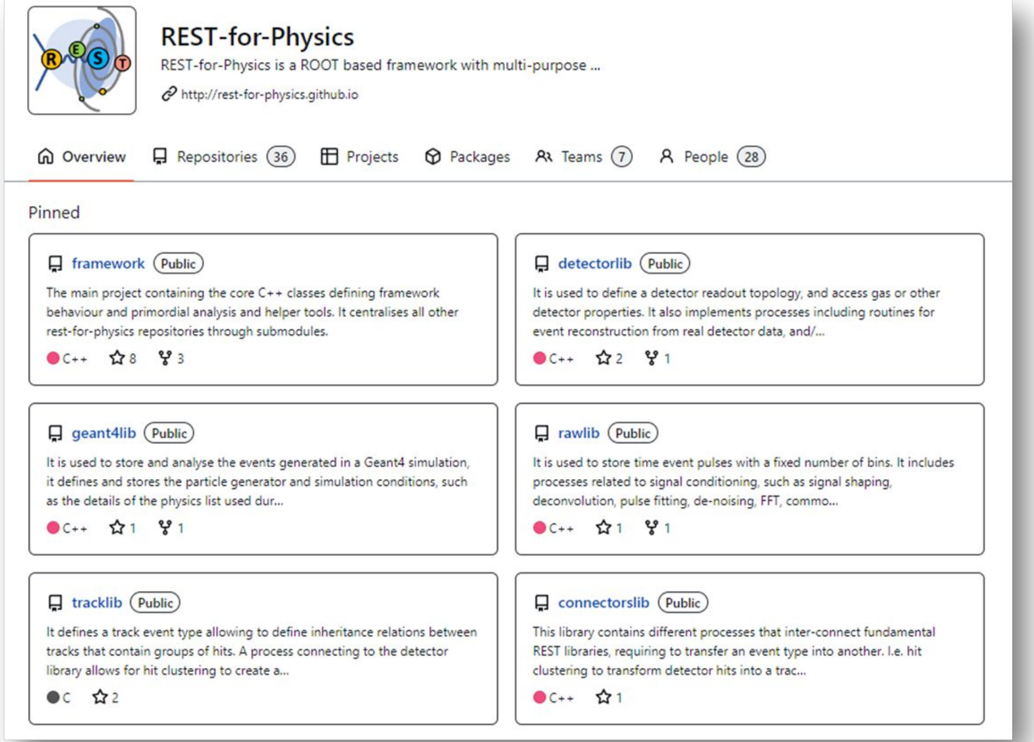
- The REST-for-Physics (Rare Event Searches Toolkit) Framework is a collaborative software effort that provides common tools for:
    - acquisition,
    - simulation,
    - data analysis
- It was originally designed to work with data of gaseous Time Projection Chambers (TPCs).
- It is mainly written in C++ and it is fully integrated with ROOT I/O interface.
- The REST framework establishes a common procedure and output data format to define input information, via configuration (.rml) files.

Centralizing site → https://rest-for-physics.github.io/

REST-for-Physics repositories ecosystem

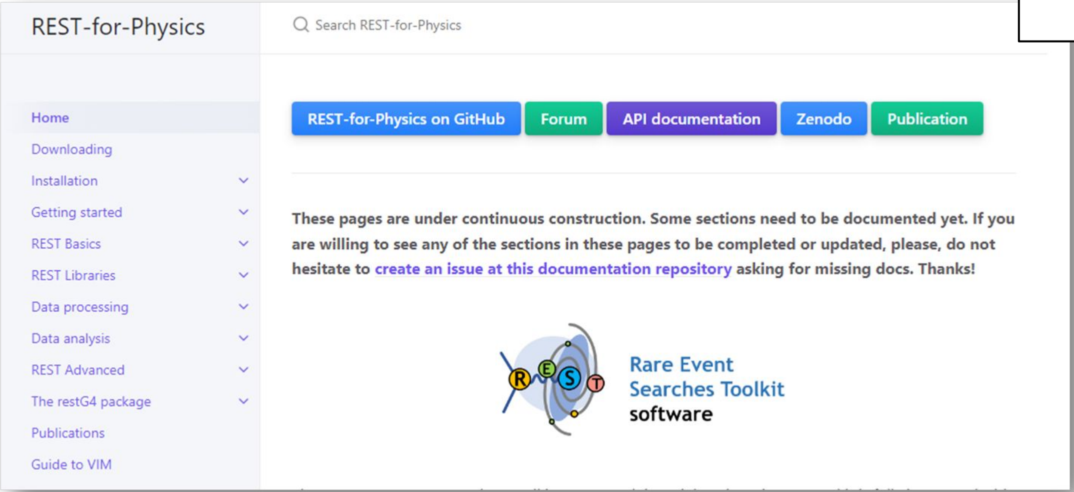
The full REST-for-Physics project is splitted in different Github repositories

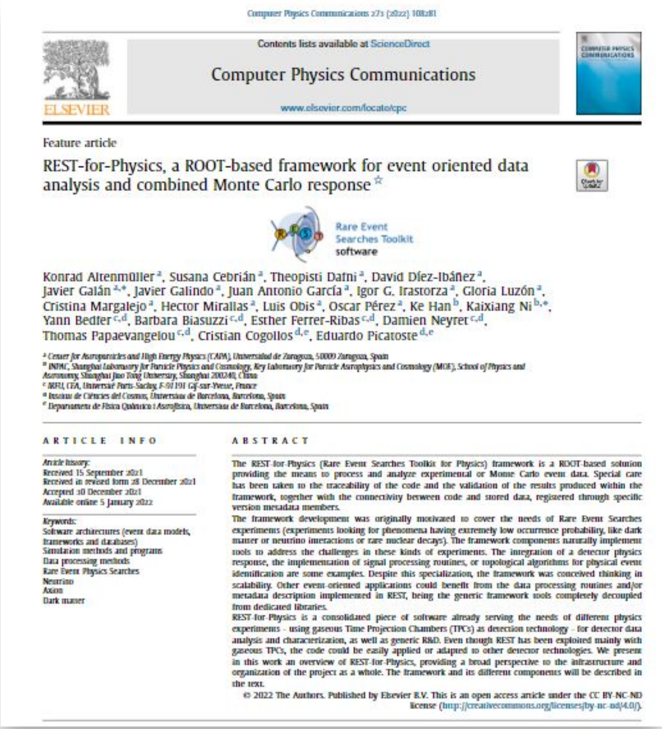
- **Main project**
  - Framework
- **Libraries** for montecarlo and detector data processing
  - Rawlib / Geant4lib
  - Detectorlib / Tracklib
  - Axionlib
  - Connectorslib
- **Packages** that exploit REST libraries
  - restG4
  - restSQL
  - …



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

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.

REST prototype classes

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.

REST prototype classes

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	TRestDetectorElectronDiffusionProcess	TRestGeant4Event
TRestGeant4Metadata	TRestTrackPathMinimizationProcess	...
...	TRestGeant4NeutronTaggingProcess	
	...	

REST prototype classes : TRestMetadata

A metadata class is any holder of data other than event data that is relevant to understand the origin and history of transformations that a given set of event data has gone through.

The TRestMetadata class identifies C++ data members with XML parameters.

C++ header

```
/// An abstract class to define common optics par
class TRestAxionOptics : public TRestMetadata {
   protected:
    /// An optics file that contains all the spec
    std::string fOpticsFile = "";

    /// The mirror length. If all mirrors got the
    Double_t fMirrorLength = 0;   //<
```

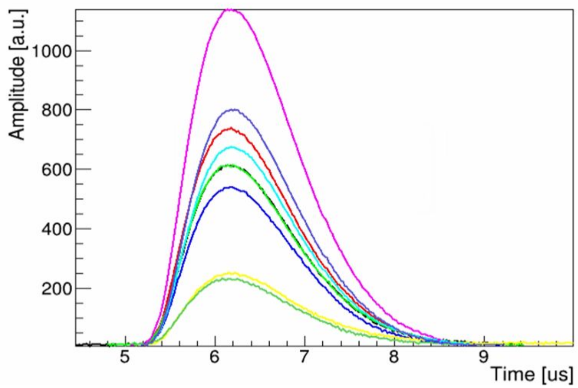
RML config file

```
<TRestAxionWolterOptics name="xmm" verboseLevel="warnin
    <parameter name="opticsFile" value="XMM.Wolter" />

    <parameter name="mirrorLength" value="300" />
```

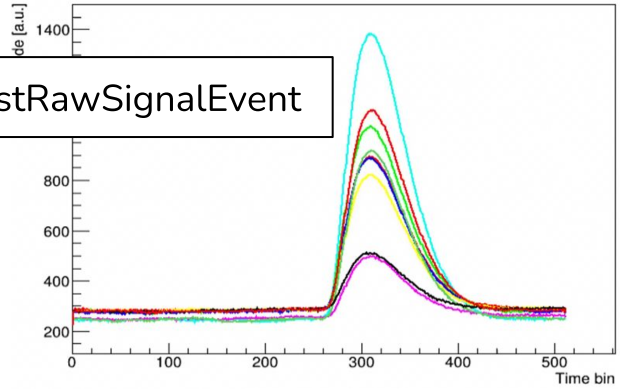
Question: Why the RML section is not the same as the C++ class?

As we will in the course, each library defines its own event. Examples of event data types are those that describe physical information on time or on a 3-dimensional space.

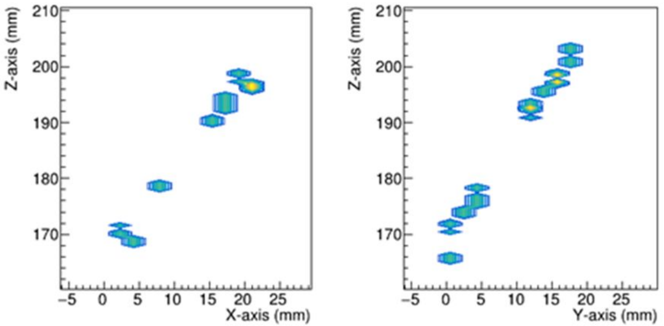
TRestDetectorSignalEvent



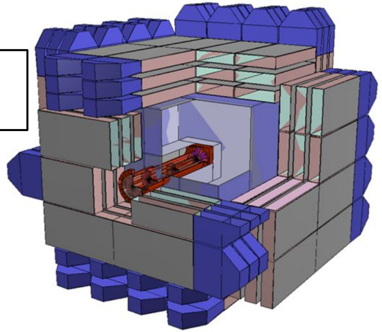
TRestRawSignalEvent

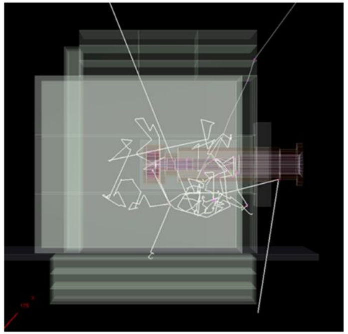


TRestDetectorHitsEvent



TRestGeant4Event

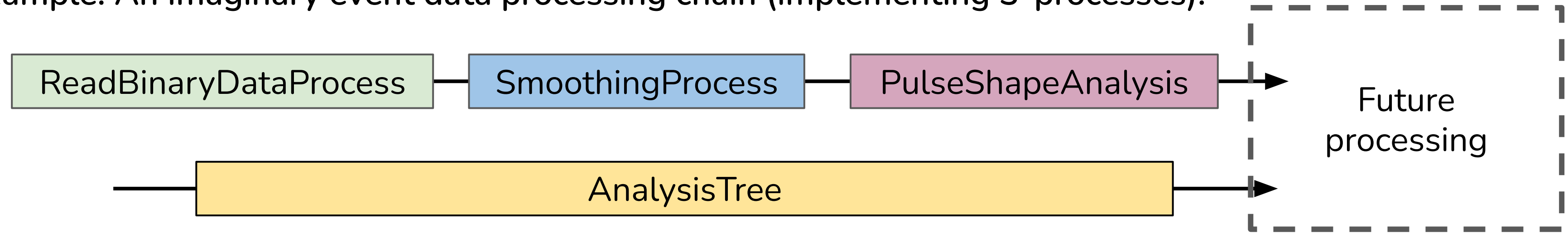




REST basic classes: TRestEventProcess

- A process is an operation that acts on an input event of a specific type and creates an output event that can be the same type than the input event or a different one.
- Any TRestEventProcess inheriting class can be connected in a sequential event processing chain.
- A process may transform the event data or extract valuable information in the form of an observable that will be added to the analysis tree.

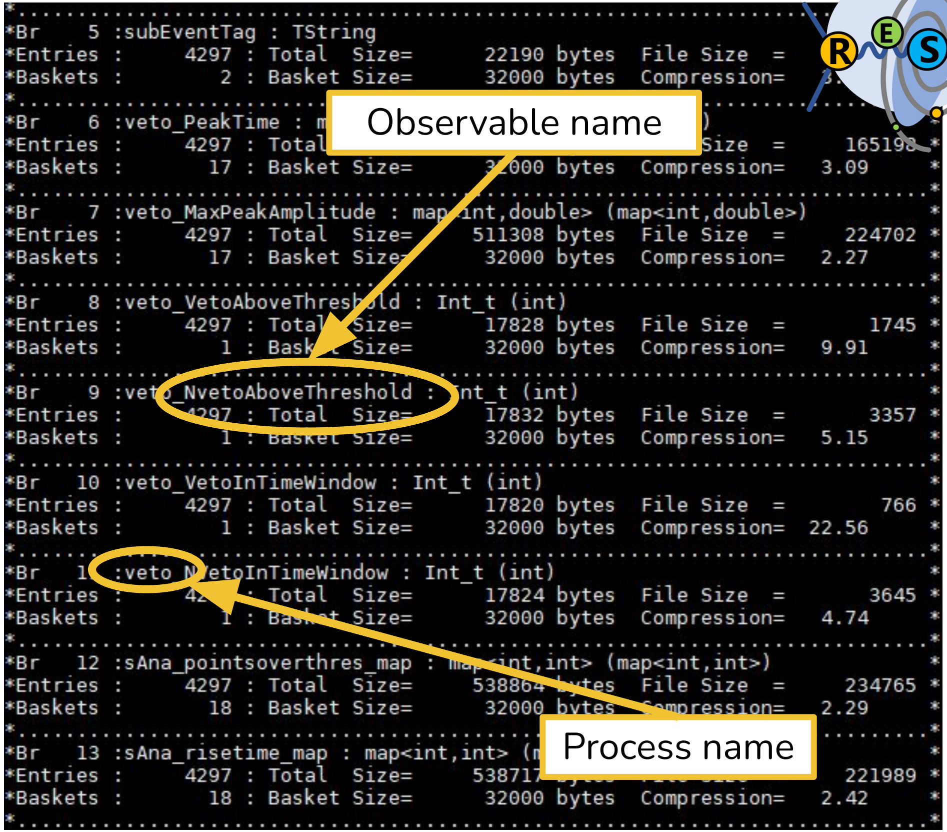
Example: An imaginary event data processing chain (implementing 3-processes):



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.

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" >
```

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 default units 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 default unit, 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);
```

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 metadat
a 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 <u>output levels</u>!

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
```

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

→

Existing tools

- TRestAnalysisPlot
- TRestMetadataPlot
- TRestDataSet

→

Used for

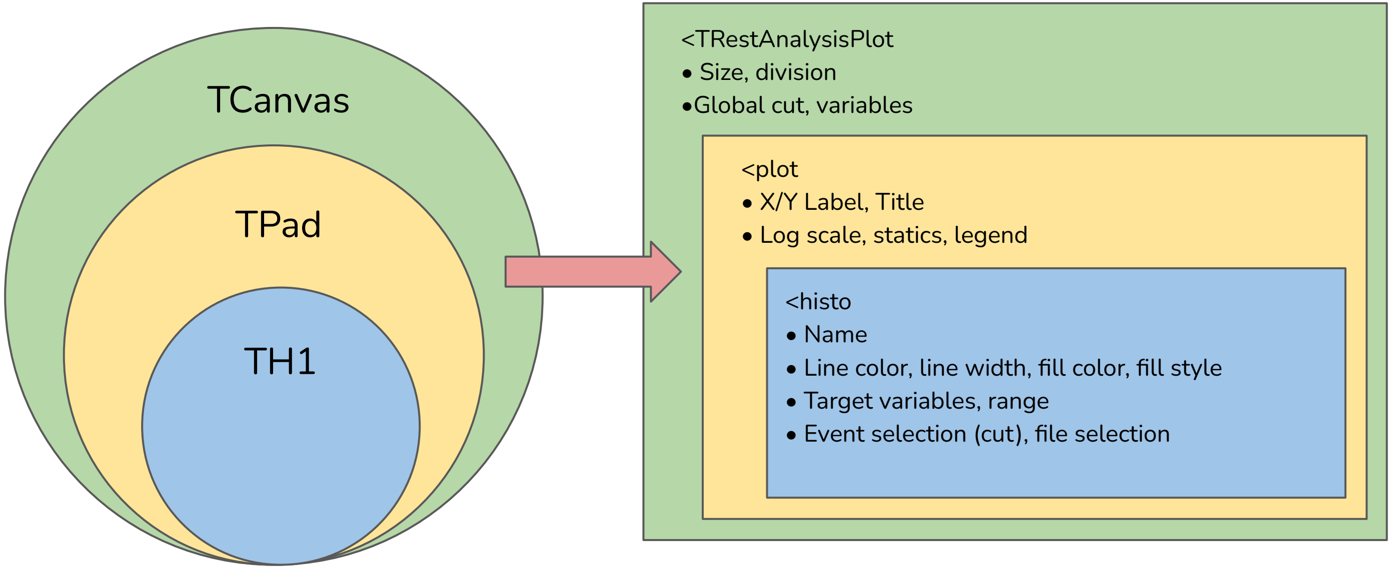
- Systematic plot generation
- Metadata correlation between different runs
- Metadata time evolution
- Dataset generation
- Export datasets (txt/SQL/tree/...)

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.

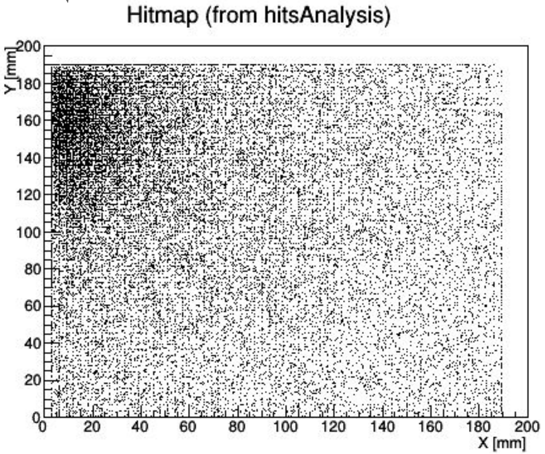


```
    <plot name="Hitmap" title="Hitmap (from hitsAnalysis)" xlabel="X [mm]"
ylabel="Y [mm]"
          logscale="false" save="/tmp/file3.png"  value="ON" >
          <variable name="hitsAna_yMean" range="(0,200)" nbins="1000" />
          <variable name="hitsAna_xMean" range="(0,200)" nbins="1000"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

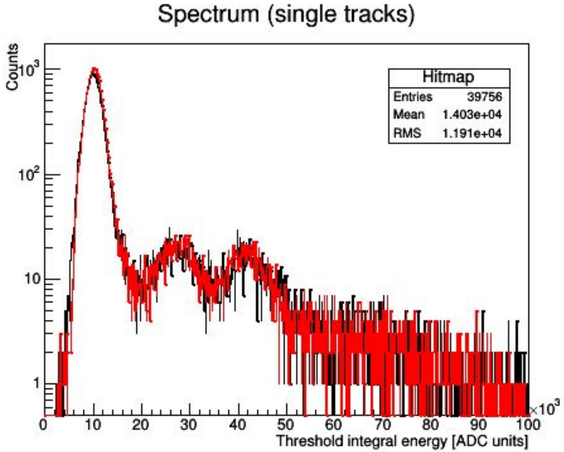
We can do 1D, 2D or 3D plots

```
    <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 <u>weights</u>, 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)" />

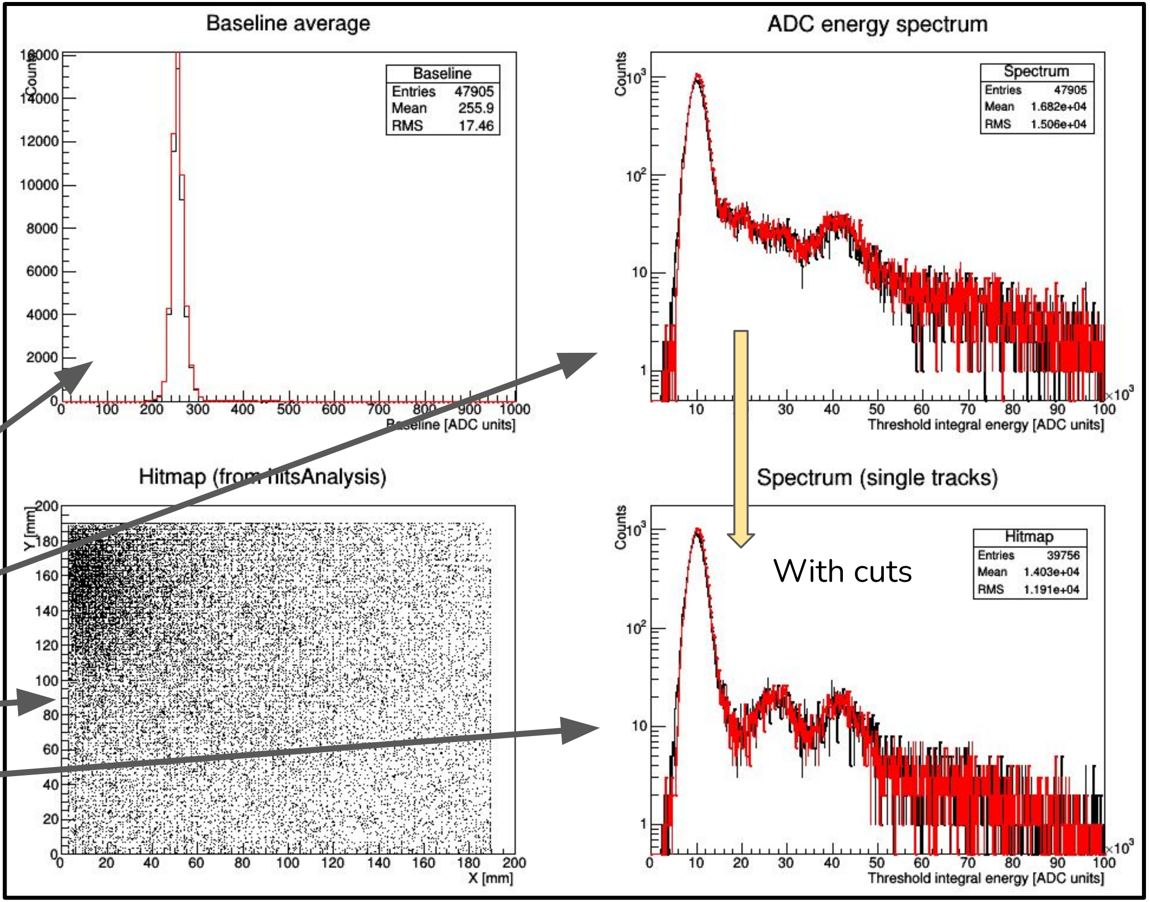
We may use the save option to write to disk the histograms generated in different formats (pdf/png images, ROOT file, or C-macro).

<plot name=”Baseline” ...>

<plot name=”Spectrum” ...>

<plot name=”Hitmap” ...>

<plot name=”Spectrum2” ...>



Systematic plot generation using TRestMetadataPlot

Full example at framework/examples/metadataPlot.rml

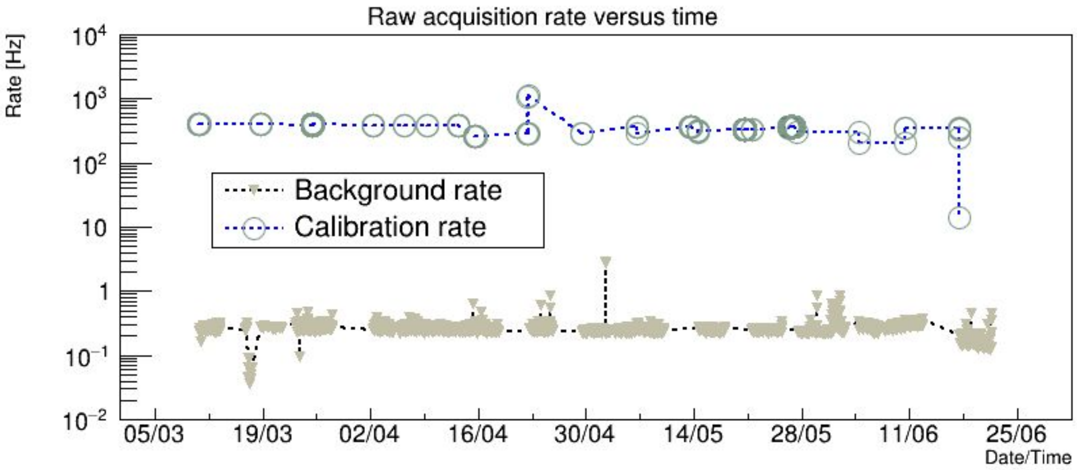
```
<TRestMetadataPlot>
          <plot name="rate" title="Raw acquisition rate versus time" xVariable="timestamp" ... >

          <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.

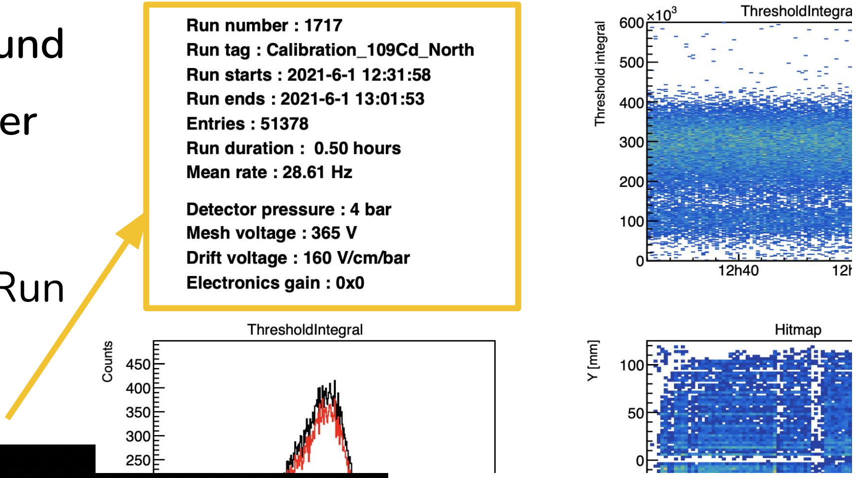
```
TRestRun->fRunTag==Background_BIPO
```



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.

Generating datasets

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.

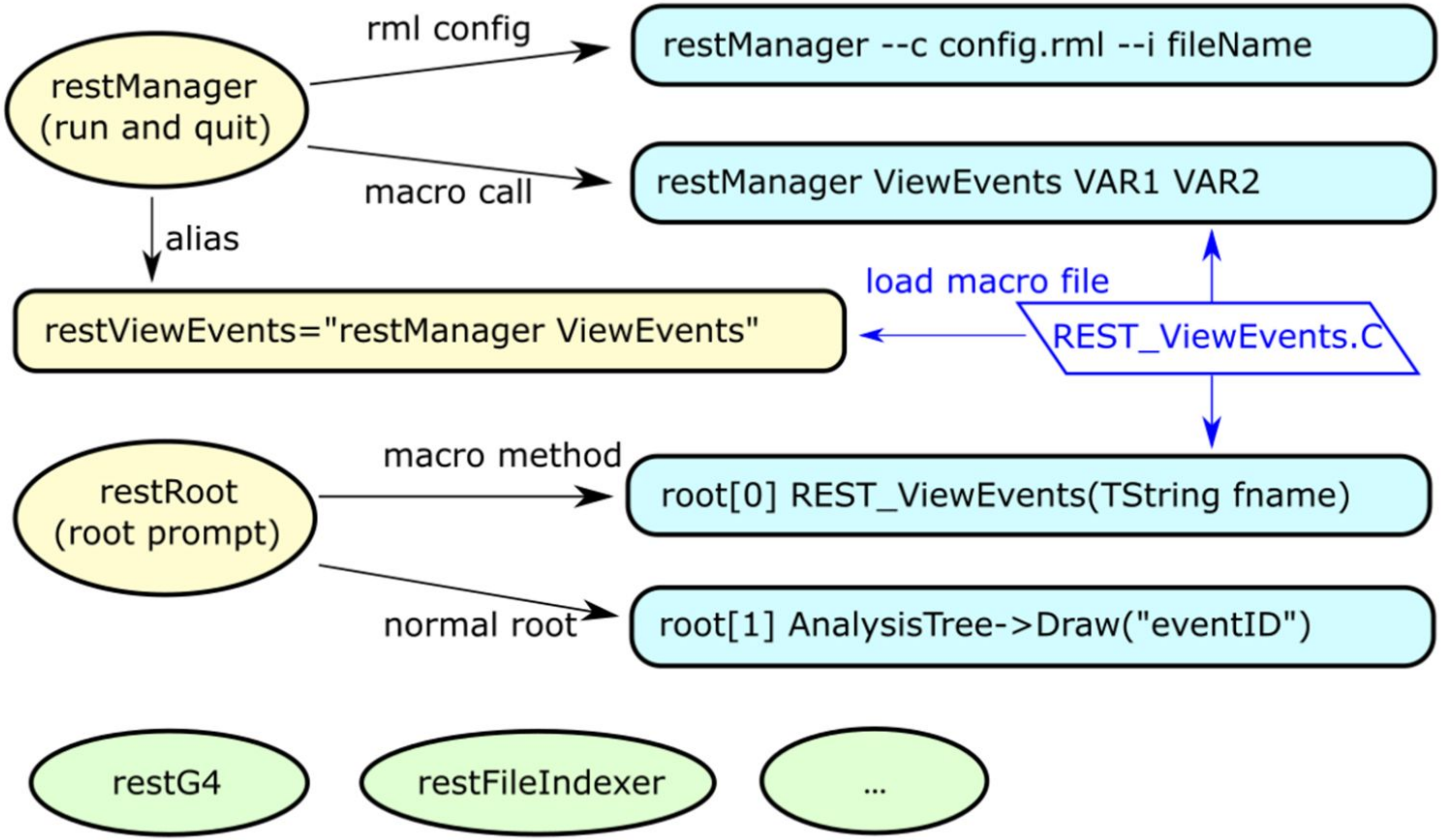
REST-for-Physics Framework

Time for exercises!

25.01.2023 - Javier Galan - javier.galan@unizar.es

Centro de Astropartículas y Física de Altas Energías
Universidad Zaragoza

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()
```