SySal.ROOT

The data analysis interface between SySal and ROOT

Salerno Emulsion Group Technical Note

May 2004

For updated versions, refer to http://sysal.sa.infn.it

Overview

SySal.ROOT is a thin software layer to interface the two worlds of SySal and ROOT. Since many of the people that are contributing to the development of SySal like to do their analysis tasks under ROOT, this package enables them not only to read directly into SySal native files without need for a file converter, but also to write native SySal files. In other words, it is possible to build off-line data processing and analysis packages, developed entirely under ROOT, and to use the results of these tasks within the SySal framework without the need for file converters. Those who are familiar with the ROOT world don’t need to know anything about SySal classes. They simply keep developing their algorithms in the familiar ROOT world, but their results are then immediately available to the SySal community.
SySal.ROOT was developed under Linux, and was then ported to Windows. Total compatibility between the two worlds has been extensively proven.

1 General structure

SySal.ROOT features three main classes: TSySalRWD (for raw data files), TSySalTLG (scanning zones with base tracks) and TSySalTSR (files with Total Scan volume reconstructions). Each one of these classes exposes the data contained respectively in RWD, TLG and TSR data files. A SySal data file is seen by ROOT as a collection of familiar TTree objects. No additional class library is needed to access SySal data.
The user of SySal.ROOT can work in two ways: one option is to read / write SySal files and keep them in the PC memory; the second option is to use a ROOT file as a temporary or persistent representation of the SySal file. In the former case, it is evident that there is a lower risk of cluttering the hard-disk with useless files; in the second case, less memory is used, and, it is possible to obtain a ROOT file that no longer depends on the SySal.ROOT library.
SySal.ROOT is compiled as SySalROOT.so under Linux, and SySalROOT.dll under Windows. It can be used both in C++ programs (compiled by GNU GCC/G++ under Linux and MSVC++ 6.0 / 7.0 under Windows) or from the ROOT command line and ROOT scripts.

2 Quick start examples

In order to make the reader familiar with SySal.ROOT, here are a few code samples that can be run from the ROOT command line (ROOT v.3.10 running on Linux RedHat 7.2/7.3). SySal data files are supposed to be in the `/tmp/data` directory. The SySalROOT.so library is supposed to be in the `/tmp/dev` directory. Indeed, it must be pointed out that SySal.ROOT does not need any specific arrangement of the directory structure, nor any specific environment variable setting.

Example #1: reading a TLG file from the ROOT command line

```
1) gSystem->Load(“/tmp/dev/SySalROOT.so”);
     // this loads the library
2) TSySalTLG *tlg = new TSySalTLG(0);
     // a TLG object container is created
3) tlg->LoadTLG(“/tmp/data/myfile.tlg”);
     // the TLG file is loaded and appears now as TTrees
```

Example #2: reading a set of RWD files from the ROOT command line

```
1) gSystem->Load(“/tmp/dev/SySalROOT.so”);
     // this loads the library
2) TSySalRWD *rwd = new TSySalRWD(0);
     // a RWD object container is created
3) rwd->LoadRWD(“/tmp/data/myset.rwd.00000001”);
4) rwd->LoadRWD(“/tmp/data/myset.rwd.00000002”);
5) rwd->LoadRWD(“/tmp/data/myset.rwd.00000003”);
6) rwd->LoadRWD(“/tmp/data/myset.rwd.00000004”);
     // four RWD files are merged and appear as TTrees
```

Example #3: converting a TSR file into a library-independent ROOT file from the ROOT command line

```
1) gSystem->Load(“/tmp/dev/SySalROOT.so”);
     // this loads the library
2)  TFile  *tf  =  new  TFile(“/tmp/data/myvolume.root”,
“RECREATE”);
3) TSySalTSR *tsr = new TSySalTSR(tf);
     /*   a TSR object container is created that uses tf
          as temporary storage */
4) tsr->LoadTSR(“/tmp/data/myvolume.tsr”);
     // the TSR file is loaded and appears now as TTrees
5) tf->Write();
     // the ROOT file is written to disk
6) tf->Close();
     // the ROOT file is closed
```

Example #4: converting a ROOT file into a TSR file from a ROOT macro

```
tsr->SaveTSR(“/tmp/data/myprocessedvolume.tsr”, 2);
```

```
     /*   the tsr object (of type TSySalTSR), that
          contains a list of TSR volumes, extracts the
          third (index 2) total scan reconstruction and
          writes it to /tmp/data/myprocessedvolume.tsr */
```

or

```
tsr->SaveTSR(“/tmp/data/myprocessedvolume.tsr”);
     /*   the tsr object (of type TSySalTSR), that
          contains a list of TSR volumes, extracts the
          first (index 0, the default) total scan
          reconstruction and writes it to
          /tmp/data/myprocessedvolume.tsr */
```

Example #5: obtaining a scatter-plot of base track slopes

```
tlg->GetBaseTracks()->Draw(“SlopeY:SlopeX”);
     /*   this obtains the TTree with base tracks from
          the TLG container and invokes the Draw method
          of TTree */
```

or

```
1) TFile *tf = new TFile(“/tmp/data/convertedtlg.root”,
“READ”);
     /*   a ROOT file with TLG trees is loaded from disk;
          notice that the SySalROOT.so library is no
          longer needed! */
2) tf->Get(“TLGBaseTracks)->Draw(“SlopeY:SlopeX”);
     /*   the TTree with base tracks is accessed from
          within the TFile, and the Draw method is
          invoked */
```

3 Library files

SySal.ROOT comes in a single library. The library file is called SySalROOT.so under Linux and SySalROOT.dll under Windows.
ROOT users just need to load the library with the usual command:

```
      gSystem->Load(“/mypath/SySalROOT.so”)
```

C++ developers need to include the library header, called SySalROOT.h.
Linux developers can compile their C++ program adding the `SySalROOT.so` entry to the gcc / g++ command line.
Windows developers can compile their C++ program adding the SySalROOT.lib file to the MSVC++ 6.0 / 7.0 project file.
Should recompilation be needed, the SySalROOT.cpp file contains the full source code. In order to compile it, the path to ROOT headers (e.g. `$ROOTSYS/include`) should be added to the list of the default include directories.
SySalROOT must be linked against the libCint, libTree and libRFIO ROOT libraries.
The appropriate dictionary for your version of ROOT can be generated by running the following command from the OS shell:

```
rootcint –f Dict.cxx –c SySalROOT.h LinkDef.h
```

This generates the Dict.cxx and Dict.h files that are provided by the ROOT interpreter. In order to get the SySalROOT.so, go to the directory where SySalROOT.h, SySalROOT.cpp and LinkDef.h are, and type (in a single line):

```
g++ SySalROOT.cpp Dict.cxx –I$ROOTSYS/include
-L$ROOTSYS/lib –lCint –lTree –lRFIO –ldl –shared –fPIC –o
SySalROOT.so
```

For MSVC++ 6.0 / 7.0, open SySalROOT.dsw and click on the “Build” menu button. The SySalROOT.dll file is built.

4 Library classes

SySal.ROOT defines three classes for data management and bare structures for data storage and representation. In the current version, these structures are purely C-like structures with no methods. In the future, if needed, methods to perform common operations might be added.
The data management classes are called TSySalRWD, TSySalTLG and TSySalTSR. Each instance of a data management class creates its own ROOT TTrees. Each branch of the TTrees has the structure of the corresponding C++ `struct`, and the association is very obvious. For example, the "Data" branch of the "TLGBaseTracks" TTree can be read or filled through a `TLGBaseTrack` variable, as shown by the following code fragment that takes data from a TLG converted into ROOT file:

```
TTree *tbasetracks = mytfile->Get("TLGBaseTrack");
TLGBaseTrack temp;
tbasetracks->GetBranch("Data")->SetAddress(&temp);
n = tbasetracks->GetEntries();
for (i = 0; i < n; i++)
{
     tbasetracks->GetEntry(i);
     printf("\nSlopeX: %f SlopeY: %f", temp.SlopeX,
          temp.SlopeY);
}
```

If data come directly from an unconverted TLG, stored in a TSySalTLG, we use:

```
TTree *tbasetracks = tlg->GetBaseTracks();
TLGBaseTrack temp;
tbasetracks->GetBranch("Data")->SetAddress(&temp);
n = tbasetracks->GetEntries();
for (i = 0; i < n; i++)
{
     tbasetracks->GetEntry(i);
     printf("\nSlopeX: %f SlopeY: %f", temp.SlopeX,
          temp.SlopeY);
}
```

In the following subsections, data structures and classes are documented.

4.1 RWD file interface

The data container class for RWD files is TSySalRWD.

TSySalRWD public methods:

`TSySalRWD(TFile *tf)`	Creates a new TSySalRWD object, using the supplied TFile for storage (PC memory is used if `tf` is 0).
`bool LoadRWD(char *path)`	Loads an RWD file into the TSySalRWD's object TTrees. Returns `true` if the operation is successful, `false` otherwise.
`bool SaveRWD(char *path, int id = 0)`	Extracts the id^{th} RWD file from the TSySalRWD container and puts it into the specified file path. Returns `true` if the operation is successful, `false` otherwise.
`TTree *GetHeader()`	Returns the RWDHeader TTree.
`TTree *GetViews()`	Returns the RWDView TTree.
`TTree *GetTopLayers()`	Returns the RWDTopLayers TTree.
`TTree *GetBottomLayers()`	Returns the RWDBottomLayers TTree.
`TTree *GetTopTracks()`	Returns the RWDTopTracks TTree.
`TTree *GetBottomTracks()`	Returns the RWDBottomTracks TTree.
`TTree *GetTopGrains()`	Returns the RWDTopGrains TTree.
`TTree *GetBottomGrains()`	Returns the RWDBottomGrains TTree.
`TTree *GetFile()`	Returns the TFile object associated to this TSySalRWD container (or 0 if PC memory is being used).

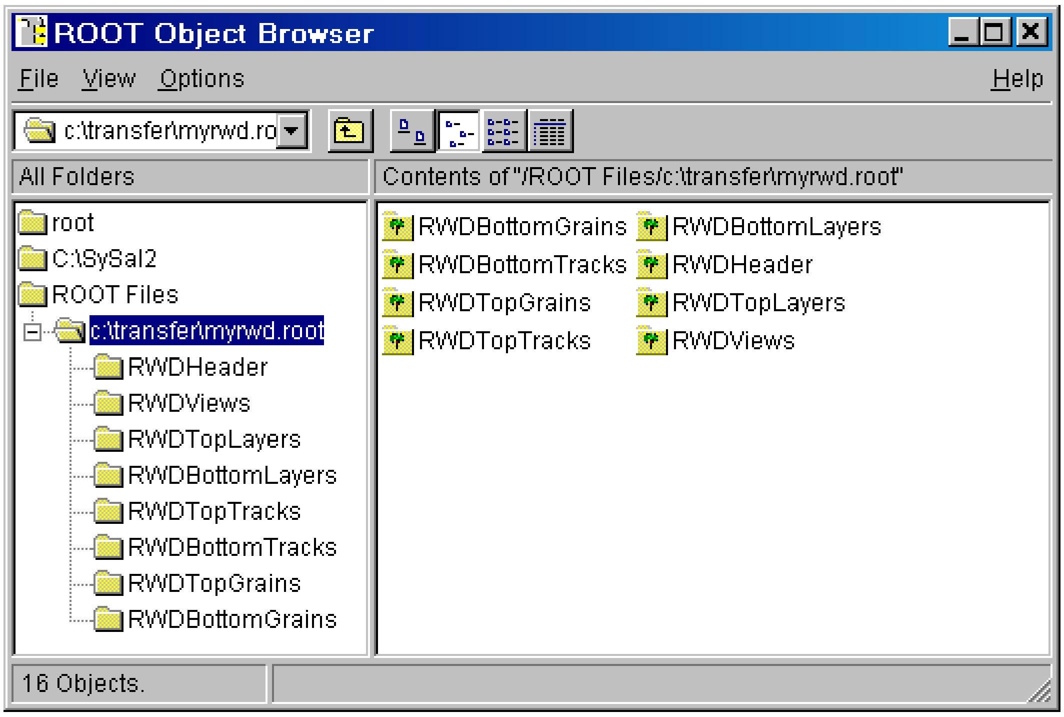
The structures used with RWD files are listed below.

Structure name	**Tree/Branch**	**Content**
`RWDHeader`	RWDHeader/Data	Global information about an RWD file.
`RWDView`	RWDViews/Data	Global information about one view.
`RWDViewMap`	RWDViews/Map	Map information to translate local camera coordinates to global coordinates
`RWDLayer`	RWDTopLayers/Data, RWDBottomLayers/Data	Aggregate information about one layer in one side of one view.
`RWDEmulsionTrack`	RWDTopTracks/Data, RWDBottomTracks/Data	Microtrack information.
`RWDGrain`	RWDTopGrains/Data, RWDBottomGrains/Data	One grain in a microtrack

It is worth to notice that these structures are defined inside the TSySalRWD, so the full name of a RWDEmulsionTrack is, for example:

```
TSySalRWD::RWDEmulsionTrack
```

This policy minimizes the probability of name clashing with other libraries.
The trees that can be found in a ROOT file obtained through conversion can be read from the snapshot below.



The documentation of RWD data structures follows.

RWDHeader
```
struct RWDHeader
{
     Int_t Id0;
     Int_t Id1;
     Int_t Id2;
     Int_t Id3;
     Int_t Index;
     Int_t StartView;
     Int_t Views;
     Int_t CodingMode;
     Int_t TopLayers;
     Int_t BottomLayers;
     Int_t TopTracks;
     Int_t BottomTracks;
     Int_t TopGrains;
     Int_t BottomGrains;
};
```

Id0, Id1, Id2, Id3 are four numeric identifiers for the RWD set.
Index is the number of the fragment in this RWD set.
StartView is the number of the first view with respect to the full RWD set.
Views is the number of views in this RWD.
CodingMode is 0 if microtrack grains are uncompressed and were stored in the file, nonzero otherwise.
TopLayers, BottomLayers tell the total number of layers in this fragment.
TopTracks, BottomTracks tell the total number of microtracks in this fragment.
TopGrains, BottomGrains tell the total number of microtrack grains in the fragment.

RWDView
```
struct RWDView
{
     Int_t Id;
     Int_t TileX;
     Int_t TileY;
     Int_t TopTracks;
     Int_t BottomTracks;
     Int_t TopFlags;
     Int_t BottomFlags;
     Int_t TopLayers;
     Int_t BottomLayers;
};
```

Id are four numeric identifiers for the RWD set.
TileX is the column position in the grid of the views that build up the scanned area.
TileY is the row position in the grid of the views that build up the scanned area.
TopTracks, BottomTracks tell the number of microtracks in this view.
TopFlags, BottomFlags are flags that signal various scanning problems.
TopLayers, BottomLayers tell the number of layers per side in this view.

RWDViewMap

```
struct RWDViewMap
{
     Double_t TopExt;
     Double_t TopInt;
     Double_t BottomInt;
     Double_t BottomExt;
     Double_t TopPosX;
     Double_t TopPosY;
     Double_t TopMapPosX;
     Double_t TopMapPosY;
     Double_t TopMapMXX;
     Double_t TopMapMXY;
     Double_t TopMapMYX;
     Double_t TopMapMYY;
     Double_t BottomPosX;
     Double_t BottomPosY;
     Double_t BottomMapPosX;
     Double_t BottomMapPosY;
     Double_t BottomMapMXX;
     Double_t BottomMapMXY;
     Double_t BottomMapMYX;
     Double_t BottomMapMYY;
};
```

TopExt is the Z position of the plate-air surface.
TopInt is the Z position of the emulsion-base surface of the top side of the sheet.
BottomInt is the Z position of the emulsion-base surface of the bottom side of the sheet.
BottomExt is the Z position of the plate-glass surface of the bottom side of the sheet.
(Top/Bottom)Pos(X/Y) is the X/Y coordinate of the centre of the field of view in stage coordinates while scanning the Top/Bottom side.
(Top/Bottom)MapPos(X/Y) is the X/Y coordinate of the centre of the field of view in sheet coordinates while scanning the Top/Bottom side.
(Top/Bottom)MapM(X/Y)(X/Y) is the X/Y,X/Y coefficient of the M matrix that transforms camera view coordinates to sheet coordinates.

The transformation for the top side, X coordinate is:

```
xsheet = TopMapPosX + TopMapMXX * x + TopMapMXY * y
```

RWDLayer

```
struct RWDLayer
{
     Int_t Id;
     Int_t Grains;
     Double_t Z;
};
```

Id is the layer number.
Grains is the number of grains in the layer.
Z is the Z coordinate of the layer.

RWDEmulsionTrack

```
struct RWDEmulsionTrack
{
     Int_t AreaSum;
     Int_t Grains;
     Double_t InterceptX;
     Double_t InterceptY;
     Double_t InterceptZ;
     Double_t SlopeX;
     Double_t SlopeY;
     Double_t Sigma;
     Double_t TopZ;
     Double_t BottomZ;
     Int_t Id;
     Int_t View;
     Int_t Fragment;
     Int_t FirstGrain;
};
```

AreaSum is the sum of the area (in pixels) of all grains in the microtrack.
Grains is the number of grains in the microtrack.
InterceptX/Y/Z are the coordinate of one point laying on the fit line of the grains. This point is usually taken at the contact surface between emulsion and base, but the Z coordinate has been added just to eliminate the need to make assumptions. The coordinates are in μm, and the reference frame for the X/Y plane is centred at the centre of the field of view, whereas the Z is read from the stage.
SlopeX/Y are the X/Y components of the track slope.
Sigma is the transverse residual of the grains w.r.t. their fit line. "Transverse" means that the microtrack and the grains are ideally projected on the X/Y plane, and only the transverse displacements of grains are considered, thus obtaining a track quality parameter that does not depend on the absolute slope.
TopZ/BottomZ Z coordinate of the top/bottom grain.
Id sequential number of the microtrack in this side of this view.
View number of the view in this fragment.
Fragment number of the fragment in the RWD set.

FirstGrain is the global sequential number of the first grain of this microtrack in the RWD file. This number is reset only at the beginning of each fragment, so it helps find the grains associated to this track in the TopGrains/BottomGrains trees. For example, if we are in the first fragment of the RWD set, and this index is 54642, and Grains is 8, entries 54642 ... 54649 in the RWTopGrains TTree will contain the associated grains; if the fragment is the third one, and the RWDHeaders for the first two fragments had TopGrains = 1000000 and 1500000, the first grain of this track is found at entry 2554642.

Although SySal.ROOT allows to merge several files together for analysis purposes, they still can be re-extracted as individual units, so results of ROOT-based processing tasks can be re-injected in the SySal analysis and reconstruction chain.

RWDGrain

```
struct RWDGrain
{
	Double_t X;
	Double_t Y;
	Double_t Z;
	Int_t Area;
	Int_t Id;
};
```

Id is the number of the grain in the fragment (reset only on the first view of each RWD file).

Area is the number of pixels in the grain image.

X/Y/Z 3D coordinates of the grain in µm; the X/Y reference frame is centred at the centre of the field of view, whereas the Z coordinate is read from the stage coordinate.

4.2 TLG file interface

The data container class for TLG files is TSySalTLG.

TSySalTLG public methods:

`TSySalTLG(TFile *tf)`	Creates a new TSySalTLG object, using the supplied TFile for storage (PC memory is used if `tf` is 0).
`bool LoadTLG(char *path)`	Loads a TLG file into the TSySalTLG's object TTrees. Returns `true` if the operation is successful, `false` otherwise.
`bool SaveTLG(char *path, int id = 0)`	Extracts the idth TLG file from the TSySalTLG container and puts it into the specified file path. Returns `true` if the operation is successful, `false` otherwise.
`TTree *GetHeader()`	Returns the TLGHeader TTree.
`TTree *GetTopTracks()`	Returns the TLGTopTracks TTree.
`TTree *GetBottomTracks()`	Returns the TLGBottomTracks TTree.
`TTree *GetBaseTracks()`	Returns the TLGBaseTracks TTree.
`TTree *GetFile()`	Returns the TFile object associated to this TSySalTLG container (or 0 if PC memory is being used).

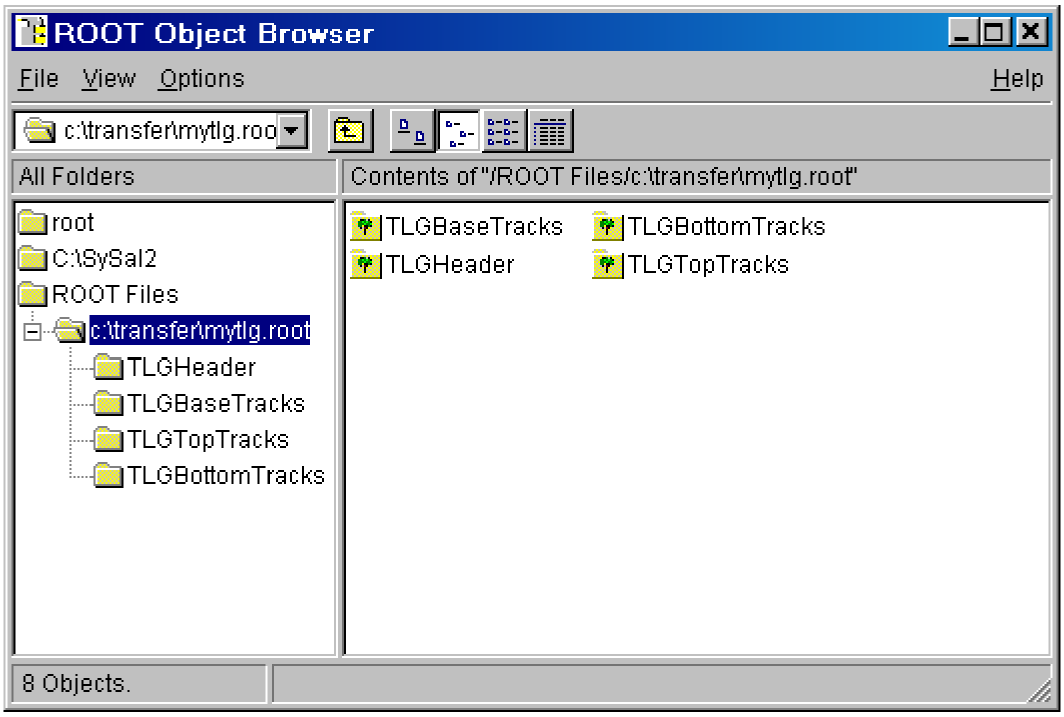
The structures used with TLG files are listed below.

Structure name	Tree/Branch	Content
`TLGHeader`	TLGHeader/Data	Global information about a TLG file.
`TLGEmulsionTrack`	TLGTopTracks/Data, TLGBottomTracks/Data	Microtrack information.
`TLGEmulsionTrackRawData`	TLGTopTracks/Index, TLGBottomTracks/Index	Indices that help location of the original microtrack in the raw data files.
`TLGBaseTrack`	TLGBaseTracks/Data	Base track information.

It is worth to notice that these structures are defined inside the TSySalTLG, so the full name of a TLGEmulsionTrack is, for example:

```
TSySalTLG::TLGEmulsionTrack
```

This policy minimizes the probability of name clashing with other libraries.
The trees that can be found in a ROOT file obtained through conversion can be read from the snapshot below.



The documentation of TLG data structures follows.

TLGHeader
```
struct TLGHeader
{
     Int_t Id0;
     Int_t Id1;
     Int_t Id2;
     Int_t Id3;
     Double_t CenterX;
     Double_t CenterY;
     Double_t MinX;
     Double_t MaxX;
     Double_t MinY;
     Double_t MaxY;
     Int_t TopTracks;
     Int_t BottomTracks;
     Int_t BaseTracks;
     Int_t Fields;
     Double_t TopExt;
     Double_t TopInt;
     Double_t BottomInt;
     Double_t BottomExt;
};
```

Id0, Id1, Id2, Id3 are four numeric identifiers for the TLG.
CenterX/Y are the coordinates of the centre of the zone.
(Min/Max)(X/Y) are the extents of the zone. Some track might slightly go out of these bounds.
(Top/Bottom)Tracks is the number of tracks in each side.
BaseTracks is the number of base tracks.
Fields is the number of fields in the TLG. Currently it is always 1.
TopExt is the Z position of the plate-air surface.
TopInt is the Z position of the emulsion-base surface of the top side of the sheet.
BottomInt is the Z position of the emulsion-base surface of the bottom side of the sheet.
BottomExt is the Z position of the plate-glass surface of the bottom side of the sheet.

TLGEmulsionTrack

```
struct TLGEmulsionTrack
{
	Int_t Id;
	Int_t Field;
	Int_t AreaSum;
	Int_t Grains;
	Double_t InterceptX;
	Double_t InterceptY;
	Double_t InterceptZ;
	Double_t SlopeX;
	Double_t SlopeY;
	Double_t Sigma;
	Double_t TopZ;
	Double_t BottomZ;
};
```

Id is the index of this track in its side.
Field is the field of view in which the track has been seen. Currently it is always 0.
AreaSum is the sum of the areas (in pixels) of all the grains in this track.
Grains is the number of grains in this track.
InterceptX/Y/Z are the coordinate of one point laying on the fit line of the grains. This point is usually taken at the contact surface between emulsion and base, but the Z coordinate has been added just to eliminate the need to make assumptions. The coordinates are in μm, and the coordinates for the X/Y plane are in the sheet reference frame.
SlopeX/Y are the X/Y components of the track slope.
Sigma is the transverse residual of the grains w.r.t. their fit line. “Transverse” means that the microtrack and the grains are ideally projected on the X/Y plane, and only the transverse displacements of grains are considered, thus obtaining a track quality parameter that does not depend on the absolute slope.
TopZ/BottomZ Z coordinate of the top/bottom grain.

TLGEmulsionTrackRawData

```
struct TLGEmulsionTrackRawData
{
	Int_t Fragment;
	Int_t View;
	Int_t Track;
};
```

Fragment is the fragment in which the original microtrack was seen.
View is the view index (within its fragment) of the view where the original microtrack was seen.
Track is the track index (within its view) of the original microtrack in the raw data.

TLGBaseTrack

```
struct TLGBaseTrack
{
	Int_t AreaSum;
	Int_t Grains;
	Double_t InterceptX;
	Double_t InterceptY;
	Double_t InterceptZ;
	Double_t SlopeX;
	Double_t SlopeY;
	Double_t Sigma;
	Int_t Id;
	Int_t IdT;
	Int_t IdB;
};
```

AreaSum is the sum of the areas (in pixels) of all the grains in this track (top grains + bottom grains).

Grains is the total number of grains in this track (top + bottom).

InterceptX/Y/Z are the coordinate of one point laying on the fit line of the grains. This point is usually taken at the contact surface between the top emulsion layer and the base, but the Z coordinate has been added just to eliminate the need to make assumptions. The coordinates are in μm, and the coordinates for the X/Y plane are in the sheet reference frame.

SlopeX/Y are the X/Y components of the track slope.

Sigma is a track quality parameter. Currently it is used to store the sum of the normalized angular disagreement between the top microtrack and the base and the normalized angular disagreement between the bottom microtrack and the base. The direction of the slope of the base track in the X/Y plane is called "longitudinal"; the direction orthogonal to this is called "transverse".

In formulae, we have:

$$\sigma = \sqrt{\left(\frac{\Delta S_{top\perp}}{Tol_{\perp}}\right)^2 + \left(\frac{\Delta S_{top=}}{Tol_{=}}\right)^2} + \sqrt{\left(\frac{\Delta S_{bottom\perp}}{Tol_{\perp}}\right)^2 + \left(\frac{\Delta S_{bottom=}}{Tol_{=}}\right)^2}$$

$$Tol_{=} = Tol_{\perp} + \alpha\sqrt{S^2_{x,base} + S^2_{y,base}}$$

α controls the increase of the longitudinal tolerance with the slope. If correctly tuned, σ is independent of the absolute slope of the base track.

TopZ/BottomZ Z coordinate of the top/bottom grain in the top/bottom side.

Id is the sequential index of the base track in the TLG.

IdT is the index of the top microtrack associated to this base track.

IdB is the index of the bottom microtrack associated to this base track.

4.3 TSR file interface

The data container class for TSR files is TSySalTSR.

TSySalTSR public methods:

`TSySalTSR(TFile *tf)`	Creates a new TSySalTSR object, using the supplied TFile for storage (PC memory is used if `tf` is 0).
`bool LoadTSR(char *path)`	Loads an TSR file into the TSySalTSR's object TTrees. Returns `true` if the operation is successful, `false` otherwise.
`bool SaveTSR(char *path, int id = 0)`	Extracts the idth TSR file from the TSySalTSR container and puts it into the specified file path. Returns `true` if the operation is successful, `false` otherwise.
`TTree *GetHeader()`	Returns the TSRHeader TTree.
`TTree *GetLayers()`	Returns the TSRLayer TTree.
`TTree *GetSegments()`	Returns the TSRSegments TTree.
`TTree *GetTracks()`	Returns the TSRTrack TTree.
`TTree *GetTrackSegments()`	Returns the TSRTrackSegments TTree.
`TTree *GetVertices()`	Returns the TSRVertices TTree.
`TTree *GetVertexTracks ()`	Returns the TSRVertexTracks TTree.
`TTree *GetFile()`	Returns the TFile object associated to this TSySalTSR container (or 0 if PC memory is being used).

The structures used with TSR files are listed below.

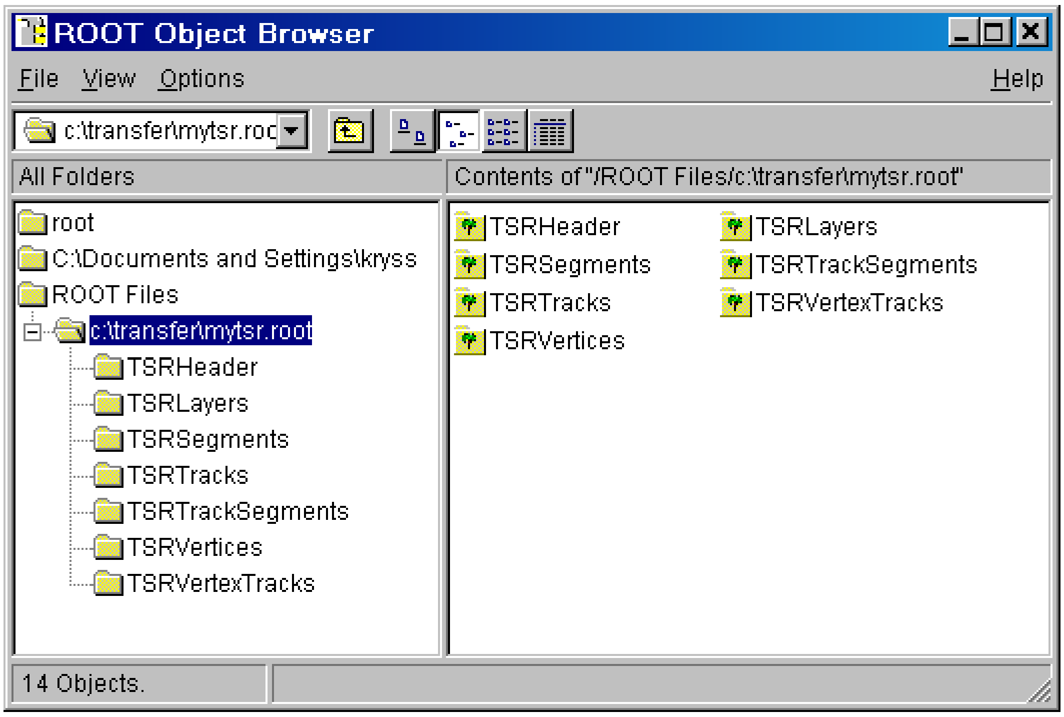
Structure name	Tree/Branch	Content
`TSRHeader`	TSRHeader/Data	Global information about a TSR file.
`TSRLayer`	TSRLayers/Data	Global information about one layer (=plate).
`TSRAlignment`	TSRLayers/Alignment	Geometrical transformations to align each layers with all the others.
`TSRSegment`	TSRSegments/Data	Information about one segment (=base track) of a track in a volume.
`TSRTrack`	TSRTracks/Data	Global information about a track in a volume spanning several layers (=plates).
`TSRTrackSegment`	TSRTrackSegments/Data	Indices that link each volume track to its segments.
`TSRVertex`	TSRVertices/Data	Global information about a vertex.
`TSRVertexTrack`	TSRVertexTracks/Data	Indices that link each vertex to the volume tracks that are attached to it.

It is worth to notice that these structures are defined inside the TSySalTSR, so the full name of a TSRTrack is, for example:

```
TSySalTSR::TSRTrack
```

This policy minimizes the probability of name clashing with other libraries.
Two structures, namely `TSRTrackSegments` and `TSRVertexTracks`, contain only indices that act as links. Strictly speaking, their presence is redundant, since `TSRSegment` contains the index of the owner track, and `TSRTrack` contains the indices of the upstream and downstream vertex. Nevertheless, they are very useful in common practice, since in some cases one wants to span quickly all segments in a layer, and in other cases one is spanning tracks and wants to refer to their segments without having to find them in the layers. The same is even more evident in the case of vertices, since each track can be connected to one, two vertices or none.

The trees that can be found in a ROOT file obtained through conversion can be read from the snapshot below.



The documentation of TSR data structures follows.

TSRHeader

```
struct TSRHeader
{
      Int_t Id0;
      Int_t Id1;
      Int_t Id2;
      Int_t Id3;
      Double_t MinX;
      Double_t MaxX;
      Double_t MinY;
      Double_t MaxY;
      Double_t MinZ;
      Double_t MaxZ;
      Int_t Sheets;
      Int_t Tracks;
      Int_t Vertices;
      Int_t MaxTracksInSegment;
      Double_t RefCenterX;
      Double_t RefCenterY;
      Double_t RefCenterZ;
      Int_t Segments;
};
```

Id0, Id1, Id2, Id3 are four numeric identifiers for the TSR.
(Min/Max)(X/Y/Z) are the coordinates of the extents of the volume.

Sheets is the number of layers (=plates) in the volume.
Tracks is the number of tracks in the volume.
Vertices is the number of vertices in the volume.
MaxTracksInSegment is the number of base tracks that can be merged into a single segment (to discard double measurements).
RefCenter(X/Y/Z) is the reference centre of the volume. All transformation have this point as a fixed pivot.
Segments is the total number of segments in all layers of the volume.

TSRLayer

```
struct TSRLayer
{
	Int_t Id;
	Int_t SheetId;
	Double_t DownstreamExt;
	Double_t DownstreamInt;
	Double_t UpstreamInt;
	Double_t UpstreamExt;
	Double_t RefCenterX;
	Double_t RefCenterY;
	Double_t RefCenterZ;
	Int_t Segments;
	Int_t FirstSegment;
};
```

Id is the index of the layer within the volume.
SheetId is the original identifier of the sheet.
(Down/Up)stream(Ext/Int) is the internal/external Z of the contact surface for the downstream/upstream emulsion layer of the sheet.
Segments is the number of segments in the layer.
FirstSegment is the index (within the volume) of the first segment in the layer. Since all segments are stored sequentially in the same tree, this information helps find quickly the sequence of segments that are associated to this layer.

TSRAlignment

```
struct TSRAlignment
{
	Double_t DeformationXX;
	Double_t DeformationXY;
	Double_t DeformationYX;
	Double_t DeformationYY;
	Double_t TranslationX;
	Double_t TranslationY;
	Double_t TranslationZ;
	Double_t DeltaSlopeX;
	Double_t DeltaSlopeY;
	Double_t DeltaShrinkX;
	Double_t DeltaShrinkY;
};
```

This structure contains the geometrical transformations that have been used to align the segments in the layer to segments in other layers.
The transformations for positions are specified by the following formulae:

$$x' = DeformationXX\,(x - x_r) + DeformationXY\,(y - y_r) + x_r + TranslationX$$
$$y' = DeformationYX\,(x - x_r) + DeformationYY\,(y - y_r) + y_r + TranslationY$$
$$z' = z + TranslationZ$$

where x_r is RefCenterX and y_r is RefCenterY.
The transformations for slopes are specified by the following formulae:

$$S_x^{'} = DeltaShrinkX\,(DeformationXX\ S_x + DeformationXY\ S_y) + DeltaSlopeX$$
$$S_y^{'} = DeltaShrinkY\,(DeformationYX\ S_x + DeformationYY\ S_y) + DeltaSlopeY$$

TSRSegment
```
struct TSRSegment
{
	Int_t AreaSum;
	Int_t Grains;
	Double_t InterceptX;
	Double_t InterceptY;
	Double_t InterceptZ;
	Double_t SlopeX;
	Double_t SlopeY;
	Double_t Sigma;
	Double_t TopZ;
	Double_t BottomZ;
	Int_t BaseTrackId;
	Int_t LayerOwnerId;
	Int_t LayerId;
	Int_t TrackOwnerId;
	Int_t TrackId;
	Int_t TrackLength;
};
```

AreaSum is the sum of the areas (in pixels) of all the grains in this track (top grains + bottom grains).
Grains is the total number of grains in this track (top + bottom).
InterceptX/Y/Z are the coordinate of one point laying on the fit line of the grains. This point is usually taken at the contact surface between the top emulsion layer and the base, but the Z coordinate has been added just to eliminate the need to make assumptions. The coordinates are in µm, and the coordinates for the X/Y plane are in the sheet reference frame.
SlopeX/Y are the X/Y components of the track slope.
Sigma is a track quality parameter. See the documentation of Sigma in TLGBaseTrack for more details.
TopZ/BottomZ Z coordinate of the top/bottom grain in the top/bottom side.
BaseTrackId index of the original base track in the TLG to which it belongs.

LayerOwnerId index of the layer to which this segment belongs.
LayerId sequential index (within the layer) of this segment.
TrackOwnerId index of the track to which this segment belongs. The segment is not attached to any track if this index is < 0.
TrackId sequential index (within the track) of this segment.

TSRTrack

```
struct TSRTrack
{
	Double_t DownstreamZeroX;
	Double_t DownstreamZeroY;
	Double_t DownstreamSlopeX;
	Double_t DownstreamSlopeY;
	Double_t UpstreamZeroX;
	Double_t UpstreamZeroY;
	Double_t UpstreamSlopeX;
	Double_t UpstreamSlopeY;
	Int_t Id;
	Int_t Segments;
	Int_t FirstSegment;
	Int_t UpstreamVertexId;
	Int_t DownstreamVertexId;
};
```

(Down/Up)streamSlope(X/Y) X/Y slope obtained from the fit of the most downstream/upstream segments of the track.
(Down/Up)streamZero(X/Y) X/Y position, projected at Z = 0, obtained from the fit of the most downstream/upstream segments of the track.
Id sequential index of the track within the volume.
Segments number of segments in this track.
FirstSegment sequential number of the first TSRTrackSegment in the tree of TrackSegments that points to the segments attached to this track. Since the number of segments in a track is a variable quantity, a TSRTrack cannot point directly to all of its segments. It can instead point to the beginning of the list that contains the pointers (indices) to its segments.
(Down/Up)streamVertexId Id of the vertex that is attached to the downstream/upstream end of the track. If an end is attached to no vertex, the Id is < 0.

TSRTrackSegment

```
struct TSRTrackSegment
{
	Int_t LayerOwnerId;
	Int_t LayerId;
};
```

LayerOwnerId index of the layer to which the segment belongs.
LayerId sequential index (within the layer) of the segment.

TSRVertex
```
struct TSRVertex
{
     Double_t X;
     Double_t Y;
     Double_t Z;
     Double_t AverageDistance;
     Double_t DeltaX;
     Double_t DeltaY;
     Int_t Id;
     Int_t Tracks;
     Int_t FirstTrack;
};
```

X/Y/Z coordinates of the vertex.
AverageDistance average distance of tracks from the vertex point.
Delta(X/Y) uncertainty on the transverse coordinate of the vertex.
Id sequential index of this vertex within the volume.
Tracks number of tracks attached to this vertex.
FirstTrack sequential index of the first TSRVertexTrack in the tree of TSRVertexTracks that points to the tracks attached to this vertex. Since the number of tracks attached to a vertex can vary, a TSRVertex cannot point directly to all of its tracks. It can instead point to the beginning of the list that contains pointers (indices) to the tracks meeting at the vertex point.

TSRVertexTrack
```
struct TSRVertexTrack
{
     Int_t TrackId;
     Int_t IsUpstream;
};
```

TrackId index of the track attached to the vertex.
IsUpstream zero if the vertex is downstream of the track, nonzero if it is upstream.

5 Index