VisIVOServer 1.2

User Guide

Introduction

VisIVO Server is a suite of software tools for creating customized views of 3D renderings from astrophysical data tables. These tools are founded on the **VisIVO Desktop** functionality (visivo.oact.inaf.it) and support the most popular Linux based platforms (e.g. www.ubuntu.com). Their defining characteristic is that no fixed limits are prescribed regarding the dimensionality of data tables input for processing, thus supporting very large scale datasets.

VisIVO Server websites are currently hosted by the University of Portsmouth, UK (visivo.port.ac.uk), the INAF Astrophysical Observatory of Catania, Italy (visivo.oact.inaf.it) and in the near future by CINECA, Italy (visivo.cineca.it). These web sites offer data management functionality for registered users; datasets can be uploaded for temporary storage and processing for a period of up to two months. The sites can also be utilized through anonymous access in which case datasets can be uploaded and stored for a maximum of four days; to maximize available resources a limited dimensionality is only supported.

Assuming that datasets are uploaded, users are typically presented with tree-like structures (for easy data navigation) containing pointers to **files**, **tables**, **volumes** as well as **visuals**.

Files point to single, or possibly several (for distributed datasets), astrophysical data tables;
Tables are highly-efficient internal VisIVO Server data representations; they are typically produced from importing datasets uploaded by users using VisIVO Importer (see below);
Volumes are internal VisIVO Server data representations; they are produced either from direct importing of user datasets or by performing operations on already existing tables;
Visuals are collections of highly-customized, user-produced views of 3D renderings of volumes.

VisIVO Server consists of three core components: **VisIVO Importer**, **VisIVO Filter** and **VisIVO Viewer** respectively. Their functionality and usage is described in the following sections.

To create customized views of 3D renderings from astrophysical data tables, a two-stage process is employed. First, VisIVO Importer is utilized to convert user datasets into **VisIVO Binary Tables** (VBTs). Then, VisIVO Viewer is invoked to display customized views of 3D renderings. As an example, consider displaying views from only three columns of an

astrophysical data table supplied in ascii form, say col_1, col_2 and col_3, by using the commands

```
VisIVOImporter --fformat ascii UserDataSet.txt

VisIVOViewer -x col_1 -y col_2 -z col_3 --scale --glyphs pixel VBT.bin
```

VisIVOServer is distributed with GPL V.2 License for NON COMMERCIAL use. VisIVOServer is hosted by sourceforge https://sourceforge.net/projects/visivoserver/ and its source code is downloadable via svn:

svn co https://visivoserver.svn.sourceforge.net/svnroot/visivoserver/branches/1.2 visivoserver

Disclaimer: user data integrity is never warranted.

VisIVO BINARY TABLE

A VisIVO Binary Table (VBT) is a highly-efficient data representation used by VisIVO Server internally. A VBT is realized through a header file (extension **.bin.head**) containing all necessary metadata, and a raw data file (extension **.bin**) storing actual data values. For example, the header may contain information regarding the overall number of fields and number of points for each field (for point datasets) or the number of cells and relevant mesh sizes (for volume datasets). The raw data file is typically a sequence of values, e.g. all X followed by all Y values.

Header

The header file contains the following fields:

```
float | double
n1
n2 [ GeoX GeoY GeoZ DX DY DZ ]
little | big
X
Y
Z
Vx
Vy
Vz
```

- **`float | double`** is the data type of the storage variables used;
- **`n1`** denotes the number of columns (fields) in the VBT;
- **`n2`** denotes the number of rows in the VBT;

- ***Geo*X GeoY GeoZ DX DY DZ** are employed only if the VBT represents volumetric datasets. In that case GeoX, GeoY and GeoZ represent the mesh geometry, while DX, DY and DZ represent the x, y and z size of volumetric cells.
- **little | big** denotes the endianism employed in the VBT. After this field there exist n1 rows that indicate the VBT columns as positions (X, Y, Z) and velocities (Vx, Vy, Vz).

Raw

The binary file is simply a sequence of *n1*n2* values. In the example shown in section 2.1 all X values, then all Y values and so on.

Note:

- **n1** represents the number of columns (fields) in the VBT (e.g. 6);
- **n2** represents the number of elements of each field in the VBT (e.g. 262144);
- **GeoX GeoY GeoZ** represent the number of the volumetric cells in each dimension of the mesh size (used only for Volumes) (e.g. 64 64 64);
- **DX DY DZ** represent the size of each cell (used only for Volumes) (e.g. 1.0 1.0 1.0)

VisIVOServer Library
Version 1.2

VisIVO Library is designed to directly use the VisIVO Server features within the user code. The full documentation on the specific functionalities of VisIVO Server is given in the specific VisIVO Server User Guides.

There are four library sections:

1) VisIVOImporter: contains all the environment variable sets to import an user file into a VBT

2) VisIVOFilter: contains all the environment variable sets to use the VisIVO Filters on a VBT

3) VisIVOViewer: contains all the environment variable sets to create an image or a movie from a VBT.

4) General Utilities.

Images and movies can be produced from an existing user file or directly from internal arrays.

VisIVO Library is organized with *environments* that are represented with a variable. More than one environment may exist for each section. Typically to set the environment the user must declare the variable (e.g. *VisIVOViewer env3;)* and initialize it (*VV_Init(&env3*)). The variable contains all settings for the operation. The *env3* variable will contain the camera position or a movie camera path, the palette, the filename or the arrays pointers used to create the image. The image creation will be done by the function VV_View(&env3).
The specific actions are executed by the following functions:

- *VI_Importer(&env1)* where env1 is a variable *VisIVOImporter* type;
- *VF_Filter(&env2)* where env2 is a variable *VisIVOFilter* type;
- *VV_View(&env3)* where env3 is a variable *VisIVOViewer* type.

All the functions of a sections starts with a prefix ***VI_*** ,***VF_*** or ***VV_*** respectively for Importer Filter and Viewer sections. There are also some general utilities, for direct actions on data. These utilities start with ***VS_*** prefix.

The corresponding functions, *VA_Importer(&env1) VA_Filter(&env2) VA_View(&env3)* are non blocking functions and they are described in the section VisIVO Api MultiThread.

This user manual is organized as follows: The first four sections are for VisIVO Importer, VisIVO Filters, VisIVO Viewer and General Utilities. The last section contains some examples on how to call the VisIVO Library to produce image from a file and from arrays pointers.

VisIVO Library Importer

The environment of this section has the main purpose to create a VBT from an user file specified with VV_SET_FILEPATH attribute. The file can be located in a remote server. The generated VBT is used from Filters and from Viewer to produce images and movies. The environment can be also used for create VBT used directly in Generic functions, that are never stored.

All the generated VBTs in an environment are removed only by VI_Clean(&env) function.

All the functions and the parameters are referred to specific parameters options and functionalities described in detail on the VisIVO Importer User Guide, here shortly recalled.

VisIVO may be compiled on a gLite system. In this case the option *GLITE* must be given for compile the tool.
When running on gLite system:

1. the option –-VO is to specifying the virtual organization (VO) name.
2. the input user file, to be imported, can be a logical filename (lfn) and must start with *lfn:// .*
3. the output VBT, can be local (-- out option) or the output lfn can be given. In any case the –-out is the local filename where a temporary VBT will be created
4. - -lfnout: is the output logical filename. In this case the temporary VBT will be saved in the grid catalogue and deleted from local system.
5. the option –-se is to specify the Storage Element (SE)

Environment Declaration

VisIVOImporter env;

env represent the environment. It is a structure that contain all information on how to create a VBT from an user file. The environment must be set using only the specific functions.

Environment Initialization

int VI_Init(VisIVOImporter *env) ;

The environment must be initialized before any usage.

Environment Set

int VI_SetAtt(VisIVOImporter *env, int code, char *value)

Set the environment attributes.
The environment is set by a code that specifies an attribute and a value that is used for the attribute. The following code are availables with the corresponding VisIVO Importer options

VI_SET_BIGENDIAN
Option: ***--bigendian*** *(Gadget and FLY data format only)*
Call: **VI_SetAtt(&env, VI_SET_BIGENDIAN,"");**

VI_SET_BINARYHEADER
Option: ***--binaryheader*** *(original VBT data format only)*
Call: **VI_SetAtt(&env, VI_SET_BINARYHEADER,"");**

VI_SET_DATASETLIST
Option: ***--datasetlist*** *(hdf5 data format only)*
Call: **VI_SetAtt(&env, VI_SET_DATASETLIST,"dataset1 dataset2");**

VI_SET_DOUBLE
Option: ***--double*** *(FLY data format only)*
Call: **VI_SetAtt(&env, VI_SET_DOUBLE,"");**

VI_SET_FFORMAT
Option: ***--fformat*** *(**Mandatory**)*
Call: **VI_SetAtt(&env, VI_SET_FFORMAT,"ascii");**

VI_SET_FILEPATH
Option: Input data file including path *(**Mandatory**)*
Call: **VI_SetAtt(&env, VI_SET_FILEPATH,"/home/user/mydata.asc");**
Call: **VI_SetAtt(&env, VI_SET_FILEPATH,"**ftp://server.oact.inaf.it/user/mydata.asc**");**

NOTE: if VisIVO Library is compiled with the LIGHT version the "sft://" is not allowed.

VI_SET_HYPERSLAB
Option: ***--hyperslab*** *(hdf5 data format only)*
Call: **VI_SetAtt(&env, VI_SET_HYPERSLAB," dataset 10,10 100,100");**
Note: The value filed must contain the dataset name the offset and the count. See VisIVOImporter User Guide for more details.

VI_SET_MISSINGVALUE
Option: ***--missingvalue***
Call: **VI_SetAtt(&env, VI_SET_MISSINGVALUE,"10.4e-10");**

VI_SET_NPOINTS
Option: ***--npoints*** *(FLY data format only)*
Call: **VI_SetAtt(&env, VI_SET_NPOINTS,"262144");**

VI_SET_OUTFILEVBT
Option: ***--out*** *(Not requested only in case of Generic Function)*
Call: **VI_SetAtt(&env, VI_SET_OUTFILEVBT,"/home/user/myvbt.bin");**

VI_SET_TEXTVALUE
Option: ***--textvalue***
Call: **VI_SetAtt(&env, VI_SET_TEXTVALUE,"10.4e-10");**

VI_SET_USERPWD
Option: ***--userpwd***
Call: **VI_SetAtt(&env, VI_SET_USERPWD,"myuser:mypassword");**

VI_SET_VOLUME
Option: ***--volume*** *(Mandatory in case of volume)*
Call: **VI_SetAtt(&env, VI_SET_VOLUME,"64 64 64 1.0 1.0 1.0");**
Note: in the value filed must be specified three integers that represent the number of cells in each direction and three float values that represent the cell size. All cells have the same size. These values correspond to the –compx, --compy, --compz, --sizex, --sizey and --sizez

VI_SET_VO
Option: ***--VO*** *(Mandatory in case of gLite grid catalogue usage)*
Call: **VI_SetAtt(&env, VI_SET_VO,"alice");**
Note: in the value filed must be specified the Virtual Organization used on gLite.

VI_SET_LFNOUT
Option: ***--lfnout***
Call: **VI_SetAtt(&env, VI_SET_LFNOUT,"lfn://grid/cometa/user/mytab.bin");**
Note: it is used to set the logical filename (lfn) when running on gLite grid. A logical filename starts with lfn://.

VI_SET_SE
Option: ***--se***
Call: **VI_SetAtt(&env, VI_SET_SE,"grid-se-01.ct.inaf.it");**
Note: it is used to set the storage element.

int VI_Import(VisIVOImporter *env)

This call execute the importer operation with the options set by the env variable, and creates a VBT from the user file specified with *VI_SET_FILEPATH* attribute.

Environment Close

int VI_Clean(VisIVOImporter *env)

This call execute delete all options set by in the env variable, and delete the VBT created by the *VI_Import* function.

Returning Values

The above described functions return one of the following error code.

noError	**0**
invalidParCode	**1**
invalidFFormatCode	**2**
invalidPathCreation	**4**
invalidImporterOptions	**9**
invalidImporterOperation	**10**
invaldInputFile	**16**

Examples

User programs that want to use the library, must include visivo.h.

1) Importing an hdf5 user File

```
#include "visivo.h"
…
int main(int argc, char*argv[])
{

VisIVOImporter env1;
int errorCode;
errorCode=VI_Init(&env1);

errorCode=VI_SetAtt(&env1,VI_SET_FFORMAT,"hdf5");
errorCode=VI_SetAtt(&env1, VI_SET_DATASETLIST,"dataset1 dataset2");
errorCode=VI_SetAtt(&env1, VI_SET_HYPERSLAB,"dataset2 10,10 100,100");
errorCode=VI_SetAtt(&env1,VI_SET_FILEPATH,"HDF5UserFile");
errorCode=VI_SetAtt(&env1,VI_SET_OUTFILEVBT,"NewTable.bin");
errorCode=VI_Import(&env1);
...
}
```

The above code correspond to the following command:

VisIVOImporter --fformat hdf5 --datasetlist datset1 dataset2 --hyperslab dataset2 10,10 100,100 --out /home/user/data/NewTable HDF5UserFile

2) Importing a VOTable user File

```
#include "visivo.h"
…
int main(int argc, char*argv[])
{

VisIVOImporter env1;
int errorCode;
errorCode=VI_Init(&env1);

errorCode=VI_SetAtt(&env1,VI_SET_FFORMAT,"votable");
errorCode=VI_SetAtt(&env1,VI_SET_FILEPATH,"VOTableUserFilename.xml");
errorCode=VI_SetAtt(&env1,VI_SET_OUTFILEVBT,"/home/user/dataNewTable.bin");
errorCode=VI_Import(&env1);
```

...
}

The above code correspond to the following command:

VisIVOImporter --fformat votable --out /home/user/dataNewTable.bin VOTableUserFilename.xml

VisIVO Library Filters

The environment of this section has the main purpose to perform operations on a VBT specified with VF_SET_FILEVBT attribute or in a list of VBTs
The new or modified VBT can be used from Viewer to produce images and movies.

All the created VBTs in an environment are removed only by VF_Clean(&env) function.

All the functions and the parameters are referred to specific parameters options and functionalities described in detail on the VisIVO Filter User Guide.

Environment Declaration

VisIVOFilter env;

env represent the environment. It is a structure that contain all information on how to modify or create a new VBT. The environment must be set using only the specific functions.

Environment Initialization

int VF_Init(VisIVOFilter *env) ;

The environment must be initialized before any usage.

Environment Set

int VF_SetAtt(VisIVOFilter *env, int code, char *value)

Set the environment attributes.
The environment is set by a code that specifies an attribute and a value that is used for the attribute. The following code are availables with the corresponding VisIVO Filter options

GENERIC OPTIONS

In the following we report the attribute that are used by many filters

VF_SET_OPERATION
Option: ***--op*** *(**Mandatory**)*
Call: **VF_SetAtt(&env, VF_SET_OPERATION,"addId");**
Note: in the value filed must be specified one filter operation with the same name as reported in VisIVO Filter User Guide.

VF_SET_ALGORITHMTSC
VF_SET_ALGORITHMNGP
VF_SET_ALGORITHMCIC
Option: ***--tsc*** or ***–ngp*** or ***--cic***

Call: **VF_SetAtt(&env, VF_SET_ALGORITHMNGP,"");**
Note: it is used for algorithm data distribution when grid domain is used.

VF_SET_APPEND
Option: ***--append***
Call: **VF_SetAtt(&env, VF_SET_APPEND,"");**
Note: it is used for append new columns to the VBT

VF_SET_FIELD
Option: ***--field***
Call: **VF_SetAtt(&env, VF_SET_FIELD,"Col1 Col2 Col3");**
Note: it is used for select VBT columns.

VF_SET_FILEVBT
Option: ***--file***
Call: **VF_SetAtt(&env, VF_SET_FILEVBT,"myVBT.bin");**
Note: it specifies the VBT where filter will be applied.

VF_SET_GRIDORIGIN
Option: ***--gridOrigin***
Call: **VF_SetAtt(&env, VF_SET_GRIDORIGIN,"0 0 0");**
Note: it specifies the starting point of a grid domain

VF_SET_GRIDSPACING
Option: ***--gridSpacing***
Call: **VF_SetAtt(&env, VF_SET_GRIDSPACING,"1.0 1.0 1.0");**
Note: it specifies the mesh-cell grid size.

VF_SET_LIMITS
Option: ***--limits*** limitsFile.txt
Call: **VF_SetAtt(&env, VF_SET_LIMITS," X 10.0 20.0");**
Call: **VF_SetAtt(&env, VF_SET_LIMITS," Y 15.0 25.0");**
Call: **VF_SetAtt(&env, VF_SET_LIMITS," Z 20.0 30.0");**
Note: It must be call for each limit on a field the user want set. In this example it is equivalent to have a file (limitsFile.txt) with the following rows
X 10.0 20.0
Y 15.0 25.0
Z 20.0 30.0

VF_SET_ LIMITSPURGE
Option:
Call: **VF_SetAtt(&env, VF_SET_LIMITSPURGE,"");**
Note: remove limits from the environment.

VF_SET_NOAPPEND
Option:
Call: **VF_SetAtt(&env, VF_SET_NOAPPEND,"");**
Note: it is used for remove the *--append* option from the environment

VF_SET_OUTCOL
Option: ***--outcol***
Call: **VF_SetAtt(&env, VF_SET_OUTCOL,"NewCol1 NewCol2");**

Note: the value field contains the new column names in a VBT

VF_SET_OUTVBT
Option: ***--out***
Call: **VF_SetAtt(&env, VF_SET_OUT,"NewVBT.bin");**
Note: the value field contains the new VBT filename

VF_SET_OPERATORAND
VF_SET_OPERATOROR
Option: ***--operator AND/OR***
Call: **VF_SetAtt(&env, VF_SET_OPERATORAND,"");**

VF_SET_POINTCOLUMNS
Option: ***--points***
Call: **VF_SetAtt(&env, VF_SET_POINTCOLUMNS,"X Y Z");**
Note: the value field contain three columns names to be assumed as coordinates system for points.

VF_SET_PERIODIC
Option: ***--periodic***
Call: **VF_SetAtt(&env, VF_SET_PERIODIC,"");**
Note: set periodic boundary condition in the environment.

VF_SET_NOPERIODIC
Option:
Call: **VF_SetAtt(&env, VF_SET_NOPERIODIC,"");**
Note: remove the periodic boundary condition in the environment.

VF_SET_RESOLUTION
Option: ***--resolution***
Call: **VF_SetAtt(&env, VF_SET_RESOLUTION,"32 32 32");**
Note: set a grid-mesh resolution on three dimension.

VF_SET_VO
Option: ***--VO*** *(Mandatory in case of gLite grid catalogue usage)*
Call: **VF_SetAtt(&env, VF_SET_VO,"alice");**
Note: in the value filed must be specified the Virtual Organization used on gLite.

VF_SET_LFNOUT
Option: ***--lfnout***
Call: **VF_SetAtt(&env, VF_SET_LFNOUT,"lfn://grid/cometa/user/mytab.bin");**
Note: it is used to set the logical filename (lfn) when running on gLite grid. A logical filename starts with lfn://.

VF_SET_SE
Option: ***--se***
Call: **VF_SetAtt(&env, VF_SET_SE,"grid-se-01.ct.inaf.it");**
Note: it is used to set the storage element.

SPECIFIC OPTIONS

In the following we report all filters and the specific defined options

Filter: Add Identifier

Generic Options:
--op, VF_SET_OPERATION (i.e. *VF_SetAtt(&env, VF_SET_OPERATION,"addId");*)
--outcol, VF_SET_OUTCOL
--file, VF_SET_FILEVBT

Specific Options:
VF_SET_ADDIDSTART
Option: ***--start***
Call: **VF_SetAtt(&env, VF_SET_ADDIDSTART,"100");**
Note: set the sarting Id.

Filter: AHFstep

Not implemented in the Library

Filter: AHF Galaxy Extraction

Not implemented in the Library

Filter: AHF Halos List

Not implemented in the Library

Filter: Append Tables

Generic Options:
--op, VF_SET_OPERATION (i.e. *VF_SetAtt(&env, VF_SET_OPERATION,"append");*)
--out, VF_SET_OUTVBT

Specific Options:
VF_SET_APPENDLIST
Option: ***--filelist*** *filelist.txt*
Call: **VF_SetAtt(&env, VF_SET_APPENDLIST,"VBT1.bin");**
Call: **VF_SetAtt(&env, VF_SET_APPENDLIST,"VBT2.bin");**
Call: **VF_SetAtt(&env, VF_SET_APPENDLIST,"VBT3.bin");**
Note: It must be call for each VBT the user want append. In this example it is equivalent to have a file (filelist.txt) with the following rows:
VBT1.bin
VBT2.bin
VBT3.bin
VF_SET_ APPENDLISTSPURGE
Option:
Call: **VF_SetAtt(&env, VF_SET_APPENDLISTPURGE,"");**
Note: remove append list from the environment.

Filter: Cartesian2polar

Generic Options:
--op, VF_SET_OPERATION (i.e. *VF_SetAtt(&env, VF_SET_OPERATION,"cartesian2polar");*)
--field, VF_SET_FIELD
--append VF_SET_APPEND
--outcol, VF_SET_OUTCOL
--file, VF_SET_FILEVBT

Specific Options:
NONE

Filter: Change column name

Generic Options:
--op, VF_SET_OPERATION (i.e. *VF_SetAtt(&env, VF_SET_OPERATION,"changecolname");*)
--field, VF_SET_FIELD
--file, VF_SET_FILEVBT

Specific Options:
VF_SET_NEWCOLNAMES
Option: ***--newnames***
Call: **VF_SetAtt(&env, VF_SET_NEWCOLNAMES,"NewCol1 NewCol2");**

Filter: Coarse Volume

Generic Options:
--op, VF_SET_OPERATION (i.e. *VF_SetAtt(&env, VF_SET_OPERATION,"coarsevolume");*)
--field, VF_SET_FIELD
--out, VF_SET_OUTVBT
--file, VF_SET_FILEVBT

Specific Options:
VF_SET_VOLUMEPERC
Option: ***--perc***
Call: **VF_SetAtt(&env, VF_SET_VOLUMEPERC,"10");**

VF_SET_NEWRES
Option: ***--newres***
Call: **VF_SetAtt(&env, VF_SET_NEWRES,"32 32 32");**

Filter: Cut

Generic Options:
--op, VF_SET_OPERATION (i.e. *VF_SetAtt(&env, VF_SET_OPERATION,"cut");*)
--field, VF_SET_FIELD
--limits VF_SET_LIMITS
--operator, VF_SET_OPERATOROR, VF_SET_OPERATORAND
--out , VF_SET_OUTVBT

--file, VF_SET_FILEVBT

Specific Options:
VF_SET_CUTTHRESHOLD
Option: ***--threshold***
Call: **VF_SetAtt(&env, VF_SET_CUTTHRESHOLD,"100");**

Filter: Decimator

Generic Options:
--op, VF_SET_OPERATION (i.e. *VF_SetAtt(&env, VF_SET_OPERATION,"decimator");*)
--file, VF_SET_FILEVBT

Specific Options:
VF_SET_DECIMATORSKIP
Option: ***--threshold***
Call: **VF_SetAtt(&env, VF_SET_DECIMATORSKIP,"10");**

Filter: Extract List

Not implemented in the Library

Filter: Extract Subregion

Generic Options:
--op, VF_SET_OPERATION (i.e. *VF_SetAtt(&env, VF_SET_OPERATION,"extraction");*)
--out , VF_SET_OUTVBT
--file, VF_SET_FILEVBT

Specific Options:
VF_SET_EXTRACTIONGEOMETRY
Option: ***-- geometry*** geometryFile.txt

Call: **VF_SetAtt(&env, VF_SET_EXTRACTIONGEOMETRY,"X 20.0");**
Call: **VF_SetAtt(&env, VF_SET_EXTRACTIONGEOMETRY,"Y 30.0");**
Call: **VF_SetAtt(&env, VF_SET_EXTRACTIONGEOMETRY,"Z 40.0");**
Call: **VF_SetAtt(&env, VF_SET_EXTRACTIONGEOMETRY,"RADIUS 20.0");**
Note: It must be call four times. The first three calls are to set a point in the system coordinate. The last call is to fix the geometry. See VisIVO Filters User Guide for more details.

VF_SET_ EXTRACTIONGEOMETRYPURGE
Option:
Call: **VF_SetAtt(&env, VF_SET_EXTRACTIONGEOMETRYPURGE,"");**
Note: remove the geometry from the environment .

Filter: Extract Subvolume

Generic Options:
--op, VF_SET_OPERATION (i.e. *VF_SetAtt(&env, VF_SET_OPERATION,"extractsubvolume");)*

--resolution, VF_SET_RESOLUTION
--field, VF_SET_FIELD
--out , VF_SET_OUTVBT
--file, VF_SET_FILEVBT

Specific Options:
VF_SET_STARTINGCELL
Option: ***--startingcell***
Call: **VF_SetAtt(&env, VF_SET_STARTINGCELL,"10 10 10");**

Filter: Grid2Point

Generic Options:
--op, VF_SET_OPERATION (i.e. *VF_SetAtt(&env, VF_SET_OPERATION,"grid2point");)*
--points, VF_SET_POINTCOLUMNS
--field, VF_SET_FIELD
--append, VF_SET_APPEND
--out , VF_SET_OUTVBT
--outcol, VF_SET_OUTCOL
--tsc,--ngp, VF_SET_ALGORITHMTSC, VF_SET_ALGORITHMNGP, VF_SET_ALGORITHMCIC
--gridOrigin, VF_SET_GRIDORIGIN
--gridSpacing, VF_SET_GRISPACING
--box, VF_SET_BOX
--file, VF_SET_FILEVBT

Specific Options:
VF_SET_DENSITY
Option: ***--density***
Call: **VF_SetAtt(&env, VF_SET_DENSITY,"");**
VF_SET_VOLUME
Option: ***--volume***
Call: **VF_SetAtt(&env, VF_SET_VOLUME,"Volume.VBT.bin");**
Note: input data volume VBT. See VisIVO Filters User Guide for more details.

Filter: Interpolate

Generic Options:
--op, VF_SET_OPERATION (i.e. *VF_SetAtt(&env, VF_SET_OPERATION,"interpolate");)*
--field, VF_SET_FIELD
--out , VF_SET_OUTVBT (root name of the generated VBTs)

Specific Options:
VF_SET_NUMBIN
Option: ***--numbin***
Call: **VF_SetAtt(&env, VF_SET_NUMBIN,"20");**
VF_SET_PERIODIC
Option: ***--periodic***
Call: **VF_SetAtt(&env, VF_SET_PERIODIC,"");**
VF_SET_INTERVAL

Option: ***--interval***
Call: **VF_SetAtt(&env, VF_SET_INTERVAL,"0.0 1.0");**
VF_SET_INFILES
Option: ***--infiles***
Call: **VF_SetAtt(&env, VF_SET_INFILES,"VBTFrame1.bin VBTFrame2.bin");**
VF_SET_NOPERIODIC
Option:
Note: remove the *--periodic* option from the environment.

Filter: Include

Generic Options:
--op, VF_SET_OPERATION (i.e. *VF_SetAtt(&env, VF_SET_OPERATION,"include");)*
--field, VF_SET_FIELD
--append, VF_SET_APPEND
--out , VF_SET_OUTVBT
--outcol, VF_SET_OUTCOL
--file, VF_SET_FILEVBT

Specific Options:
VF_SET_CENTER
Option: ***--center***
Call: **VF_SetAtt(&env, VF_SET_CENTER,"1.0 1.0 1.0");**
VF_SET_RADIUS
Option: ***--radius***
Call: **VF_SetAtt(&env, VF_SET_RADIUS,"25.0");**
VF_SET_OUTVALUE
Option: ***--outvalue***
Call: **VF_SetAtt(&env, VF_SET_INTERVAL,"0.0");**
VF_SET_INVALUE
Option: ***--invalue***
Call: **VF_SetAtt(&env, VF_SET_INTERVAL,"1.0");**

Filter: Mathematical Operations

Generic Options:
--op, VF_SET_OPERATION (i.e. *VF_SetAtt(&env, VF_SET_OPERATION,"mathop");)*
--append, VF_SET_APPEND
--out , VF_SET_OUTVBT
--outcol, VF_SET_OUTCOL
--file, VF_SET_FILEVBT

Specific Options:
VF_SET_MATEXPRESSION
Option: ***--compute***
Call: **VF_SetAtt(&env, VF_SET_MATEXPRESSION,"sqrt(A*B)");**
Note: The value argument must contain a valid mathematical expression using VBT column names. See VisIVO Filters User Guide form more detail. The *--expression* option is not necessary in the library.

Filter: Merge Tables

Generic Options:
--op, VF_SET_OPERATION (i.e. *VF_SetAtt(&env, VF_SET_OPERATION,"merge");)*

Specific Options:
VF_SET_MERGEHUGE, VF_SET_MERGESMALL
Option: ***--size HUGE, --size SMALLEST***
Call: **VF_SetAtt(&env, VF_SET_MERGEHUGE,"");**
VF_SET_MERGEPAD
Option: ***--pad***
Call: **VF_SetAtt(&env, VF_SET_MERGEPAD,"0");**
VF_SET_MERGELIST
Option: ***--filelist***
Call: **VF_SetAtt(&env, VF_SET_MERGELIST,"VBT1.bin Col_1");**
Call: **VF_SetAtt(&env, VF_SET_MERGELIST,"VBT2.bin Col_2");**
Call: **VF_SetAtt(&env, VF_SET_MERGELIST,"VBT3.bin *");**
Note: It can be call several times. The value argument must contain two elements: the VBT filename and the Column (or wildcard). See VisIVO Filters User Guide for more details.
VF_SET_MERGELISTPURGE
Option:
Call: **VF_SetAtt(&env, VF_SET_MERGELISTPURGE,"");**
Note: remove the --*filelist* option from the environment.

Filter: Module

Generic Options:
--op, VF_SET_OPERATION (i.e. *VF_SetAtt(&env, VF_SET_OPERATION,"module");)*
--field, VF_SET_FIELD
--append, VF_SET_APPEND
--out , VF_SET_OUTVBT
--outcol, VF_SET_OUTCOL
--file, VF_SET_FILEVBT

Specific Options:
NONE

Filter: Point Distribute

Generic Options:
--op, VF_SET_OPERATION (i.e. *VF_SetAtt(&env, VF_SET_OPERATION,"pointdistribute");)*
--resolution, VF_SET_RESOLUTION
--points, VF_SET_POINTCOLUMNS
--field, VF_SET_FIELD
--append, VF_SET_APPEND
--out , VF_SET_OUTVBT
--outcol, VF_SET_OUTCOL
--tsc,--ngp, VF_SET_ALGORITHMTSC, VF_SET_ALGORITHMNGP, VF_SET_ALGORITHMCIC
--gridOrigin, VF_SET_GRIDORIGIN
--gridSpacing, VF_SET_GRISPACING

--periodic, VF_SET_PERIODIC
--file, VF_SET_FILEVBT

Specific Options:
VF_SET_NODENSITY
Option: ***--nodensity***
Call: **VF_SetAtt(&env, VF_SET_NODENSITY,"");**
VF_SET_AVG
Option: ***--avg***
Call: **VF_SetAtt(&env, VF_SET_AVG,"");**

Filter: Point Property

Generic Options:
--op, VF_SET_OPERATION (i.e. *VF_SetAtt(&env, VF_SET_OPERATION,"pointproperty");)*
--resolution, VF_SET_RESOLUTION
--points, VF_SET_POINTCOLUMNS
--field, VF_SET_FIELD
--append, VF_SET_APPEND
--out , VF_SET_OUTVBT
--outcol, VF_SET_OUTCOL
--periodic, VF_SET_PERIODIC
--file, VF_SET_FILEVBT

Specific Options:
NONE

Filter: Randomizer

Generic Options:
--op, VF_SET_OPERATION (i.e. *VF_SetAtt(&env, VF_SET_OPERATION,"randomizer");)*
--field, VF_SET_FIELD
--out , VF_SET_OUTVBT
--file, VF_SET_FILEVBT

Specific Options:
VF_SET_RANDOMPERC
Option: ***--perc***
Call: **VF_SetAtt(&env, VF_SET_RANDOMPERC,"50.0");**
VF_SET_RANDOMSEED
Option: ***--iseed***
Call: **VF_SetAtt(&env, VF_SET_RANDOMSEED,"1");**

Filter: Select Columns

Generic Options:
--op, VF_SET_OPERATION (i.e. *VF_SetAtt(&env, VF_SET_OPERATION,"selcolumns");)*
--field, VF_SET_FIELD
--out , VF_SET_OUTVBT
--file, VF_SET_FILEVBT

Specific Options:
VF_SET_DELETECOLUMNS
Option: ***--delete***
Call: **VF_SetAtt(&env, VF_SET_DELETECOLUMNS,"");**

Filter: Select Rows

Generic Options:
--op, VF_SET_OPERATION (i.e. *VF_SetAtt(&env, VF_SET_OPERATION,"selfield");*)
--limits VF_SET_LIMITS
--operator, VF_SET_OPERATOROR, VF_SET_OPERATORAND
--out , VF_SET_OUTVBT
--file, VF_SET_FILEVBT

Specific Options:
NONE

Filter: Show Table

Generic Options:
--op, VF_SET_OPERATION (i.e. *VF_SetAtt(&env, VF_SET_OPERATION,"showtable");*)
--field, VF_SET_FIELD
--file, VF_SET_FILEVBT

Specific Options:
VF_SET_NUMROWS
Option: ***--numrows***
Call: **VF_SetAtt(&env, VF_SET_NUMROWS,"1000");**
VF_SET_RANGEROWS
Option: ***--rangerows***
Call: **VF_SetAtt(&env, VF_SET_RANGEROWS,"1 1000");**
VF_SET_WIDTH
Option: ***--width***
Call: **VF_SetAtt(&env, VF_SET_RANGEROWS,"10");**
VF_SET_PRECISION
Option: ***--precision***
Call: **VF_SetAtt(&env, VF_SET_RANGEROWS,"5");**
VF_SET_OUT
Option: ***--out***
Call: **VF_SetAtt(&env, VF_SET_OUT,"MyFile.ascii");**
Note: The *--out* option contain an ascii output filename

Filter: Show Volume

Generic Options:
--op, VF_SET_OPERATION (i.e. *VF_SetAtt(&env, VF_SET_OPERATION,"showvol");*)
--field, VF_SET_FIELD
--limits VF_SET_LIMITS

--operator, VF_SET_OPERATOROR, VF_SET_OPERATORAND
--file, VF_SET_FILEVBT
Note: The *--out* option contain an ascii output filename

Specific Options:
VF_SET_NUMCELLS
Option: ***--numcells***
Call: **VF_SetAtt(&env, VF_SET_NUMCELLS,"100");**
VF_SET_OUT
Option: ***--out***
Call: **VF_SetAtt(&env, VF_SET_OUT,"MyVolFile.ascii");**
Note: The *--out* option contain an ascii output filename

Filter: Sigma Contours

Generic Options:
--op, VF_SET_OPERATION (i.e. *VF_SetAtt(&env, VF_SET_OPERATION,"sigmacontours");*)
--field, VF_SET_FIELD
--file, VF_SET_FILEVBT
Note: The *--out* option contain an ascii output filename

Specific Options:
VF_SET_NSIGMA
Option: ***--nsigma***
Call: **VF_SetAtt(&env, VF_SET_NSIGMA,"3");**
VF_SET_ALLCOULMNS
Option: ***--allcolumns***
Call: **VF_SetAtt(&env, VF_SET_ALLCOLUMNS,"");**

Filter: Statistic

The VisIVO Library has a generic method to obtain the data values (range, average etc.) of a VBT. The following function

int VS_VBTMetaData(VBT *tab, int satistic ,char *value)

allow to obtain the values of a VBT. It is described in the Generic section of this manual.

Filter: Split Table

Generic Options:
--op, VF_SET_OPERATION (i.e. *VF_SetAtt(&env, VF_SET_OPERATION,"splittable");*)
--field, VF_SET_FIELD
--out, VF_SET_OUTVBT
--file, VF_SET_FILEVBT

Specific Options:
VF_SET_VOLUMESPLIT
Option: ***--volumesplit***
Call: **VF_SetAtt(&env, VF_SET_VOLUMESPLIT,"1");**

VF_SET_NUMOFTABLES
Option: ***--numoftables***
Call: **VF_SetAtt(&env, VF_SET_NUMOFTABLES,"2");**
VF_SET_MAXSIZETABLE
Option: ***--maxsizetable***
Call: **VF_SetAtt(&env, VF_SET_MAXSIZETABLE,"100");**
Note: The maxsizetable value is given in MegaBytes.
VF_SET_HUGESPLIT
Option: ***--hugesplit***
Call: **VF_SetAtt(&env, VF_SET_HUGESPLIT,"");**

Filter: Swap

Generic Options:
--op, VF_SET_OPERATION (i.e. *VF_SetAtt(&env, VF_SET_OPERATION,"swap");*)
--out, VF_SET_OUTVBT
--file, VF_SET_FILEVBT

Specific Options:
VF_SET_OVERRIDE
Option: ***--override***
Call: **VF_SetAtt(&env, VF_SET_OVERRIDE,"");**

Filter: VBT2AHF

Not implemented in the Library

Filter: Visual

Generic Options:
--op, VF_SET_OPERATION (i.e. *VF_SetAtt(&env, VF_SET_OPERATION,"visualop");*)
--out, VF_SET_OUTVBT

Specific Options:
VF_SET_VISUALSIZE
Option: ***--size***
Call: **VF_SetAtt(&env, VF_SET_VISUALSIZE,"2000000");**
VF_SET_VISUALLIST
Option: ***--filelist***
Call: **VF_SetAtt(&env, VF_SET_VISUALLIST,"VBT1.bin X");**
Call: **VF_SetAtt(&env, VF_SET_VISUALLIST,"VBT2.bin Y");**
Call: **VF_SetAtt(&env, VF_SET_VISUALLIST,"VBT3.bin *");**
Note: It can be call several times. The value argument must contain two elements: the VBT filename and the Column (or wildcard). See VisIVO Filters User Guide for more details.
VF_SET_VISUALLISTPURGE
Option:
Call: **VF_SetAtt(&env, VF_SET_VISUALLISTPURGE,"");**
Note: remove the *--filelist* option from the environment.

Filter: Write VOTable

Generic Options:
--op, VF_SET_OPERATION (i.e. *VF_SetAtt(&env, VF_SET_OPERATION,"wrvotable");*)
--field, VF_SET_FILED
--out, VF_SET_OUTVBT
--file, VF_SET_FILEVBT

Specific Options:
VF_SET_WRVOTABLEFORCE
Option: ***--force***
Call: **VF_SetAtt(&env, VF_SET_WRVOTABLEFORCE,"");**

Filter Operation

int VF_Filter(VisIVOFilter *env)

This call execute the filter operation with the options set by the env variable.

Environment Close

int VF_Clean(VisIVOFilter *env)

This call execute delete all options set by in the env variable, and delete the VBT created by the *VF_Filter* function.

Returning Values

The above described functions return one of the following error code.

noError	**0**
invalidParCode	**1**
invalidPathCreation	**4**
invalidFilterOptions	**7**
invalidFilterOperation	**8**

Example

User programs that want to use the library, must include visivo.h.

Randomize a very large user File

```
#include "visivo.h"
…
int main(int argc, char*argv[])
{

VisIVOImporter env1;
VisIVOFilter env2;
int errorCode;
char op[256];
char vbt[256];
char ranvbt[256];
```

```
float rperc=1.0;
char sperc[10];
sprintf(sperc,"%f",rperc);



strcpy(vbt,"VBT1.bin");
strcpy(ranvbt,"RANVBT1.bin");

errorCode=VI_Init(&env1);

errorCode=VI_SetAtt(&env1,VI_SET_FFORMAT,"ascii");
errorCode=VI_SetAtt(&env1,VI_SET_FILEPATH,"ASCIILargeUserFile");
errorCode=VI_SetAtt(&env1,VI_SET_OUTFILEVBT,vbt);
errorCode=VI_Import(&env1);

strcpy(op,"randomizer");
errorCode=VF_Init(&env2);
errorCode=VF_SetAtt(&env2,VF_SET_OPERATION,op);
errorCode=VF_SetAtt(&env2,VF_SET_FILEVBT,vbt);
errorCode=VF_SetAtt(&env2, VF_SET_RANDOMPERC,sperc);
errorCode=VF_SetAtt(&env2, VF_SET_OUTVBT,ranvbt);
errorCode=VF_Filter(&env2);
.....
VI_Clean(&env1);
VF_Clean(&env2);

….
}
```

VisIVO Library Viewer

The environment of this section has the main purpose to create images and movies (a collecion of numbered images to be externally mounted) from VBTs or from internal arrays.

All the created images are never diretcly removed. Intermediate and temporary files (starting with ***.VS*** are removed by VV_Clean(&env) function.

All the functions and the parameters that are referring to specific options and functionalities are described in detail on the VisIVO Viewer User Guide.

VisIVO Library allows the image generation directly using the arrays of the user program (see sect. *Internal User Arrays*). The specific function are described in the following. Examples are given.

This section is organized as follows:

- *VisIVO Viewer using external VBTs* section describes all the functions for the usage of externals VBT. The section
- *VisIVO Viewer using internal arrays* describes all the functions for the usage of internal user arrays (without any VBT).
- *VisIVO Viewer General Options,* describes all options that can be given to generate the images and movie.
- *VisIVO Viewer Specific Options,* describes all the specific options that can be given to generate the images and movie for datapoints, volume, vector and Splotch visualization
- *VisIVO Viewer Splotch,* describes all the specific options that can be given to generate the images and movie for Splotch visualization

All the options are described in detail in the VisIVO Viewer User Guide.

Environment Declaration

VisIVOViewer env;

env represent the environment. It is a structure that contain all information on how to create images or movies from a VBT. The environment must be set using only the specific functions.

Environment Initialization

int VV_Init(VisIVOViewer *env) ;

The environment must be initialized before any usage.

VisIVO Viewer using external VBTs

One or more of the following functions must be given to set the environment using external VBT.

int VV_SetAtt(VisIVOViewer *env, code,char *value);

The following codes and value must be given to set the environment with external VBT

errorCode=VV_SetAtt(&env,VV_SET_FILEVBT,"VBT.bin");
VBT.bin is the VisIVO binary table that will be used to generate the images.

errorCode=VV_SetAtt(&env,VV_SET_FIELD,"X Y Z");

X, Y and *Z* are three field of VBT, these represent the the points coordinates for data point display (options --x, --y and --z)

errorCode=VV_SetAtt(&env,VV_SET_VFIELD,"Vx Vy Vz");
Vx, Vy and *Vz* are three field of VBT, these represent the vector components for vector display (options --vx, --vy and --vz)

errorCode=VV_SetAtt(&env,VV_SET_ISOSURFIELD,"density");
density is a filed of the VBT used to generate the isosurface image of a volume (options --isosurfacefield)

errorCode=VV_SetAtt(&env,VV_SET_SLICEFIELD,"density");
density is a filed of the VBT used to generate the slice image of a volume (options --slicefield)

errorCode=VV_SetAtt(&env,VV_SET_VRENDERFIELD,"density");
density is a field of the VBT used to generate the volume renedering image of a volume (options --vrenderingfield)

errorCode=VV_SetAtt(&env,VV_SET_COLORSCALAR,"density");
density is a field of the VBT used as a scalar property for the palette (options --colorscalar).

errorCode=VV_SetAtt(&env,VV_SET_RADIUSSCALAR,"Prop1");
errorCode=VV_SetAtt(&env,VV_SET_HEIGHTSCALAR,"Prop2");
Prop1 and *Prop2* are fields of the VBT used as a scalar for radius in the glyphs data representation (options --radiusscalar and --heightscalar).

VisIVO Viewer using internal arrays

One or more of the following functions must be given to set the environment using internal code arrays and not a VBT.
Remark: If one of the following function is called, the external VBT is always ignored. This means that the two methods cannot be mixed. The external reference can be set again using the VV_SetAtt(&env,VV_SET_EXTERNAL,"") described in the following.

int VV_SetXYZ(VisIVOViewer *env,float *X,float *Y, float *Z,char *names,int nRows)

X, Y and *Z* are three allocated pointers of the user code, each having nRows elements. These represent the points coordinates for data point display (options --x, --y and –z). The parameter *names* is a sequence of three labels for the system coordinates, nRows is the number of total points
Ex.: VV_SetXYZ(&env,x,y,z,"X Y Z",262144);

int VV_SetVect(VisIVOViewer *env,float *VX,float *VY, float *VZ,char *names,int nRows)
VX, VY and V*Z* are three allocated pointers of the user code, each having nRows elements. These represent the vectors components for vector display (options --vx, --vy and --vz). The parameter *names* is a sequence of three labels for the vector coordinates, nRows is the number of total points
Ex.: VV_SetVect(&env,vx,vy,vz,"VX VY VZ",262144);

int VV_SetVolume(VisIVOViewer *env,float *vol,char *name,char *geo,char *size)
vol is an allocated pointer of the user code representing a volume. These parameter *name* will be the label used in volumerendering or isosurface or slice representation (options --vrenderingfiled, --isosurfacefield, --slicefield). The parameter *geo* describe the geometry of the volume as a sequence of thre mesh-nodes on each dimension (options --compx, --compy and --compz). The parameter *size* describe the geometrical dimension of the cell (options --sizex --sizey and --sizez).
Ex.: VV_SetVolume(&env,vol,"density",”32 32 32”,”1.0 1.0 1.0”);

int VV_SetColorScalar(VisIVOViewer *env,float *color, char *name, int nRows)
color is an allocated pointer of the user code used as a scalar property for the palette (options --colorscalar).
Ex.: VV_SetColorScalar(&env,density,"density",262144);

int VV_SetRadiusScalar(VisIVOViewer *env,float *radiusscalar, char *name, int nRows)
int VV_SetHeightScalar(VisIVOViewer *env,float *heightscalar, char *name, int nRows)
radiusscalar and *heightscalar* are allocated pointer of the user code used as a scalar for radius and height in the glyphs data representation (options --radiusscalar--heightscalar).
Ex.: VV_SetRadiusScalar(&env,rad,"radius",262144);

VisIVO Viewer General Options

One or more of the following functions must be given to set the environment to generate images and movie frames both by using internal code arrays or by using an external VBT.

Environment Set

int VV_SetAtt(VisIVOImporter *env, int code, char *value)

This function will set some property of the environment.
Some specific calls must be given for particular tasks. In the following we will describe all the functions to full set the environment with some call examples.

Input File (used only for external VBT)
VV_SET_ FILEVBT
Option: nothing (***Mandatory only for external VBT***)
Call: **VV_SetAtt(&env,VV_SET_FILEVBT,"myvbt.bin");**

External File VBT
VV_SET_ EXTERNAL

Option: nothing
Call: **VV_SetAtt(&env,VV_SET_EXTERNAL,"");**
NOTE: remove any reference to internal arrays

Image name
VV_SET_OUT
Option: ***--out***
Call: **VV_SetAtt(&env, VV_SET_OUT,"VVImage");**
Note: set the sarting Id.

Defualt Images
VV_SET_DEFAULTIMAGES
Option: ***--nodefault***
Call: **VV_SetAtt(&env,VV_SET_DEFAULTIMAGES,"");**
Note: the --nodefault option is always set in the library. To force the default images generation this set must be given.

Camera position, focal point
VV_SET_CAMPOS, VV_SET_CAMFP
Option: ***--campos, --camfp***
Calls:
VV_SetAtt(&env,VV_SET_CAMPOS,"100 55 21.4");
VV_SetAtt(&env,VV_SET_CAMFP,"50 55 21.4");

Camera azimuth, elevation and zoom
VV_SET_CAMERA, VV_SET_AZIMUTH, VV_SET_ELEVATION, VV_SET_ZOOM
Option: ***--camazim, --camelev, --zoom***
Calls:
VV_SetAtt(&env,VV_SET_CAMERA,"10 35 1.4");
VV_SetAtt(&env,VV_SET_AZIMUTH,"10");
VV_SetAtt(&env,VV_SET_ELEVATION,"35");
VV_SetAtt(&env,VV_SET_ZOOM,"1.4");
Note: the camera can be set with the option VV_SET_CAMERA to set azimuth elevation and zoom with one call, or withs specific calls.

Image size
VV_SET_DEFAULTIMAGES
Option: ***--imagesize***
Call: **VV_SetAtt(&env,VV_SET_IMAGESIZE,"medium");**

Background color
VV_SET_BACKCOLOR
Option: ***--backcolor***
Call: **VV_SetAtt(&env,VV_SET_BACKCOLOR,"blue");**
Note: the parameter value may assume one of the following values : y*ellow, red, green, blue, white, black, cyan, violet.* Default value is black.

Foreground color
VV_SET_ONECOLOR
Option: ***--onecolor***
Call: **VV_SetAtt(&env,VV_SET_ONECOLOR,"cyan");**
Note: the parameter value may assume one of the following values : y*ellow, red, green, blue, white, black, cyan, violet.* Default value is white.

Enable the Palette
VV_SET_COLOR
Option: ***--color***
Call: **VV_SetAtt(&env,VV_SET_COLOR,"");**
Note: the parameter value is ignored.

Select the Palette
VV_SET_COLORTABLE
Option: ***--colortable***
Calls:
VV_SetAtt(&env,VV_SET_COLORTABLE,"/home/user/mypal.pal");
VV_SetAtt(&env,VV_SET_COLORTABLE,"temperature");
Note: the parameter can be or an user external palette or may assume one of the following pre-defined palettes : *default, default_step, efield, glow, gray, min_max, physics_contour, pure_red , pure_green, pure_blue, run1, run2 , sar, temperature, tensteps, volren_glow, volren_green, volren_rg, volren_twolevel,all_yellow,all_red,all_green,all_blue,all_white,all_black,all_cyan,all_violet*
Select the Palette range
VV_SET_COLORRANGE, VV_SET_COLORRANGEFROM, VV_SET_COLORRANGETO
Option: ***--colorrangefrom, --colorrangeto***
Calls:
VV_SetAtt(&env,VV_SET_COLORRANGE,"0 1000");
VV_SetAtt(&env,VV_SET_COLORRANGEFROM,"0");
VV_SetAtt(&env,VV_SET_COLORRANGETO,"1000");
Note: the palette range can be set with the option VV_SET_ COLORRANGE to set the range with one call, or withs specific calls to set the upper and lower limit. The default values of the upper and the lower limits are the maximum and minimum value of the field set with VV_SET_COLORSCALR or with the function VV_SetColorScalar above described.

Set the Stereo capability
VV_SET_STEREO
Option: ***--stereo***
Calls:
VV_SetAtt(&env,VV_SET_STEREO,"RedBlue");
Note: the value parameter can assume one of the following: *RedBlue*, *CrystalEyes*, *Anaglyph*. More details on VisIVO Vierer User Guide.

Select the anaglyph color saturation
VV_SET_ANAGLYPHSAT
Option: ***--anaglyphsat***
Calls:
VV_SetAtt(&env,VV_SET_ANAGLYPHSAT,"0.65");

Select the anaglyph color saturation
VV_SET_ANAGLYPHMASK
Option: ***--anaglyphmask***

Calls:
VV_SetAtt(&env,VV_SET_ANAGLYPHMASK,"4 3");

Set the visibility of the palette, the box and axes
VV_SET_BOX, VV_SET_AXES, VV_SET_PALETTE
Option: ***--showlut, --showbox, --showaxes***
Calls:
VV_SetAtt(&env,VV_SET_BOX,"");
VV_SetAtt(&env,VV_SET_AXES,"");
VV_SetAtt(&env,VV_SET_PALETTE,"");

Set camera for movie generation

int VV_SetCameraPath(VisIVOViewer *env,int type,float *camera,int zoomend,float *zsf,int framesec, int length, float *campos, float *camfp, float *camroll,int ftype)

This function set the camera path and the Viewer Operation will genearet a sequence of frames as here set by this function.
type is an integer value: 0, 1 or 2. See VisIVO Utils user guide for more details.
0 Create path for azimuth, elevation, zoom and roll. Default value.
1 Create path for azimut, elevation, zoom, focal point and roll
2 Create path for zoom, camera position, focal point and roll
3 Create path for azimuth,elevation,zoom, camera position, focal point and roll

It is suggested to use this last type to move the camera on azimuth, elevation and zoom only, putting fix the other values. See the VisIVO User Guide for more details

camera is a float array with six values for Azimuth, Elevation and Zoom, start and end:

zoomend = 1 the zooming is at the end
zsf Number of frame values for each second. Suggested value is 10.0
framesec Number of frame values for each second. Suggested value is 10 .
length Movie length in seconds.
campos is a float array with six values for camera position start and end. Ignore if type=1.
camfp is a float array with six values for camera position start and end.
camroll is a float array with two values for camera roll start and end.
ftype is an integer value: 0, 1 or 2.
0 The parameters of this call are append to a previous call.
1 The parameters of this call verride any other set of the camera path.
2 All parametrs are cleaned. No movie images will be produced

Note: *campos*, *camfp* and *camroll* can be set to NOSET_CAM. In this case VisIVO Viewer will maintain the last used setting (or the default setting).

Example:
```
float camera[6];
float camfp[6];
float campos[6];
float camroll[2];
camera[0]=0.0; //Azimuth start
```

```
camera[1]=70.0; //Azimuth End
camera[2]=0.0; //Elevation start
camera[3]=70.0; //Elevation End
camera[4]=1.0; //Zoom start
camera[5]=2.0; //Zoom end
int zoomend=1;
float zstepframe[1];
zstepframe[0]=0.1;
int framesec=10;
int length=10;
camfp[0]=35;
camfp[0]=35;
camfp[0]=35;
camfp[0]=35;
camfp[0]=35;
camfp[0]=70;
camroll[0]=NOSET_CAM;
camroll[1]=NOSET_CAM;

errorCode==VV_SetCycle(&env,1,camera, zoomend,zstepframe,framesec,length,campos,camfp,camroll);
```

Note: the function only set the camera path and parameters for thegeneration of the movie. The VV_Viewer function will create a sequence of images progressively numbered (see also VV_SET_CYCLEOFFSET) that must be mounted with external program (e.g. *convert*).

Set cycle offset
VV_SET_CYCLEOFFSET
Option: ***--cycleoffset***
Calls:
VV_SetAtt(&env,VV_SET_CYCLEOFFSET,"30000");
Note: It is the starting number of the generated images sequence for movie creation.

VV_SET_VO
Option: ***--VO*** *(Mandatory in case of gLite grid catalogue usage)*
Call: **VV_SetAtt(&env, VV_SET_VO,"alice");**
Note: in the value filed must be specified the Virtual Organization used on gLite.

VV_SET_LFNOUT
Option: ***--lfnout***
Call: **VV_SetAtt(&env, VV_SET_LFNOUT,"lfn://grid/cometa/user/myImage");**
Note: it is used to set the logical filename (lfn) when running on gLite grid. A logical filename starts with lfn://.

VV_SET_SE
Option: ***--se***
Call: **VV_SetAtt(&env, VV_SET_SE,"grid-se-01.ct.inaf.it");**
Note: it is used to set the storage element.

VisIVO Viewer Specific Options

Datapoints

One or more of the following functions must be given to set the environment to generate images and movie frames with data points.

Set scale
VV_SET_SCALE
Option: ***--scale***
Calls:
VV_SetAtt(&env,VV_SET_SCALE,"");

Set logarithm scale for the palette
VV_SET_LOGSCALE
Option: ***--logscale***
Calls:
VV_SetAtt(&env,VV_SET_LOGSCALE,"");

Set glyphs
VV_SET_GLYPHS
Option: ***--glyphs***
Calls:
VV_SetAtt(&env,VV_SET_GLYPHS,"sphere");
Note: The value parameter can assume one of the folowing: *pixel sphere cone cylinder cube*. Scaled glyphs properties are described at the beginning of this section.

Set radius and height
VV_SET_RADIUS, VV_SET_HEIGHT
Option: ***--radius, --height***
Calls:
VV_SetAtt(&env,VV_SET_RADIUS,"2.0");
VV_SetAtt(&env,VV_SET_HEIGHT,"5");
Note: The value parameter *represent the radius/height* for geometrical forms.

Set points opacity
VV_SET_OPACITY
Option: ***--opacity***
Calls:
VV_SetAtt(&env,VV_SET_OPACITY,"0.65");

Volumes

One or more of the following functions must be given to set the environment to generate images and movie frames with volums.

Set volume representation
VV_SET_VOLUME
Option: ***--volume***
Calls:
VV_SetAtt(&env,VV_SET_VOLUME,"");
Note: This attribute **must** be set to represent volumes. The volume rendering, isosurface and slice fields are set by using the functions described at the beginning of this section.

Set volumerenfering representation
VV_SET_VOLUMERENDERING
Option: **-**
Calls:
VV_SetAtt(&env,VV_SET_VOLUMERENDERING,"");
Note: this attribute enables the volume rendering represenation

Set volume shadow representation
VV_SET_SHADOW
Option: ***--shadow***
Calls:
VV_SetAtt(&env,VV_SET_SHADOW,"");

Set volume isosurface representation
VV_SET_ISOSURFACE
Option: ***--isosurface***
Calls:
VV_SetAtt(&env,VV_SET_ISOSURFACE,"");
Note: This attribute **must** be set to represent volume isosurface.

Set isosurface value representation
VV_SET_ISOSURFACEVALUE
Option: ***--isosurfacevalue***
Calls:
VV_SetAtt(&env,VV_SET_ISOSURFACE,"125");

Set isosurface value representation with wireframes
VV_SET_WIREFRAME
Option: ***--wireframe***
Calls:
VV_SetAtt(&env,VV_SET_WIREFRAME,"");

Set smoothed isosurface representation
VV_SET_ISOSMOOTH
Option: ***--isosmooth***
Calls:
VV_SetAtt(&env,VV_SET_ISOSMOOTH,"high");
Note: The value may be *medium* or *high*

Set volume slice representation
VV_SET_SLICE
Option: ***--slice***
Calls:
VV_SetAtt(&env,VV_SET_SLICE,"");
Note: This attribute **must** be set to represent volume slice.

Set the orthonormal plane of slice
VV_SET_SLICEPLANE
Option: ***--sliceplane***
Calls:
VV_SetAtt(&env,VV_SET_SLICEPLANE,"x");

Note: This attribute may assume one of the following: *x*, *y* or *z*.

Set the orthonormal plane position in the volume.
VV_SET_SLICEPOS
Option: ***--sliceposition***
Calls:
VV_SetAtt(&env,VV_SET_SLICEPOS,"32");
Note: This attribute may assume one value inside the grid mesh dimension. For a volume of 64x32x32 and slicelplane *x*, sliceposition is in the range 0-63, if sliceplane is *y* or *z*, sliceposition is in the range 0-31.

Set the generic plane of slice: point position
VV_SET_SLICEPLANEPOINT
Option: ***--sliceplanepoint***
Calls:
VV_SetAtt(&env,VV_SET_SLICEPLANEPOINT,"32 32 32");
Note: the three coordinates of a point of the plane

Set the generic plane of slice: normal axes
VV_SET_SLICEPLANENORMAL
Option: ***--sliceplanenormal***
Calls:
VV_SetAtt(&env,VV_SET_SLICEPLANENORMAL,"64 64 64");
Note: the three coordinate fixing a point belonging to the normal axes to the slice. The sliceplanepoint and the sliceplanenormal fix to pints and an axes in the space. The slice is normal to this axes and the point in sliceplanepoint is a point of this plane.

Orthonormal Scan Slice

Cycle can be given for Orthogonal Normal planes (x, y or z). In this case the cycle file must contain a sequence of integers (one for each row) inside the volume range (e.g 0-64).

Set camera for movie generation

int VV_SetOSliceScan(VisIVOViewer *env,int *slice)

This function set the slices and the Viewer Operation will generate a sequence of frames containing orthonormal slices.

Slice is an int array with four values. The first two values are for the slice position from-to in the volume. Values outsides the volume size are ignored. The third value is for selecting the parallel plane: 1=x plane, 2= y plane and 3=zplane. The last value is the step increment for slice position.

Note: the function only set the slices and parameters for the movie generation. The VV_Viewer function will create a sequence of images progressively numbered (see also VV_SET_CYCLEOFFSET) that must be mounted with external program (e.g. *convert*).

Example:
```
int oslice[4];
oslice[0]=0; // from 0
oslice[1]=63; // to 64
oslice[2]=0; // slice paralell to the x plane
```

oslice[3]=1; // step 1: 64 slices will be generated. The slices are parallel to x axes, at the positions 0, 1... 63 mesh points

errorCode=VV_SetOSliceScan(&env,oslice);

Generic Scan Slice

Cycle can be given for point-planenormal slice. In this case is recommended to set the **VV_SET_BOX** attribute.

Set camera for movie generation. This function move the plane point along the normal axes.

int VV_SetGSliceScan(VisIVOViewer *env,float *fgslice, int *iglsice)

This function set the generic slices and the Viewer Operation will generate a sequence of frames containing orthonormal slices.

fgslice is a float array with seven values. The first three values fix the points belonging to the slice and the next three values fix the point belonging to the normal axes to the slice. The last value fix increment (or decrement) of the point coordinates.

Iglsice is an int pointer with two values. The first is for the number of slices. The second is for the direction: if it is equal 0 the plane point coordinates are increased; if it is equal to 1 the plane point coordinates are decreased.

The plane point is moved along the normal axis. The product *fgslice[6]*igslice[0]* determine the movement of the plain point: if the product is equal to 1, at the end the plane point will be at the same point of the normal point.

Note: the function only set the slices and parameters for the movie generation. The VV_Viewer function will create a sequence of images progressively numbered (see also VV_SET_CYCLEOFFSET) that must be mounted with external program (e.g. *convert*).

Example:
ifloat fgslice[7]; //3 for point, 3 for normal, 1 for step_size
int igslice[2]; // step, 0-1 Suggested: fgslice[6]*igslice[0]=1

```
fgslice[0]=0;
fgslice[1]=0;
fgslice[2]=0;

fgslice[3]=64;
fgslice[4]=64;
fgslice[5]=64;

fgslice[6]=0.1;

igslice[0]=10;
igslice[1]=0;
errorCode=VV_SetGSliceScan(&env,fgslice,igslice);
```

Vectors

One or more of the following functions must be given to set the environment to generate images and movie frames with vectors.

Set vectors representation
VV_SET_VOLUME
Option: ***--vector***
Call:
VV_SetAtt(&env,VV_SET_VECTOR,"");
Note: This attribute **must** be set to represent vectors. The applications points and the vectors components are set by using the functions described at the beginning of this section.

Set vectors representation with lines
VV_SET_VECTORLINE
Option: ***--vectorline***
Call:
VV_SetAtt(&env,VV_SET_VECTORLINE,"");

Set vectors scale factor
VV_SET_**VECTORSCALINGFACTOR**
Option: ***--vectorscalefactor***
Call:
VV_SetAtt(&env,VV_SET_VECTORSCALINGFACTOR,"1.3");

Set vectors scale
VV_SET_**VECTORSCALE**
Option: ***--vectorscale***
Call:
VV_SetAtt(&env,VV_SET_VECTORSCALE,"0");
Note: The value parameter may assume one of the following values: **0** - the scale of the vector dimension is given by the active scalar (colorscalar option); **1** - the scale of the vector dimension is given by the vector magnitude; **-1** - the vectors are not scaled (default).

VisIVO Viewer Splotch Specific Options

One or more of the following functions must be given to set the environment to generate images and movie frames with splotch. The field for Gas Intensity, Gas Color and Smoothing length are set with the functions described at the beginning of this section.

Set Splotch representation
Option: ***--splotch***
Call:
int VV_EnableSplotch(VisIVOViewer *env)
Note: This function **must** be call to enable Splotch representation.

Set Splotch parameters
See the VisIVO Viewer User Guide for a detailed description of the followin attributes:
Calls:

VV_SetAtt(&env,vVV_SET_SPL_INTENSITYLOG0,”TRUE”)

iVV_SetAtt(&env,VV_SET_SPL_INTENSITYMIN0,”100”)
VV_SetAtt(&env,VV_SET_SPL_INTENSITYMAX0,”400”)
VV_SetAtt(&env,VV_SET_SPL_SIZEFIX0,”1.0”)
VV_SetAtt(&env,VV_SET_SPL_SIZEFAC0,”1.0”)
VV_SetAtt(&env,VV_SET_SPL_COLORISVECTOR0,”TRUE”)
VV_SetAtt(&env,VV_SET_SPL_XRES,”800”)
VV_SetAtt(&env,VV_SET_SPL_YRES,”800”)
VV_SetAtt(&env,VV_SET_SPL_FOV,”10.0”)
VV_SetAtt(&env,VV_SET_SPL_GRAYABSORBTION,”0.2”)
VV_SetAtt(&env,VV_SET_SPL_BRIGHTNESS0,”1”)
VV_SetAtt(&env,VV_SET_SPL_COLORLOG0,”TRUE”)
VV_SetAtt(&env,VV_SET_SPL_COLORASINH0,”TRUE”)
VV_SetAtt(&env,VV_SET_SPL_ROLSKY_X,"0");
VV_SetAtt(&env,VV_SET_SPL_ROLSKY_Y,"1");
VV_SetAtt(&env,VV_SET_SPL_ROLSKY_Z,"0");

Viewer Operation

int VV_View(VisIVOViewer *env)

This function execute the viewer operation with the options set by the env variable: it generates images in png format

Environment Close

int VV_Clean(VisIVOViewer *env)

This call execute delete all options set by in the env variable, and delete the temporary files created by the above functions.

Returning Values

The above described functions return one of the following error code.

noError	**0**
invalidParCode	**1**
invalidVBT	**3**
invalidPathCreation	**4**
invalidPointers	**6**
invalidFilterOptions	**7**
invalidFilterOperation	**8**
invalidInternalData	**11**
invalidImage	**12**

Examples

User programs that want to use the library, must include visivo.h.

Visualization of verly large unstructred points: external ascii data. Generation of ann image and movie (sequence of frames)

```
#include “visivo.h”
…
```

```
int main(int argc, char*argv[])
{

VisIVOImporter env1;
VisIVOFilter env2;
VisIVOViewer env3;
int errorCode;
char op[256];
char vbt[256];
char ranvbt[256];
float rperc=1.0;
char sperc[10];
sprintf(sperc,"%f",rperc);



strcpy(vbt,"VBT1.bin");
strcpy(ranvbt,"RANVBT1.bin");

errorCode=VI_Init(&env1);

errorCode=VI_SetAtt(&env1,VI_SET_FFORMAT,"ascii");
errorCode=VI_SetAtt(&env1,VI_SET_FILEPATH,"ASCIILargeUserFile");
errorCode=VI_SetAtt(&env1,VI_SET_OUTFILEVBT,vbt);
errorCode=VI_Import(&env1);

strcpy(op,"randomizer");
errorCode=VF_Init(&env2);
errorCode=VF_SetAtt(&env2,VF_SET_OPERATION,op);
errorCode=VF_SetAtt(&env2,VF_SET_FILEVBT,vbt);
errorCode=VF_SetAtt(&env2, VF_SET_RANDOMPERC,sperc);
errorCode=VF_SetAtt(&env2, VF_SET_OUTVBT,ranvbt);
errorCode=VF_Filter(&env2);

VV_Init(&env3);
errorCode=VV_SetAtt(&env3,VV_SET_FILEVBT,ranvby);
errorCode=VV_SetAtt(&env3,VV_SET_FIELD,"X Y Z");  // X Y Z in the VBT file
errorCode=VV_SetAtt(&env3,VV_SET_COLOR,"");
errorCode=VV_SetAtt(&env3,VV_SET_COLORSCALAR,"density"); //density is a field of the VBT
errorCode=VV_SetAtt(&env3,VV_SET_CAMERA,"10 35 1.4");
errorCode=VV_SetAtt(&env3,VV_SET_BOX,"");
errorCode=VV_SetAtt(&env3,VV_SET_AXES,"");
errorCode=VV_SetAtt(&env3,VV_SET_PALETTE,"");
errorCode=VV_SetAtt(&env3,VV_SET_OUT,"VVImage"); //VVImage.png will be generated

errorCode=VF_Viewer(&env3);  // image generation

float camera[6];
camera[0]=0.0; //Azimuth start
camera[1]=70.0; //Azimuth End
camera[2]=0.0; //Elevation start
camera[3]=70.0; //Elevation End
camera[4]=1.0; //Zoom start
camera[5]=2.0; //Zoom end
int zoomend=1;
float zstepframe[1];
zstepframe[0]=0.1;
int framesec=5;
```

```
int length=5;

errorCode==VV_SetCycle(&env3,camera, zoomend,zstepframe,framesec,length);
errorCode=VV_SetAtt(&env3,VV_SET_OUT,"MovieImg"); //MovieImg###.png will be generated

errorCode=VV_View(&env3); // Sequence of images generation
.....
VI_Clean(&env1);
VF_Clean(&env2);
VV_Clean(&env3);


...
}
```

Visualization of internal volume. Generation of images (volumerendering isosurface) and scan movie (sequence of slices)

```
#include "visivo.h"
…
int main(int argc, char*argv[])
{

VisIVOViewer env3;
VV_init(&env3);

float vol[262144]; / a volume dimension with 64x64x64 mesh points
…..
errorCode=VV_SetVolume(&env3,vol,"myDens","64 64 64","1 1 1");
errorCode=VV_SetAtt(&env3,VV_SET_VOLUME,"");
errorCode=VV_SetAtt(&env3,VV_SET_BOX,"");
errorCode=VV_SetAtt(&env3,VV_SET_AXES,"");
errorCode=VV_SetAtt(&env3,VV_SET_PALETTE,"");
errorCode=VV_SetAtt(&env3,VV_SET_OUT,"VolRenImage");
errorCode=VV_SetAtt(&env3,VV_SET_CAMERA,"10 35 1.4");

VV_View(&env3);

errorCode=VV_SetAtt(&env3,VV_SET_ISOSURFACE,"");
errorCode=VV_SetAtt(&env3,VV_SET_OUT,"VolIsoImage");

VV_View(&env3);

int oslice[4];
oslice[0]=0;
oslice[1]=64;
oslice[2]=0;
oslice[3]=1;
errorCode=VV_SetOSliceScan(&env3,oslice);
errorCode=VV_SetAtt(&env3,VV_SET_SLICE,"");
errorCode=VV_SetAtt(&env3,VV_SET_SLICEPLANE,"x");
errorCode=VV_SetAtt(&env3,VV_SET_OUT,"SliceScanImage");

VV_View(&env3);

VV_Clean(&env3);

return 0;
```

```
}
```

VisIVO Library General Utilities

The following functions cane be used everywhere in the user program to have informations on external VBTs or to produce images directly from user file in a simplified way.

int VS_VBTMetaData(VBT *tab, int satistic ,char *value)

VBT is a structred variables defined in visivoserver.h as follow

```
struct VBT
{
   char locator[256];
 char datatype[20];
  int nOfFields;
  unsigned long long int nOfRows;
  int nCells[3];
  float cellSize[3];
  char endianity[20];
  char **field;
  float **statistic; //max, min, avg, sum
};
```

The function open the VBT given in the *value* parameter and fill the above structure as follows:
locator: the name of the VBT (tha same of the *value* parameter)
datatype: the VBT datatype (e.g. float)
nOfFields: number of fields.
NOfRows. number of Rows
nCells: in case of volume the number of mesh points in each direction.
cellSize:in case of volume the dimension of each cell
endianity: the VBT endianism (e.g. *little*)
field: the field names as reported in the VBT
statistic: max, min, avg and sum of each field. These values are given if the *statistic* parameter is equal to 1.

int VS_VBTPointers(VBT *tab, unsigned int *colList,unsigned int nOfCols ,long unsigned int fromRow, long unsigned int toRow, float **fArray)

This function return pointers to the VBT field values. The colList is a list of number of columns the user need to read (first column is 0), nOfCols is the length of colList, fromRow and toRow are the first and last row the user want to read: 0 is the first row (fromRow >=0) tab.nOfRows-1 is the last row. (toRow<=tab.nOfRows-1). fArray is an allocatetd array: first dimension is equal to nOfCols, the second dimension is *toRow-fromRow+1*

Example:

```
{
….
VBT tab;

VS_VBTMetaData(&tab, 1 ,”userVBT.bin”)

int numberOfFields=tab.nOfFields;
float **fArray;
fArray=(float **) calloc(numberOfFields,sizeof(float*));

for(int i=0;i< numberOfFields;i++)
        fArray[i]=(float *) calloc(mytable.nOfRows,sizeof(float));

unsigned long long int *colSet;
colSet=(unsigned long long int *) calloc( numberOfFields,sizeof(unsigned long long int));
for(int i=0;i< numberOfFields;i++)
        colSet[i]=i;
VS_VBTPointers(&tab, colSet,numberOfFields ,0, tab.nOfRows-1,fArray)
….
}
```

fArray will contain all the VBT data

int VS_VBTAllColumn(VBT *tab, int idCol ,float* oneCol)

This function return a pointer to a VBT field values. idCol is a the number of column (first column is 0), oneCo is an allocatetd array with the dimension tab.nOfRows

Example:

```
{
….
VBT tab;
VS_VBTMetaData(&tab, 1 ,”userVBT.bin”)
….
float *column;
column=(float *) calloc(tab.nOfRows,sizeof(float));
VS_VBTAllColumn(&tab, 0,column)
….
}
```

column will contain all the first filed of the VBT

int VS_VBTPartialColumn(VBT *tab, int idCol ,float* oneCol,int from, int to)

The function is very similar to the above funtion, but allow to read only a portion of the column: 0 is the first element (fromRow >=0) tab.nOfRows-1 is the last element
Example:

```
{
….
VBT tab;
VS_VBTMetaData(&tab, 1 ,”userVBT.bin”)
….
float *column;
column=(float *) calloc(100,sizeof(float));
VS_VBTPartialColumn(&tab, 0,column, 0, 99)
….
}
```

column will contain the 100 values of the first filed of the VBT

int VS_DirectImage(VisIVOViewer *VVenv,VisIVOImporter *VIenv,float random)

This function allow to produce directly images from user data. This method represent the simplest way to obtain images from user data.
The user data format must be in a supported data format. The environment varaibles for viewer and for the importer must be given. No filters operation can be applied with this method but a randomization to visualize huge dataset is provided.
All the environment variable set (as above described) can be given to obtain images (or movie) from user data.

```
{
….
// direct image of a data in a file
char filename[256];
VisIVOImporter  myVIstruct;
VisIVOViewer  myVVstruct;
strcpy(filename,"UserAscii.txt");
errorCode=VI_Init(&myVIstruct);
errorCode=VI_SetAtt(&myVIstruct,VI_SET_FFORMAT,"ascii");
errorCode=VI_SetAtt(&myVIstruct,VI_SET_FILEPATH,filename);
errorCode=VV_Init(&myVVstruct);
errorCode=VV_SetAtt(&myVVstruct,VV_SET_COLOR,"");
errorCode=VV_SetAtt(&myVVstruct,VV_SET_COLORSCALAR,"density");
//density  in the file UserAscii.txt

errorCode=VV_SetAtt(&myVVstruct,VV_SET_BOX,"");
errorCode=VV_SetAtt(&myVVstruct,VV_SET_AXES,"");
errorCode=VV_SetAtt(&myVVstruct,VV_SET_PALETTE,"");
errorCode=VV_SetAtt(&myVVstruct,VV_SET_OUT,"MovieImage");

float camera[6];
```

```
camera[0]=0.0; //Azimuth start
camera[1]=70.0; //Azimuth end
camera[2]=0.0; //Elevation start
camera[3]=70.0; //Elevation end
camera[4]=1.0; //Zoom start
camera[5]=2.0; //Zoom end
int zoomend=1;
float zstepframe[1];
zstepframe[0]=0.1;
int framesec=5;
int length=2;

errorCode==VV_SetCameraPath(&myVVstruct,&camera[0],
zoomend,zstepframe,framesec,length);



errorCode=VV_SetAtt(&myVVstruct,VV_SET_FIELD,"X Y Z");  // X Y Z in the file UserAscii.txt

float random=10.0; //radom percentile: between 100.0 (full data) 0.0(nothing to do)
errorCode=VS_DirectImage(&myVVstruct,&myVIstruct,random);
// produces the movie  frames
…..
}
```

VisIVO Api Multithread

The above functions are blocking and these return when finished. The functions that set environment attribute are very low time consuming. But the real operation performed by VI_Import, VF_Filter, VV_View and VS_DirectImage could last several minutes or hours. To avoid the calling program to wait for it, we have developed asynchronous calls that fork new processes or new threads. On Windows systems the asynchronus functions, even if these can be called, are always blocking calls.

It is defined a new type **VisIVOAsynchId** that contain all information to start a new thread or a new process. The program must define a variable :

VisIVOAsynchId id;

the id variable will contain all information, it will be used to start a thread or a new process and will be tested waiting for completion. Each new process or thread is associated to a variable. To use multithread or processes it is necessary to define multiple VisIVOAsynchId variables.

The variable must be initialized with the function

void VA_Init(VisIVOAsynchId *id)

The Importer and Filter operation always use the multithread. The multiprocess is used only for the View operation.
The id variable also contain the information if mutithread or multiprocess will be used. By default we adopt the multiprocess. To force the multithread or multiprocess the following functions mus be called.

void VA_SetMultiThread(VisIVOAsynchId *id)
void VA_SetMultiProc(VisIVOAsynchId *id)

When in the View the multiprocess is set, the user program can call several instance for VA_Importer, VA_Filter, VA_View and several threads or processes will be started simultaneously. In case of View multithread only the first call is started as a thread, the other threads will be queued: only one new single thread will run on the system, the others will wait for completion.

NOTE: for MAC platform the View multiprocesses mode cannot be activated. Always the View multithread is active.

int VA_Importer(VisIVOImporter *env, VisIVOAsynchId *id)
int VA_Filter(VisIVOFilter *env, VisIVOAsynchId *id)
int VA_View(VisIVOViewer *env, VisIVOAsynchId *id)

int VA_DirectImage(VisIVOViewer *env1,VisIVOImporter *env2,float random, VisIVOAsynchId *id)

start a new thread or a new process for the View.

The following functions can be used to test for completion:

int VA_GetState(VisIVOAsynchId *id)

Return: **successfulEndThread** if the thread/process is completed, **runningThread** if the thread/process is running; **errorThread** in case of error, **undefined** in case of id was not yet used in asynchronous functions

int VA_Wait(VisIVOAsynchId *id)
This a blocking call and wait for thread/process completion. It is necessary to call the function before ending the user program.

Return: **noError** if the thread/process is finished, **errorThread** The implementation has detected that the value specified by *thread* does not refer to a joinable thread;

int VA_GetError(VisIVOAsynchId *id)
Return the same error code of VI_Import, VF_Filter and VV_View when called by the corresponding VA functions

Returning Values

The above described functions return one of the following error code.

noError	**0**
invalidParCode	**1**
invalidVBT	**3**
invalidPathCreation	**4**
invalidPointers	**6**
invalidFilterOptions	**7**
invalidFilterOperation	**8**
invalidInternalData	**11**
invalidImage	**12**
runningThread	**13**
errorThread	**14**
successfulEndThread	**15**
invaldInputFile	**16**
undefined	**17**

VisIVO Library
Examples

Image from user file

User programs that want to use the library, must include visivo.h.
To obtain images from an existing file the following statements can be used in the code:

```
#include "visivo.h"
…
int main(int argc, char*argv[])
{

// environment variables
VisIVOViewer myVVstruct;
VisIVOImporter myVIstruct;
VisIVOFilter myVFstruct;
VBT mytable;
int errorCode;
```

// Start the importer to import the user file in a VBT

```
errorCode=VI_Init(&myVIstruct);  //initialize the importer environment

//set the environment
errorCode=VI_SetAtt(&myVIstruct,VI_SET_FFORMAT,"ascii");
errorCode=VI_SetAtt(&myVIstruct,VI_SET_FILEPATH,"myasciifile.txt");
errorCode=VI_SetAtt(&myVIstruct,VI_SET_OUTFILEVBT,"myvbt.bin");

// perform the import with this environment and create the myvbt VBT table
errorCode=VI_Import(&myVIstruct);
```

// Start the filter (optional) i.e. randomize myvbt.bin

```
errorCode=VF_Init(&myVFstruct);  //initialize the filter environment

//set the filter environment
errorCode=VF_SetAtt(&myVFstruct,VF_SET_OPERATION,"randomizer");
errorCode=VF_SetAtt(&myVFstruct,VF_SET_FILEVBT,"myvbt.bin");
errorCode=VF_SetAtt(&myVFstruct,VF_SET_RANDOMPERC,"10.0");

errorCode=VF_SetAtt(&myVFstruct,VF_SET_OUTVBT,"myrandvbt.bin"); //output new VBT

errorCode=VF_Filter(&myVFstruct);  // perform the filter for the environment
```

//Read table data (optional)

```
errorCode=VS_VBTMetaData(&mytable, 1 ,"myrandvbt.bin");
```

// Start the Viewer

```
errorCode=VV_Init(&myVVstruct); //initialize the filter environment

// set the environment varaiable
```

```
errorCode=VV_SetAtt(&myVVstruct,VV_SET_FILEVBT,"myrandvbt .bin");
errorCode=VV_SetAtt(&myVVstruct,VV_SET_FIELD,"X Y Z");  // X Y Z fields in the VBT
errorCode=VV_SetAtt(&myVVstruct,VV_SET_COLOR,"");
errorCode=VV_SetAtt(&myVVstruct,VV_SET_BOX,"");
errorCode=VV_SetAtt(&myVVstruct,VV_SET_AXES,"");
errorCode=VV_SetAtt(&myVVstruct,VV_SET_PALETTE,"");
errorCode=VV_SetAtt(&myVVstruct,VV_SET_COLORSCALAR,"density"); //density field in the VBT
errorCode=VV_SetAtt(&myVVstruct,VV_SET_OUT,"VVImage1");
errorCode=VV_SetAtt(&myVVstruct,VV_SET_CAMERA,"10 35 1.4");

//create the image
errorCode=VV_View(&myVVstruct); //this creates VVImage1.png

errorCode=VV_SetAtt(&myVVstruct,VV_SET_OUT,"VVImage2");
errorCode=VV_SetAtt(&myVVstruct,VV_SET_CAMERA,"45 45 1.0");

errorCode=VV_View(&myVVstruct); //this creates VVImage2.png

erroCode=VI_Clean(&myVIstruct);
erroCode=VF_Clean(&myVFstruct);
erroCode=VV_Clean(&myVVstruct);
```

Image from user file by using Multithread

User programs that want to use the library, must include visivo.h.
To obtain images from an existing file the following statements can be used in the code:

```
#include "visivo.h"
…
int main(int argc, char*argv[])
{

// environment variables
VisIVOViewer myVVstruct;
VisIVOImporter myVIstruct;
VisIVOFilter myVFstruct;
VisIVOAsinchId id1;
VisIVOAsinchId id2;
VisIVOAsinchId id3;
VBT mytable;
int errorCode;

VA_init(&id1)
VA_init(&id2)
VA_init(&id3)
```

// Start the importer to import the user file in a VBT

```
errorCode=VI_Init(&myVIstruct);  //initialize the importer environment

//set the environment
errorCode=VI_SetAtt(&myVIstruct,VI_SET_FFORMAT,"ascii");
errorCode=VI_SetAtt(&myVIstruct,VI_SET_FILEPATH,"myasciifile.txt");
errorCode=VI_SetAtt(&myVIstruct,VI_SET_OUTFILEVBT,"myvbt.bin");

// perform the import with this environment and create the myvbt VBT table
```

```
errorCode=VA_Import(&myVIstruct,&id1);

….. //doing other work
```

// Start the filter (optional) i.e. randomize myvbt.bin

```
errorCode=VF_Init(&myVFstruct);  //initialize the filter environment

//set the filter environment
errorCode=VF_SetAtt(&myVFstruct,VF_SET_OPERATION,”randomizer”);
errorCode=VF_SetAtt(&myVFstruct,VF_SET_FILEVBT,"myvbt.bin");
errorCode=VF_SetAtt(&myVFstruct,VF_SET_RANDOMPERC,"10.0");

errorCode=VF_SetAtt(&myVFstruct,VF_SET_OUTVBT,"myrandvbt.bin"); //output new VBT

errorVode=VA_Wait(&id1); // before run the filter wait for importer completion

errorCode=VA_Filter(&myVFstruct,&id2);  // perform the filter for the environment

….. //doing other work
```

//Read table data (optional)

```
errorCode=VS_VBTMetaData(&mytable, 1 ,"myrandvbt.bin");
```

// Start the Viewer

```
errorCode=VV_Init(&myVVstruct); //initialize the filter environment

// set the environment varaiable
errorCode=VV_SetAtt(&myVVstruct,VV_SET_FILEVBT,"myrandvbt .bin");
errorCode=VV_SetAtt(&myVVstruct,VV_SET_FIELD,"X Y Z");  // X Y Z fields in the VBT
errorCode=VV_SetAtt(&myVVstruct,VV_SET_COLOR,"");
errorCode=VV_SetAtt(&myVVstruct,VV_SET_BOX,"");
errorCode=VV_SetAtt(&myVVstruct,VV_SET_AXES,"");
errorCode=VV_SetAtt(&myVVstruct,VV_SET_PALETTE,"");
errorCode=VV_SetAtt(&myVVstruct,VV_SET_COLORSCALAR,"density"); //density field in the VBT
errorCode=VV_SetAtt(&myVVstruct,VV_SET_OUT,"VVImage1");
errorCode=VV_SetAtt(&myVVstruct,VV_SET_CAMERA,"10 35 1.4");

//create the image
errorCode=VV_View(&myVVstruct); //this creates VVImage1.png

errorCode=VV_SetAtt(&myVVstruct,VV_SET_OUT,"VVImage2");
errorCode=VV_SetAtt(&myVVstruct,VV_SET_CAMERA,"45 45 1.0");

errorVode=VA_Wait(&id2); // before run the viewer wait for filter completion

errorCode=VA_View(&myVVstruct,&id3); //this creates VVImage2.png

….. //doing other work

errorVode=VA_Wait(&id3); // waith for  viewer completion before the end

erroCode=VI_Clean(&myVIstruct);
erroCode=VF_Clean(&myVFstruct);
erroCode=VV_Clean(&myVVstruct);
```

Image with Generic Functions

The image could be directly created by the input file without create the VBT using the generic functions.

```
#include "visivo.h"
…
int main(int argc, char*argv[])
{

// environment variables
VisIVOViewer myVVstruct;
VisIVOImporter myVIstruct;

int errorCode;

errorCode=VI_Init(&myVIstruct);
errorCode=VI_SetAtt(&myVIstruct,VI_SET_FFORMAT,"ascii");
errorCode=VI_SetAtt(&myVIstruct,VI_SET_FILEPATH,""myasciifile.txt");


errorCode=VV_Init(&myVVstruct);
errorCode=VV_SetAtt(&myVVstruct,VV_SET_FIELD,"X Y Z");  // X Y Z in the file
errorCode=VV_SetAtt(&myVVstruct,VV_SET_COLOR,"");
errorCode=VV_SetAtt(&myVVstruct,VV_SET_COLORSCALAR,"density");
errorCode=VV_SetAtt(&myVVstruct,VV_SET_BOX,"");
errorCode=VV_SetAtt(&myVVstruct,VV_SET_AXES,"");
errorCode=VV_SetAtt(&myVVstruct,VV_SET_PALETTE,"");
errorCode=VV_SetAtt(&myVVstruct,VV_SET_OUT,"VVImage1");
errorCode=VV_SetAtt(&myVVstruct,VV_SET_CAMERA,"10 35 1.4");

float random=10.0; //radom percentile

errorCode=VS_DirectImage(&myVVstruct,&myVIstruct,random);

errorCode=VV_SetAtt(&myVVstruct,VV_SET_OUT,"VVImage2");
errorCode=VV_SetAtt(&myVVstruct,VV_SET_CAMERA,"10 35 1.4");

errorCode=VS_DirectImage(&myVVstruct,&myVIstruct,random);

// the above function call produces VVImage1.png and VVImage2.png   with the set given for camera and
// colours, using X, Y, Z and density columns in the ascii file that is randomized at 10%..

erroCode=VI_Clean(&myVIstruct);
erroCode=VV_Clean(&myVVstruct);
```

Image from internal arrays

To obtain images from internal arrays the following statements can be used in the code:

```
#include "visivo.h"
…
int main(int argc, char*argv[])
```

```
{

// environment variables
VisIVOViewer myVVstruct;

int errorCode;
float *X, *Y, *Z, *density;

X= (float *) calloc(nbodies,sizeof(float));
Y= (float *) calloc(nbodies,sizeof(float));
Z= (float *) calloc(nbodies,sizeof(float));
density= (float *) calloc(nbodies,sizeof(float));
```

// Start the Viewer

```
errorCode=VV_Init(&myVVstruct); //initialize the filter environment

// set the environment varaiable
errorCode=VV_SetXYZ(&myVVstruct,X,Y,Z,"Xlabel Ylabel Zlabel",nbodies);  // X Y Z in local RAM
errorCode=VV_SetColorScalar(&myVVstruct,density,"Densitylabel",nbodies);
errorCode=VV_SetAtt(&myVVstruct,VV_SET_COLOR,"");
errorCode=VV_SetAtt(&myVVstruct,VV_SET_BOX,"");
errorCode=VV_SetAtt(&myVVstruct,VV_SET_AXES,"");
errorCode=VV_SetAtt(&myVVstruct,VV_SET_PALETTE,"");
errorCode=VV_SetAtt(&myVVstruct,VV_SET_CAMERA,"10 35 1.4");

errorCode=VV_SetAtt(&myVVstruct,VV_SET_OUT,"VVImage1");

//create the image VVImage1.png
errorCode=VV_View(&myVVstruct);

errorCode=VV_SetAtt(&myVVstruct,VV_SET_CAMERA,"10 35 1.4");

errorCode=VV_SetAtt(&myVVstruct,VV_SET_OUT,"VVImage2");

//create the image VVImage2.png
errorCode=VV_View(&myVVstruct);

erroCode=VV_Clean(&myVVstruct);
```